Tei-Wei Kuo Edwin Sha
Minyi Guo Laurence T. Yang
Zili Shao (Eds.)

LNCS 4808

Embedded and Ubiquitous Computing

International Conference, EUC 2007
Taipei, Taiwan, December 2007
Proceedings

ifip

Lecture Notes in Computer Science 4808

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Tei-Wei Kuo Edwin Sha Minyi Guo
Laurence T. Yang Zili Shao (Eds.)

Embedded and Ubiquitous Computing

International Conference, EUC 2007
Taipei, Taiwan, December 17-20, 2007
Proceedings

Springer

Volume Editors

Tei-Wei Kuo
National Taiwan University
Taiwan 106, Republic of China
E-mail: ktw@csie.ntu.edu.tw

Edwin Sha
University of Texas at Dallas
Richardson, TX 75083-0688, USA
E-mail: edsha@utdallas.edu

Minyi Guo
The University of Aizu
Aizu-Wakamatsu City, Japan
E-mail: minyi@u-aizu.ac.jp

Laurence T. Yang
St Francis Xavier University
Antigonish, NS, B2G 2W5, Canada
E-mail: ltyang@gmail.com

Zili Shao
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
E-mail: cszlshao@comp.polyu.edu.hk

Library of Congress Control Number: 2007940386

CR Subject Classification (1998): C.2, C.3, D.4, D.2, H.4, H.3, H.5, K.4

LNCS Sublibrary: SL 3 – Information Systems and Application,
incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-77091-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-77091-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12199662 06/3180 5 4 3 2 1 0

Preface

It has been widely recognized that embedded and ubiquitous computing will have tremendous impacts on many aspects of our daily life. Innovation and close collaboration between academia and industry are the keys to guaranteeing success in the development and deployment of the technology in embedded and ubiquitous computing.

The IFIP International Conference on Embedded and Ubiquitous Computing (EUC) provides a forum for engineers and scientists in academia, industry, and government to address challenges and to present and discuss their ideas, results, work in progress and experience. The Technical Program Committee (TPC) of EUC 2007 was lead by the TPC Chair, Tei-Wei Kuo, and TPC Vice Chairs. A strong international TPC was formed to review and evaluate the submissions. Each paper was reviewed carefully by at least three TPC members or external reviewers. It was extremely difficult for the TPC to select the presentations because there were so many excellent and interesting submissions. There were 217 submissions from all over the world, and only 65 papers are published in this proceedings volume.

We wish to thank the PC members for the time and thought that they gave in creating the excellent program. We also want to thank all of the authors who submitted their papers and made this conference a success. We are also grateful to the Organizing Committee in organizing the conference, and to the keynote speakers who agreed to give exciting speeches. Special thanks also go to Edwin Sha, the General Chair, for his excellent leadership, Zili Shao, Chi-Sheng Shih, Mieso Denko, Shih-Hao Hung, Chia-Lin Yang, Tai-Yi Huang, Chih-wen Hseuh, Morris Chang, Zhen Liu, Agustinus Borgy Waluyo, and Shi-Wu Lo for all the excellent work in the conference organization.

September 2007

Tei-Wei Kuo
Edwin Sha
Laurence T. Yang
Minyi Guo
Zili Shao

Organization

EUC 2007 was organized and supported by the International Federation for Information Processing (IFIP). It was held in cooperation with the National Taiwan University and [illegible] (LNCS) of Springer.

Orgnizing Committee

Honoraray Chair	Si-Chen Lee, National Taiwan University, Taiwan
General Chair	Edwin Sha, University of Texas at Dallas, USA
Program Chair	Tei-Wei Kuo, National Taiwan University, Taiwan
Program Vice Chairs	Li-Pin Chang, National Chiao-Tung University, Taiwan
	X. Sharon Hu, University of Notre Dame, USA
	Jinsoo Kim, KAIST, Korea
	Dongsheng Wang, Tsinghua University, China
	Hiroyuki Tomiyama, Nagoya University, Japan
	Samarjit Chakraborty, National University of Singapore, Singapore
	Yu-Chee Tseng, National Chiao-Tung University, Taiwan
	Nicolas Navet, INRIA Lorraine, France
	I-Ling Yen, University of Texas at Dallas, USA
	Cho-Li Wang, University of Hong Kong, Hong Kong
	Mohan Kumar, University of Texas at Arlington, USA
	Ai-Chung Pang, National Taiwan University, Taiwan
	Joseph Ng, Hong Kong Baptist University, Hong Kong
	Jiman Hong, Soongsil University, Korea
Steering Committee Chairs	Minyi Guo, University of Aizu, Japan
	Laurence Yang, St. Francis Xavier University, Canada
	Jane Liu, National Taiwan University and Academia Sinica, Taiwan
Local Organizing Chairs	Chia-Lin Yang, National Taiwan University, Taiwan
	Chih-Wen Hsueh, National Taiwan University, Taiwan
Registration and Finance Chairs	Tai-Yi Huang, National Tsing Hua University, Taiwan
	Shih-Hao Hung, National Taiwan University, Taiwan
	Shi-Wu Lo, National Chung-Cheng University, Taiwan

Workshop Chairs	Mieso Denko, University of Guelph, Canada Chi-Sheng Shih, National Taiwan University, Taiwan
Panel Chair	Ted Chang, Quanta Computer Inc., Taiwan
Publicity Chairs	Morris Chang, Iowa State University, USA Zhen Liu, Nagasaki Institute of Applied Science, Japan Agustinus Borgy Waluyo, Monash University, Australia
Publication Chairs	Zili Shao, Hong Kong Polytech University, Hong Kong Chi-Sheng Shih, National Taiwan University, Taiwan
Keynote	Dr. L.G. Chen, National Taiwan University, Taiwan Dr. Wei Zhao, Rensselaer Polytechnic Institute, USA

Technical Committee

Real-Time/Embedded Operating Systems	Li-Pin Chang, National Chiao-Tung University, Taiwan Tien-Fu Chen, National Chung Cheng University, Taiwan Jihong Kim, Seoul National University, Korea Kyu Ho Park, Korea Advanced Institute of Science and Technology, Korea Guohui Li, Huazhong University of Science and Technology, China Sam Hyuk Noh, Hong-Ik University, Korea Stephen A. Edwards, Columbia University, USA Sree. Rajan, Fujitsu Laboratories of America, USA Tai-Yi Huang, National Tsing Hua University, Taiwan Ying-Dar Lin, National Chiao-Tung University, Taiwan
Power-Aware Computing	X. Sharon Hu, University of Notre Dame, USA Chris Poellabauer, University of Notre Dame, USA Chi-Ying Tsui, Hong Kong University of Science and Technology, Hong Kong Tianzhou Chen, Zhejiang University, China Gang Quan, University of South Carolina, USA Joerg Henkel, Univeristy of Karlsruhe, Germany Kanishka Lahiri, NEC Laboratories America, USA Luca Benini, University of Bologna, Italy Pai Chou, University of California, Irvine, USA Vijaykrishnan Narayanan, Penn State University, USA Yung-Hsiang Lu, Purdue University, USA

HW/SW Co-design and Design Automation	Samarjit Chakraborty, National University of Singapore, Singapore Twan Basten, Eindhoven University of Technology, Nethelands Andy Pimentel, University of Amsterdam, Netherlands Aviral Shrivastava, Arizona State University, USA Chi-Ying Tsui, Hong Kong University of Science and Technology, Hong Kong Karam S. Chatha, Arizona State University, USA Mathias Gries, Intel Corp., Germany M. Balakrishnan, IIT Delhi, India Miguel Miranda, IMEC, Belgium Naehyuck Chang, Seoul National University, Korea Soumitra K. Nandy, Indian Institute of Science, India Marco Platzner, University of Paderborn, Germany Christian Plessl, ETH Zurich, Switzerland Prabhat Mishra, University of Florida, USA Sri Parameswaran, University of New South Wales, Australia Hiroto Yasuura, Kyushu University, Japan Zonghua Gu, Hong Kong University of Science and Technology, Hong Kong
Network Protocol	Ai-Chung Pang, National Taiwan University, Taiwan Jelena Misic, University of Manitoba, Canada Masayuki Murata, Osaka University, Japan Noel Crespi, GET-INT, France Shun-Ren Yang, National Tsing-Hua University, Taiwan Andreas Terzis, Johns Hopkins University, USA Yang Xiao, University of Alabama, USA
Embedded and Reconfigurable Hardware	Hiroyuki Tomiyama, Nagoya University, Japan Adam Donlin, Xilinx, USA Chia-Tien Dan Lo, University of Texas at San Antonio, USA Elaheh Bozorgzadeh, University of California, Irvine, USA Shinya Honda, Nagoya University, Japan Yin-Tsung Hwang, National Yunlin University of Science and Technology, Taiwan Ing-Jer Haung, National Sun Yat-Sen University, Taiwan Koji Inoue, Kyushu University, Japan

	Ki-Seok Chung, Hanyang University, Korea
	Yuichiro Shibata, Nagasaki University, Japan
Embedded System Software and Optimization	Jinsoo Kim, Korea Advanced Institute of Science and Technology, Korea
	Pete Beckman, Argonne National Laboratory, USA
	Francois Bodin, IRISA, France
	Jinsung Cho, Kyung Hee University, Korea
	Chanik Park, Samsung, Electronics, Korea
	Hwansoo Han, Korea Advanced Institute of Science and Technology, Korea
	Jenq-Kuen Lee, National Tsing-Hua University, Taiwan
	Rong-Guey Chang, National Chung-Cheng University, Taiwan
	Stephen A. Edwards, Columbia University, USA
	Sungsoo Lim, Kookmin University, Korea
	Yeh-Ching Chung, National Tsing-Hua University, Taiwan
Sensor Networks	Yu-Chee Tseng, National Chiao-Tung University, Taiwan
	Chih-Min Chao, National Taiwan Ocean University, Taiwan
	Chien-Chung Shen, University of Delaware, USA
	Hsi-Lu Chao, Naitonal Chiao-Tung University, Taiwan
	Hyuncheol Park, Information and Communication University, Korea
	Chung-Ta King, National Tsing Hua University, Taiwan
	Loren Schwiebert, Wayne State University, USA
	Mario Cagalj, University of Split, Croatia
	Ming-Hour Yang, Chung Yuan University, Taiwan
	Sameer S. Tilak, University of California, San Diego
	Sandeep Gupta, Arizona State University, USA
	Jang-Ping Sheu, National Central University, Taiwan
	Silvia Giordano, University of Applied Science - SUPSI, Switzerland
	Shih-Lin Wu, Chang Gung University, Taiwan
	Wang-Chien Lee, Penn State University, USA
	Yang Yang, University College London, Taiwan
	Yuh-Shyan Chen, National Taipei University, Taiwan

Mobile Computing	Nicolas Navet, INRIA Lorraine, France Ben A. Abderazek, National University of Electro-communications, Japan Jiannong Cao, Hong Kong Polytechnic University, Hong Kong Eric Fleury, INRIA-INSA Lyon, France Guoliang Xing, City University of Hong Kong, Hong Kong Jean-Dominique Decotignie, Centre Suisse d'Electronique et de Microtechnique, Switzerland Jiman Hong, Soongsil University, Korea Luis Almeida, University of Aveiro, Portugal Lucia Lo Bello, University of Catania, Italy Neil Audsley, University of York, UK Simonot-Lion Francoise, LORIA-INPL, France
Agent and Distributed Computing	I-Ling Yen, University of Texas at Dallas, USA Alessio Bechini, University of Pisa, Italy Ann T. Tai, IA Tech., Inc., USA Ing-Ray Chen, Virginia Tech., USA Kane Kim, UC, Irvin, USA Insup Lee, University of Pennsylvania, USA Yunhao Liu, Hong Kong University of Science and Technology, Hong Kong Neeraj Mittal, The University of Texas at Dallas, USA Shangping Ren, Illinois Institute of Technology, USA Jeffrey Tsai, University of Illinois at Chicago, USA Dongfeng Wang, Wind River, USA
Security and Fault Tolerance	Jiman Hong, Soongsil University, Korea Jean-Philippe Martin, Microsoft Research, UK Andres Marin, University Carlos III of Madrid, Spain Roberto Di Pietro, University of Rome La Sapienza, Italy Zhenhai Duan, Florida State University, USA Geyong Min, University of Bradford, UK Gwangil Jeon, Korea Polytechnic University, Korea Haklin Kimm, East Stroudsburg University of Pennsylvania, USA Hung-Chang Hsiao, National Tsing-Hua University, Taiwan Heejun Ahn, Seoul National University of Technology, Korea Jordi Forne, Technical University of Catalonia, Spain

	Junghoon Lee, Cheju National University, Korea
	Klaus Kursawe, Katholieke Universiteit Leuven, Belgium
	Madjid Merabti, Liverpool John Moores University, UK
	Marc Lacoste, France Telecom Division R&D, France
	Emilia Rosti, University of Milan, Italy
	Sangjun Lee, Soongsil University, Korea
	Willy Susilo, University of Wollongong, Australia
	Yi Mu, University of Wollongong, Australia
	Yunghsiang S. Han, National Taipei University, Taiwan
	Zhaoyu Liu, University of North Carolina at Charlotte, USA
	Yingwu Zhu, Seattle University, USA
Embedded System Architectures	Dongsheng Wang, Tsinghua University, China
	Achim Rettberg, University of Paderborn, Germany
	Guangzuo Cui, Peking University, China
	Franz Rammig, University of Paderborn, Germany
	Yunde Jia, Beijing Institute of Technology, China
	Junzhao Sun, University of Oulu, Finland
	Mingyu Lu, Dalian Maritime University, China
	Huadong Ma, Beijing University of Posts and Telecommunications, China
	Neil Bergmann, The University of Queensland, Australia
	Roger Woods, Queen's University of Belfast, UK
	Rajesh Gupta, University of California, USA
	Ming Xu, National University of Defense Technology, China
	Xiao Zong Yang, Harbin Institute of Technology, China
	Yingfei Dong, University of Hawaii, USA
	Zoran Salcic, University of Auckland, New Zealand
	Zhimin Zhang, Chinese Academy of Sciences, China
Middleware and P2P	Cho-Li Wang, University of Hong Kong, Hong Kong
	Bo Hong, Drexel University, USA
	Ching-Hsien Hsu, Chung Hua University, Taiwan
	Bin Xiao, Hong Kong Polytechnic University, Hong Kong
	Jemal Abbawajy, Deakin University, Australia
	Kuan-Ching Li, Providence University, Taiwan
	Zhiling Lan, Illinois Institute of Technology, USA

	Yunhao Liu, Hong Kong University of Science and Technology, Hong Kong Yuanchun Shi, Tsinghua University, China Young-Sik Jeong, Wonkwang University, Korea Weisong Shi, Wayne State University, USA Zhaohui Wu, Zhejiang University, China
Multimedia, Human-Computer Interface and Data Management	Joseph Ng, Hong Kong Baptist University, Hong Kong Leonard Barolli, Fukuoka Institute of Technology, Japan Jong Hyuk Park, R&D Institute in Hanwha S&C Co., Ltd., Korea Clement Leung, Victoria University, Australia Reynold Cheng, Hong Kong Polytechnic University, Hong Kong Victor Lee, City University of Hong Kong, Hong Kong David Tanier, Monash University, Australia Kazunori Takashio, Keio University, Japan Hidenori Nakazato, Waseda University, Japan Seongsoo Hong, Seoul National University, Korea Tatsuo Nakajima, Waseda University, Japan Timothy Shih, Tamkang University, Taiwan Jianliang Xu, Hong-Kong Baptist University, Hong Kong
Wireless Networks	Mohan Kumar, University of Texas at Arlington, USA Giusseppe Anastasi, University of Pisa, Italy Manimaran Govindarasu, Iowa State University, USA Kwan-Wu Chin, Wollongong University, Australia Mijeom Kim, Korea Telecom, Korea Nallasamy Mani, Monash University, Australia Stephan Olariu, Old Dominion University, USA Cristina Pinotti, University of Perugia, Italy Swaroop Kalasapur, Samsung Research, USA Sieteng Soh, Curtin University of Technology, Australia Yonghe Liu, University of Texas at Arlington, USA
Main Track	Tei-Wei Kuo, National Taiwan University, Taiwan Chih-Yuan Huang, Silicon Integrated Systems Corp., Taiwan

Young-Sik Jeong, Wonkwang University, Korea
Bernd Kleinjohann, University of Paderborn, Germany
Chin-Fu Kuo, National University of Kaohsiung, Taiwan
Chi-Sheng Shih, National Taiwan University, Taiwan
Zili Shao, Hong Kong Polytech University, Hong Kong
Chih-Wen Hsueh, National Taiwan University, Taiwan
Doohyun Kim, Konkuk University, Korea
Shih-Hao Hung, National Taiwan University, Taiwan
Ken-ichi Itoh, Siebold University of Nagasaki, Japan
Jen-Wei Hsieh, National Chiayi University, Taiwan
Jun Wu, National Pingtung Institute of Commerce, Taiwan
Jianwu Zhang, Hangzhou Dianzi University, China
Lung-Jen Wang, National Pingtung Institute of Commerce, Taiwan
Nei-Chiung Perng, Genesys Logic, Taiwan
Shi-Wu Lo, National Chung-Cheng University, Taiwan
Ting-Ao Tang, Fudan University, China
Tai-Yi Huang, National Tsing Hua University, Taiwan
Xiaoyang Zeng, Fudan University, China
Chia-Lin Yang, National Taiwan University, Taiwan
Cheng-Zhong Xu, Wayne State University, USA
Dakai Zhu, University of Texas at San Antonio, USA

Table of Contents

Power Aware Computing

Reconfigurable Embedded Systems

Wireless Networks

Real-Time/Embedded Operating Systems

Embedded System Architectures

Scheduling and Resource Management

Mobile Computing

System Security

Networks Protocols

Fault Tolerance

Human-Computer Interface and Data Management

HW/SW Co-design and Design Automations

Service-Aware Computing

Sensor Networks

Ad Hoc and Sensor Networks

Ubiquitous Computing

Embedded Software Designs

Real-Time Loop Scheduling with Energy Optimization Via DVS and ABB for Multi-core Embedded System

Guochen Hua[1], Meng Wang[1], Zili Shao[1], Hui Liu[2], and Chun Jason Xue[3]

[1] Department of Computing, The Hong Kong Polytechnic University, Hong Kong
{04994029d,csmewang,cszlshao}@comp.polyu.edu.hk
[2] Software Engineering Institute, Xidian University, Xi'an, China
liuhui@xidian.edu.cn
[3] City University of Hong Kong Kowloon, Hong Kong,
jasonxue@cityu.edu.hk

Abstract. Dynamic Voltage Scaling (DVS) is an effective technique to reduce energy consumption of processors by dynamically adjusting the operational frequency and supply voltage. However, with feature sizes shrinking, the achievable power saving by DVS is becoming limited as the leakage power increases exponentially. Adaptive Body Biasing (ABB) is an effective technique to reduce leakage power by increasing the circuit's threshold voltage via body biasing. In this paper, we propose a novel real-time loop scheduling technique to minimize both dynamic and leakage energy consumption via DVS and ABB for applications with loops considering voltage transition overhead. The proposed algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), is designed to repeatedly regroup a loop based on rotation scheduling [4,5] and decrease the energy consumption via DVS and ABB within a timing constraint. We conduct experiments on a set of DSP benchmarks based on the power model of 70nm technology. The results show that our technique achieves big energy saving compared with list scheduling [8] and the algorithm in [11].

1 Introduction

Power consumption has become an extremely important issue for real-time embedded systems due to its significant impact on battery life, system density, cooling cost, and system reliability, etc. Traditionally, dynamic power is the primary contributor to total power consumption. Dynamic Voltage Scaling (DVS) is one of the most effective techniques to reduce the dynamic power by dynamically scaling down the supply voltage and operational frequency. However, supply voltage scaling often requires a reduction in the threshold voltage, which leads to an exponential rise in the sub-threshold leakage current, and hence the leakage power consumption. As technology feature size continues to shrink, leakage power is becoming comparable to dynamic power in the current generation of technology, and it will dominate the overall energy consumption in future

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 1–12, 2007.

technologies. Therefore, with the trend of adopting multi-cores in embedded systems, it is important to reduce both dynamic and leakage energy for multi-core embedded systems. As loops are prevalent in multimedia processing and Digital Signal Processing (DSP) applications, we focus on minimizing both dynamic and leakage energy consumption via DVS and ABB for real-time applications with loops considering time and energy transition overhead.

Many researches have been done on energy optimization using DVS and ABB for real-time embedded systems [9,11,1,5,2,13,6]. In [9], Martin et al. proposed a technique combining DVS and ABB to minimize both dynamic and leakage power consumption for unrelated tasks. And the expression for obtaining the optimal tradeoff between supply voltage and bias voltage is derived in that work. Yan et al. [13] proposed an algorithm with DVS and ABB for a task graph with real-time constraints. Huang et al. [6] proposed a leakage-aware compilation methodology that targets embedded processors with both DVS and ABB capabilities. Although these work can effectively reduce energy consumption, loop optimization is not considered. There has been some work on minimizing energy for application with loops. Saputra et al. [10] used several classic loop-oriented compiler optimization techniques such as loop fusion to reduce the execution time and then applied DVS to reduce energy. However, the whole loop is scaled with the same voltage in their work. Shao et al. [11] proposed a Dynamic Voltage Loop Scheduling algorithm (DVLS) to minimize the energy consumption considering transition overhead for applications with loops on multi-core embedded systems. However, leakage power is not considered in their work.

In this paper, we propose a novel real-time loop scheduling algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), to minimize energy consumption via performing DVS and ABB simultaneously based on the power model [9] for applications with loops considering transition overhead. Our basic idea is to repeatedly regroup a loop based on rotation scheduling [4,5] and decrease the energy consumption via DVS and ABB within a timing constraint. We conduct experiments on a set of DSP benchmarks based on a 70nm processor. The results show that our technique achieves big energy saving compared with list scheduling [8] and the algorithm in [11]. On average, our algorithm contributes to 65.61% energy reduction over list scheduling and 42.74% over DVLS algorithm [11].

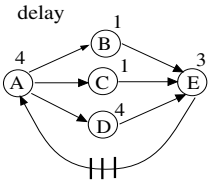
The rest of the paper is organized as follows. The motivational examples are shown in Section 2. The models and basic concepts are introduced in Section 3. The EOLSDA algorithm is presented in Section 4. The experimental results and analysis are provided in Section 5, and the conclusion is given in Section 6.

2 Motivational Examples

In this section, we motivate the loop scheduling problem with energy minimization via DVS and ABB by showing how to schedule a cyclic DFG that represents a loop on a real-time multi-core embedded system. We compare the energy consumption of the schedules generated by the list scheduling algorithm, the DVLS algorithm [11], and our technique.

```
E[0]=E[−1]=E[−2]=1;
for ( i=1; i<=N; i++){
    A[i]=E[i−3]*E[i−3];
    B[i]=A[i]+1;
    C[i]=A[i]+3;
    D[i]=A[i]*A[i];
    E[i]=B[i]+C[i]+D[i];
}
```

(a)



(b)

Node	Num of clock cycles	Dynamic Voltage Scaling				Combined DVS and ABB			
		High Voltage		Low Voltage		High Voltage		Low Voltage	
		Time	Energy	Time	Energy	Time	Energy	Time	Energy
A	4	0.4	10.48	0.8	6.72	0.4	5.88	0.8	3.04
B	1	0.1	2.62	0.2	1.68	0.1	1.47	0.2	0.76
C	1	0.1	2.62	0.2	1.68	0.1	1.47	0.2	0.76
D	4	0.4	10.48	0.8	6.72	0.4	5.88	0.8	3.04
E	3	0.3	7.86	0.6	5.04	0.3	4.41	0.6	2.28

(c)

Fig. 1. (a) A loop application. (b) The corresponding DFG (Data Flow Graph). (c) The number of clock cycles, execution time, and the energy of each node.

A loop application is shown in Figure 1(a). The corresponding DFG is shown in Figure 1(b). In DFG, each node represents the computation task in the loop, the edge without delay represents the intra-iteration data dependency (e.g. $A \rightarrow B$), the edge with delays represents the inter-iteration data dependency (e.g. $E \rightarrow A$ has three delays which are denoted by three bars), the number of delays represents the number of iterations involved, and the number beside each node represents the execution time of that node.

Assume that there are two processor cores in the multi-core embedded system, and the maximum frequency of each processor core is 10 GHz with the supply voltage/body bias voltage pair of (0.7244V, 0V). Note that the body bias voltage is set to 0 since there is no body bias voltage applied here. According to the power model introduced in [9], we get $P = 26.2\mu W$, $T = 0.1ns$, where P and T represent the power consumption and clock period, respectively. There are two voltage scaling approaches for reducing power consumption. One is Dynamic Voltage Scaling, and the other is combined Dynamic Voltage Scaling and Adaptive Body Biasing.

We first consider applying the Dynamic Voltage Scaling. Assume that the operational frequency can be scaled to 50% of the maximum frequency. By performing DVS, the frequency and the supply voltage can be scaled down from 10 GHz to 5 GHz, and 0.7244V to 0.4770V, respectively. Thus, the power consumption is reduced from $26.2\mu W$ to $8.4\mu W$.

Then we consider the approach that combines Dynamic Voltage Scaling and Adaptive Body Biasing. In this approach, we first apply the body bias voltage to the maximum frequency. Based on the power model in [9], we can obtain the optimal pair of supply and body bias voltage, which is (0.8189V, -0.6566V). Therefore, we get the power consumption reduced from $26.2\mu W$ to $14.7\mu W$.

Similarly, if the frequency is scaled down by 50%, by obtaining the optimal pair of supply and body bias voltage, which is (0.5795V, -0.7124V), the power consumption can be reduced to $3.8\mu W$. The execution time and energy of each node via DVS alone and combined DVS and ABB are shown in 1(c), where the time unit is ns and the energy unit is $10^{-15}J$. Since the voltage transition causes both time and energy overhead, we assume it takes $0.1ns$ with $10^{-15}J$ to transit between different frequency levels. These assumptions are only for demonstration purpose. Our technique is general enough to deal with general energy models as discussed in later sections.

Assume that the timing constraint is $1.2ns$ which is the upper bound for the schedule length of the loop. If the execution time of a loop iteration is less than a given timing constraint, we use the real execution time for energy calculation.

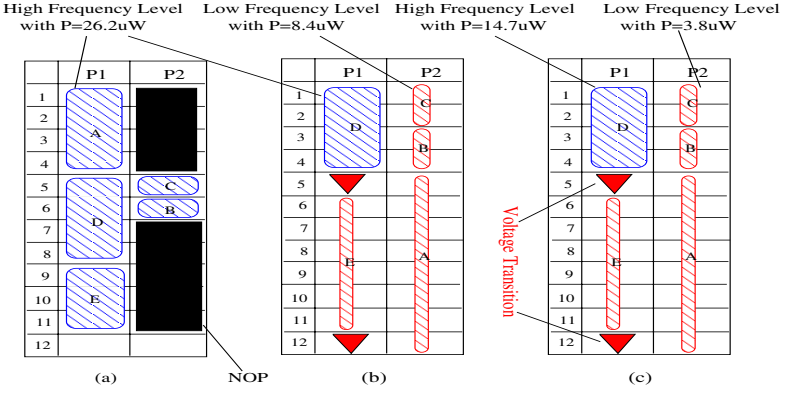


Fig. 2. The schedules generated by list scheduling, DVLS and our technique, respectively. The energy: (a) $2 * 1.1 * 26.2 = 57.64 \times 10^{-15}J$, (b) $26.2 * 0.4 + 2 + 8.4 * 1.8 = 27.6 \times 10^{-15}J$, and (c) $14.7 * 0.4 + 2 + 3.8 * 1.8 = 14.72 \times 10^{-15}J$.

In the following, we compare the energy consumption of the schedules generated by different techniques. Note that each time slot in the schedule represents 0.1ns. We obtain the first schedule by the list scheduling (shown in Figure 2(a)), where both processor cores operate at the maximum frequency level for the best timing performance. Consider all empty slots filled with NOP operations, we get the energy consumption $E = 2 * 1.1 * P_{max} = 2 * 1.1 * 26.2 = 57.64 \times 10^{-15}J$.

The second schedule shown in Figure 2(b) is generated by the DVLS algorithm [11] which repeatedly rotates the schedule and applies DVS for energy minimization. Two frequency levels 10 GHz (the high level) and 5 GHz (the low level) are used in this example, with $P_H = 26.2\mu W$, $P_L = 8.4\mu W$, and $T_H = 0.1ns$, $T_L = 0.2ns$, respectively. This schedule is generated after applying DVLS for 3 rotation steps based on the list schedule in Figure 2(a). In this schedule, nodes A, B, C, and E are assigned to the low level, while node D is assigned to the high level. Processor core P1 operates at two different frequency levels by processing D at the high level and E at the low level.

Considering the transition overhead, the energy consumption of this schedule is $E = P_H * 0.4 + 2 * E_{tran} + P_L * (0.6 + 1.2) = 26.2 * 0.4 + 2 + 8.4 * 1.8 = 27.6 \times 10^{-15} J$.

The schedule generated by our EOLSDA algorithm is shown in Figure 2(c). Compared with the schedule in Figure 2(b), this schedule consumes less energy since leakage power is reduced as well as dynamic power by performing DVS and ABB simultaneously. In EOLSDA, we also use the two frequency levels 10 GHz and 5 GHz, with $T_H = 0.1ns$ and $P_H = 14.7\mu W$ for the high level, and $T_L = 0.2ns$ and $P_L = 3.8\mu W$ for the low level. The total energy consumption of this schedule is $E = P_H * 0.4 + 2 * Etran + P_L * (0.6 + 1.2) = 14.7 * 0.4 + 2 + 3.8 * 1.8 = 14.72 \times 10^{-15} J$. The results show that our EOLSDA algorithm achieves big energy saving compared with the list scheduling and DVLS algorithm.

3 Models and Concepts

In this section, we introduce the power model and some basic concepts that will be used in this paper.

3.1 Power Model

In this subsection, we summarize the previous power model [9] that derives power consumption and the performance as functions of the supply and body bias voltages.

Power Consumption. In MOSFET circuit, there are three major sources of power consumption: dynamic power, static power, and short circuit power. The short circuit power consumption occurs only during signal transitions and is negligible [12]. The dynamic power, P_{AC} is given by,

$$P_{AC} = C_{eff} V_{dd}^2 f \tag{1}$$

where C_{eff} is the average switched capacitance per cycle, and f is the clock frequency. The static power consumption consists of the power due to sub-threshold leakage current and reverse bias junction current. Thus, the static power consumption, P_{DC} is given by,

$$P_{DC} = V_{dd} I_{subn} + |V_{bs}| (I_{jn} + I_{bn}) \tag{2}$$

where I_{subn} is the sub-threshold leakage current, I_{jn} and I_{bn} are the drain to body junction leakage current and source to body junction leakage current in the NMOS device. As $I_{jn} + I_{bn}$ can be approximated as a constant, Ij , the static power can be expressed as,

$$P_{DC} = V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j \tag{3}$$

thus the total power consumption, P, becomes,

$$P = C_{eff} V_{dd}^2 f + V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j \tag{4}$$

therefore, the total energy consumed per cycle is given by,

$$E_{cyc} = C_{eff} V_{dd}^2 + L_g f^{-1} (V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j) \tag{5}$$

Energy Optimization. To optimize the energy consumption, we utilize the equation (8) and (9) derived in [9] to find the optimal pairs of supply and body bias voltages for given frequencies and process technologies. Equation (8) illustrates the relationship between the body bias voltage and the derivative of the total energy consumption per cycle. Equation (9) formulates the supply voltage as a function of the body bias voltage.

$$\frac{\partial E_{cyc}}{\partial V_{bs}} = \{ L_g K_3 f^{-1}(k_1 V_{bs} + k_2) e^{k_3 V_{bs} + k_4} - I_j L_g f^{-1} \quad + 2C_{eff}(k_5 V_{bs} + k_6) \tag{6}$$

$$V_{dd} = (L_d K_6 f - K_2 V_{bs} + V_{th1})/(1 + K_1) \tag{7}$$

3.2 Rotation Scheduling

Rotation Scheduling [3,4] is a scheduling technique used to optimize a loop schedule with resource constraints. It transforms a schedule to a more compact one iteratively in a DFG. By using rotation scheduling, we can get more opportunities to reschedule nodes of DFG to better locations so that the length of a schedule can be reduced. In Figure 3, we show an example to explain how to obtain a new schedule via rotation scheduling. Using the schedule generated by list scheduling in Figure 2(a) as an initial schedule, we rotate the nodes at the first row of the schedule down. The rotated graph is shown in Figure 3(a), the new schedule is shown in Figure 3(b), and the equivalent loop body after rotation is shown in Figure 3(c).

From the program point of view, rotation scheduling regroups a loop body and attempts to reduce intra-dependencies among nodes. In this case, after the rotation, a new loop is obtained by the transformation as shown in Figure 3(c), where the corresponding computation for node A is rotated and put at the end of

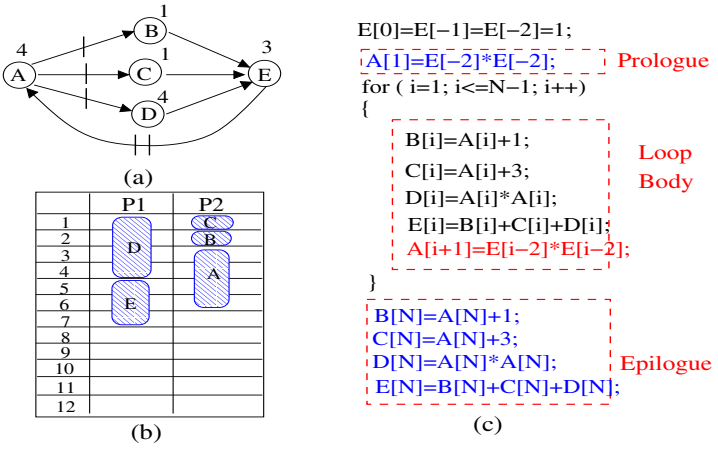


Fig. 3. (a) The rotated DFG. (b) The schedule after the rotation. (c) The equivalent loop after regrouping loop body.

the new loop body. One iteration from the old loop is separated and put outside the new loop body: the computation for node A is put in the prologue and those for the other nodes are put in the epilogue. In the new loop body, node A performs the computation for the $(i+1)_{th}$ iteration when the other nodes do the computation of the i_{th}. The transformed loop body after the rotation scheduling can be obtained based on the retiming values of nodes [4]. The code size is increased by introducing the prologue and epilogue after the rotation is performed. This problem can be solved by the code size reduction technique proposed in [14].

In the new schedule, the schedule length is reduced from 11 to 7 after the first rotation. Therefore, more opportunities are provided for energy optimization by DVS and ABB. In the following, we introduce our EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB) technique that can effectively reduce energy with loop optimization and combined DVS and ABB.

4 The Energy Optimization Loop Scheduling with DVS and ABB Algorithm

In this section, we propose our real-time loop scheduling algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), to minimize energy consumption via DVS and ABB for applications with loops. Our basic idea is to repeatedly regroup a loop based on rotation scheduling [3,4] and decrease the energy by combined DVS and ABB technique as much as possible within a timing constraint. The EOLSDA algorithm is shown in Algorithm. In EOLSDA, based on an initial schedule, we repeatedly reschedule nodes and adjust their frequency levels for energy minimization. The initial schedule can be obtained by assigning each node with the maximum frequency level using the list scheduling.

In the EOLSDA algorithm, we first put all rotatable nodes into Rotate_Node _Set and do retiming. If a node is the first node scheduled on a processor and there is at least one delay in each of its incoming edge, then the node is rotatable. For a node u that is not the first node scheduled on a processor, it is rotatable if the following two conditions are satisfied:

1. There is at least one delay in each of its incoming edge, and
2. All nodes scheduled before u on the same processor are rotatable.

In this way, we can rotate all nodes without disobeying dependencies [4].

After the rotatable set is obtained, the nodes in it are ordered based on the execution time so the node with longer execution time will be rescheduled earlier. Following such order, we can obtain a schedule that can satisfy the timing constraint as much as possible. The reason is that the frequency level of a rotated node will be adjusted for energy minimization in the next step so its execution time may be increased. Therefore, we should first reschedule the node with the longest execution time to avoid the situation that we can not find an empty slot with enough length to put it in.

When rescheduling a node, we try to find the earliest empty slot to put the node in, following the dependencies in the retimed graph. After finding this empty slot, we first put the node into it and adjust its frequency level so the minimum energy consumption can be obtained for the empty slot. This can be achieved by trying all possible frequency levels of the node and filling the rest of the empty slot with all frequency levels of NOP operations. Then we calculate the total energy of the empty slot and select the frequency level of the node with the minimum energy consumption.

It is possible that no empty slots can be found to put the rotated node in. In this case, we assign the node with the highest frequency level so that the node has the minimum execution time. We then repeat the above rescheduling steps to try to put the node into an empty slot with the earliest schedule time and adjust the frequency level for energy minimization. If we still can not find an empty slot even with the highest frequency level, we will pick up a processor to put the node to the end of it in such a way that the node is rescheduled as the last node on the processor with the earliest schedule time. Finally, we calculate the energy of the schedule and record the schedule with the minimum energy consumption if the timing constraint is satisfied with the schedule.

Figure 4 shows an example. Given the DFG shown in Figure 1(b) and the initial schedule in Figure 2(a), the schedules generated by the EOLSDA algorithm in three consecutive rotations are shown in Figure 4(a)-(c), respectively. As shown in the example, in each rotation, all rotated nodes are put into the rotation set and rescheduled with frequency adjustment for energy minimization. After three rotations, the schedule shown in Figure 4(c) achieves the minimum energy consumption for the DFG.

In the following, we perform complexity analysis on the EOLSDA algorithm. Let P_Num be the number of processor cores and k the number of the available frequency levels. Let $|V|$ and $|E|$ be the node and edge numbers of a given graph,

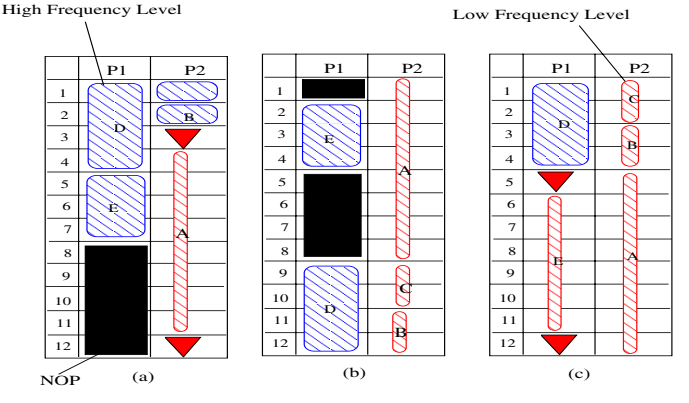


Fig. 4. The schedules generated by the EOLSDA algorithm for the DFG in Figure 1(b) with the initial schedule in Figure 2(a) after (a) the first rotation, (b) the second rotation, and (c) the third rotation

respectively. For the EOLSDA algorithm, let R_Num be the number of rotations. In one rotation, we can finish the retiming in $O(|V||E|)$ and it takes at most $O(|V| * log|V|)$ to order the nodes in the rotation set. When doing rescheduling, we need to reschedule at most $|V|$ nodes and it takes $O(|E| + P\_Num * |V|)$ to find an empty slot with the earliest schedule time where it takes at most $O(P\_Num*|V|)$ to find all empty slots in a schedule and takes at most $O(|E|)$ to determine the earliest available slot considering the edge dependencies. So totally it takes $O(|V|(|V| + |E|))$ to do rescheduling considering P_Num is a constant, and the EOLSDA algorithm can be finished in $O((|V|^2 + |V||E|) * R\_Num)$.

5 Experiments

In this section, we do experiments with a set of benchmarks including Infinite Impulse Response filter (IIR), 8-stage lattice filter (8-Stage), the differential equation solver (DEQ), elliptic filter (Elliptic), and voltera filter (Voltera). In the experiments, we set the rotation times as $10 * Node\_Num$ where $Node\_Num$ is the number of nodes in the DFG. The results show that the rotation times to generate the best schedule is less than $1 * Node_Num$ and close to the times when all nodes have been rotated one time.

Table 1. The frequency levels, corresponding Supply voltage, Body Bias Voltage and Power based on the power model of a 70nm processor

Frequencies f_{op} (GHz)	Supply Voltage V_{dd} (V)	Bias Voltage V_{bs} (V)	Power P (μW)
7.8	0.712	-0.667	8.76
10.4	0.839	-0.656	15.97
13	0.968	-0.661	26.28
15.6	1	-0.676	40.27

The experiments are conducted base on the power model of 70nm processor. We configure the parameters of the processor according to the power model in [9]. The frequency level of the processor can be varied between 50-100% in 16% steps, with the maximum frequency of 15.6GHz. According to the power model, we can obtain the optimal supply and body bias voltage for a given frequency. Then, the energy consumption per cycle can be calculated by using equation (9), and the power is derived from the formula $E_{cyc} = P/f$. The four frequency levels, as well as their corresponding Vdd and Vbs and energy consumptions, are shown in Table 1. The time overhead during a voltage transition among the above four voltage levels is calculated based on equation (12), and the energy overhead is calculated based on equation (13).

We obtain the number of clock cycles for instructions from the IA-32 architecture manual [7]. Basically, a NOP instruction takes one clock cycle, an Addition instruction with memory operands takes three clock cycles, and a Multiplication instruction with memory operands takes six clock cycles. Base on the above

Table 2. The frequency levels, corresponding Supply voltage and Power of a 70nm processor applying DVS only

Frequencies f_{op} (GHz)	Supply Voltage V_{dd} (V)	Power P (μW)
7.8	0.616	16.12
10.4	0.744	27.15
13	0.8728	43
15.6	1	65

Table 3. The energy comparison for the schedules generated by list scheduling, the DVLS algorithm and the EOLSDA algorithm

Bench.	TC Range (ns)	List Energy (10^{-15}J)	DVLS Energy (10^{-15}J)	EOLSDA Energy (10^{-15}J)	EOLSDA Imp-List (%)	EOLSDA Imp-DVLS (%)
2 processor cores						
IIR	3-7.5	315.21	186.11	107.15	66.01	42.43
DEQ	2-6.5	229.11	138.49	80.47	64.88	41.89
ELF	5-9.5	547.16	390.73	223.48	59.16	42.80
Voltera	5-9.5	566.42	385.19	223.59	60.53	41.95
8-Stage	9-13.5	820.80	533.23	302.96	63.09	43.18
Average Imp. (2 Cores)					**62.73**	**42.45**
3 processor cores						
IIR	3-7.5	321.60	177.63	99.84	68.96	43.79
DEQ	2-6.5	236.19	131.47	76.27	67.71	41.98
ELF	5-9.5	548.49	360.46	202.57	63.07	43.80
Voltera	5-9.5	563.52	335.05	181.71	67.75	45.77
8-Stage	9-13.5	820.60	429.30	237.66	71.21	44.64
Average Imp. (3 Cores)					**67.74**	**44.00**
4 processor cores						
IIR	3-7.5	322.88	179.49	105.58	67.30	41.17
DEQ	2-6.5	246.80	138.21	82.04	66.76	40.64
ELF	5-9.5	554.44	384.20	221.26	60.09	42.41
Voltera	5-9.5	568.32	318.92	191.44	66.32	39.97
8-Stage	9-13.5	830.40	426.71	238.05	71.33	44.21
Average Imp. (4 Cores)					**66.36**	**41.68**
Total Average Improvement					**65.61**	**42.71**

latencies, the execution time of each node is small in terms of transition overhead. Therefore, an enlarge factor of 1000000 is applied to the execution time of each node while all data dependency relations are kept. Moreover, since the clock cycle is too small (for instance, 0.13ns), we set the unit of time slot for scheduling is 0.01ns. Thus, the instructions can be scheduled in integer number of time slots, e.g. an NOP instruction will take 13 time slot to execute at the frequency level of 7.8GHz.

To evaluate the performance of our algorithm, we compare the energy consumption with the list scheduling and DVLS algorithm [11]. The energy consumption of the schedule generated by list scheduling is calculated under the maximum frequency level. For DVLS algorithm, we set all the $V_{bs} = 0$ since body bias voltage is not considered in [11]. In this case, for the same frequency scaling levels, we calculate the corresponding supply voltage and energy consumption for each frequency level according to equation (15) and (9). Table 2 lists the frequency levels of the 70nm processor when applying DVS.

For each benchmark, we apply 10 timing constraints with $0.5ns$ steps, and first timing constraint we use is the approximated minimum execution time from the list scheduling. The experiments are conducted on the systems with 2, 3 and 4 processor cores, respectively.

The experimental results are shown in Table 3. For each benchmark, we list the average energy consumption. In Table 3, column "TC" represents the range of timing constraints we applied. Sub-columns "Energy" under columns "List", "DVLS" and "EOLSDA" represent the average energy obtained by the list scheduling, DVLS and EOLSDA, respectively. Sub-column "Imp-List" under column "EOLSDA" represents the energy reduction of the EOLSDA algorithm over list scheduling. Sub-column "Imp-DVLS" under column "EOLSDA" represents the energy reduction of the EOLSDA over DVLS. The total average energy reduction of EOLSDA is shown in the last row.

The results show that the EOLSDA algorithm achieves great energy reduction compared with list scheduling and DVLS. On average, EOLSDA has a energy reduction of 65.61% over list scheduling, and a reduction of 42.71% over DVLS.

6 Conclusion

In this paper, we proposed a novel real-time loop scheduling algorithm to minimize both dynamic and leakage power consumption via performing DVS and ABB simultaneously for applications with loops considering transition overhead. The proposed algorithm, EOLSDA, is designed to repeatedly regroup a loop based on rotation scheduling [4] and decrease the energy by combining DVS and ABB as much as possible within a timing constraint. We conducted experiments on a set of DSP benchmarks based on the power model of the 70nm processor. The experimental results show that our algorithm achieves great energy reduction compared with list scheduling [8] and the algorithm in [11].

Acknowledgments

The work described in this paper was partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU A-PH13, PolyU A-PA5X, PolyU A-PH41, and PolyU B-Q06B).

References

1. Andrei, A., Schmitz, M., Eles, P., Peng, Z., Al-Hashimi, B.: Overhead-conscious voltage selection for dynamic and leakage power reduction of time-constraint systems. In: DATE 2004, pp. 518–523 (2004)
2. Andrei, A., Schmitz, M., Eles, P., Peng, Z., Al-Hashimi, B.: Simultaneous communication and processor voltage scaling for dynamic and leakage energy reduction in time-constrained systems. In: DATE 2004, pp. 362–369 (2004)
3. Chao, L.-F.: Scheduling and Behavioral Transformations for Parallel Systems. PhD thesis, Princeton University (1993)
4. Chao, L.-F., LaPaugh, A.S., Sha, E.H.-M.: Rotation scheduling: A loop pipelining algorithm. TCAD 16(3), 229–239 (1997)
5. Chen, Y., Shao, Z., Zhuge, Q., Xue, C., Xiao, B., Sha, E.: Minimizing energy via loop scheduling and dvs for multi-core embedded systems. Parallel and Distributed Systems 2, 2–6 (2005)
6. Huang, P., Ghiasi, S.: Power-aware compilation for embedded processors with dynamic voltage scaling and adaptive body biasing capabilities. In: DATE 2006, pp. 943–944 (2006)
7. Intel. IA-32 Intel Architecture Optimization Reference Manual (April 2006)
8. Landskov, D., Davidson, S., Shriver, B., Mallett, P.W.: Local microcode compaction techniques. ACM Computing Surveys 12(3), 261–294 (1980)
9. Martin, S., Flautner, K., Mudge, T., Blaauw, D.: Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: ICCAD 2002, pp. 721–725 (2002)
10. Saputra, H., Kandemir, M.: Energy-conscious compilation based on voltage sacling. In: LCTES 2002 (2002)
11. Shao, Z., Wang, M., Chen, Y., Xue, C., Qiu, M., Yang, L.T., Sha, E.H.-M.: Real-time dynamic voltage loop scheduling for multi-core embedded systems. Accepted in IEEE Transactions on Circuits and Systems II (TCAS-II)
12. Veendrick, H.: Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. IEEE J. Solid-State Circuits 19, 468–473 (1984)
13. Yan, L., Luo, J., Jha, N.K.: Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. TCAD 24(7), 1030–1041 (2005)
14. Zhuge, Q., Xiao, B., Sha, E.H.-M.: Code size reduction technique and implementation for software-pipelined dsp applications. TECS 2(4), 1–24 (2003)

A Software Framework for Energy and Performance Tradeoff in Fixed-Priority Hard Real-Time Embedded Systems

Gang Zeng, Hiroyuki Tomiyama, and Hiroaki Takada

Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan
{sogo,tomiyama,hiro}@ertl.jp

Abstract. A dynamic energy performance scaling (DEPS) framework is proposed to save energy in fixed-priority hard real-time embedded systems. In this generalized framework, two existing technologies, i.e., dynamic hardware resource configuration (DHRC) and dynamic voltage frequency scaling (DVFS) can be combined for energy performance tradeoff. The problem of selecting the optimal hardware configuration and voltage/frequency parameters is formulated to achieve maximal energy savings and meet the deadline constraint simultaneously. Through a case study, the effectiveness of DEPS has been validated.

1 Introduction

Power consumption has become one of the major concerns in today's embedded system design. Reducing power consumption can extend battery lifetime of portable systems, decrease chip cooling costs, as well as increase system reliability. In contrast to the traditional hardware-based low power designs, software-based energy performance tradeoff approaches have attracted much attention recently due to its flexibility and easy implementation. This approach is based upon the following observations: (1) Application needs for particular hardware resources such as caches, issue queues, and instruction fetch logic within an embedded processor can vary significantly from application to application and even within the different phases of a given application [8]. (2) In real-time systems the utilization of processor is less than 100% even if all tasks run at worse case execution time (WCET). Moreover, the actual workload even for the same task may vary from instance to instance, which depends on the specific input data and execution path. To take advantage of this application-dependent potential for energy and performance tradeoff, software-based approach tries to select the appropriate hardware resource for different applications or different program phases to save energy and meet the deadline constraint simultaneously.

There are two kinds of commonly used energy performance tradeoff technologies. One is dynamic hardware resource configuration (DHRC), such as adaptive-issue queue [13], adaptive branch prediction [10], selective cache way [11] etc. This technology tries to improve processor energy efficiency by dynamically tuning major processor resources in accordance with varied needs of applications [8]. However, its effectiveness on specific application is difficult to predict for two reasons. First, DHRC

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 13–24, 2007.

is application-dependent, i.e., a specific DHRC technique may be effective for some applications, but may be ineffective for other ones [9]. Second, even for a DHRC-effective application, the specific energy and performance relation for different hardware configuration is also difficult to predict. Another technology for energy performance tradeoff is dynamic voltage frequency scaling (DVFS) [1-7]. Because the dynamic power consumption of CMOS circuits is proportional to its clock frequency and its voltage square, DVFS tries to save energy by lowering both frequency and voltage of processor subject to deadline constraint. In contrast to DHRC, DVFS generally has similar effectiveness on different applications. That is, lowering frequency and voltage in a range always leads to longer execution time and less energy consumption. Moreover, the variation of execution time and energy consumption can be estimated by simple calculations. For example, most DVFS algorithms assume the execution time is linear-inversely proportional to the processor frequency.

Based on different criteria, the software-based energy performance tradeoff approaches can be classified into different categories. First, according to the granularity at which the technologies are applied, they can be classified into inter-task and intra-task approaches. While the inter-task approach targets for different applications (tasks) or different jobs of the same task; the intra-task approach is applied on periodic intervals [24], program phases [11, 12] or subroutines [9] within one application. Second, they can be classified into static (off-line) and dynamic (on-line) approaches according to when the configuration decisions are made.

Although both DHRC and DVFS are very effective for energy and performance tradeoff, unfortunately, combining them to achieve more energy savings is not a trivial problem. The reasons are that (1) while the energy consumption and execution time can be predicted by calculation after voltage/frequency scaling; they cannot be done so after hardware configuration is changed. Thus to guarantee hard real time for DHRC application, the only way to predict execution time is measurement. (2) As a general energy performance tradeoff technology, DVFS can be effectively applied to various applications. On the contrary, one kind of hardware resource configuration may be effective for some applications, but may be useless for other applications. Thus a framework should have the capability to accommodate different hardware configuration mechanisms.

In this work, we propose a generalized software framework, i.e., dynamic energy performance scaling (DEPS), to combine the two energy performance tradeoff technologies for more energy savings. This framework targets for hard real-time embedded systems with preemptive scheduling policy. As a first step, we discuss its static inter-task based application. In general, the static and inter-task based approach has global view of program power behaviors, low runtime overhead, simple implementation, and it is particularly suitable for task with stable workload. Through analysis of an actual DVFS application, it is suggested in [23] that while dynamic DVFS is of limited use in case of large DVFS overhead and without precise prediction of CPU load, static DVFS generally is sufficient. In addition, it is shown that static application of DHRC achieved better energy savings than dynamic one due to its global information of program behaviors [9]. Furthermore, though off-line approach cannot handle dynamic variations of workload, it can often be used as a complement to on-line approaches. The main contributions of this work are as follows: (1) Formulate the problem of selecting

the optimal hardware configuration and CPU voltage/frequency to achieve the maximal energy savings and meet the deadline requirements simultaneously. (2) Proposes a static application scheme of DEPS. (3) Construct a simulation environment for evaluating the proposed framework, and demonstrate the effectiveness of DEPS by a case study.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents the proposed DEPS framework. Section 4 gives a case study. Finally, Section 5 summarizes the paper.

2 Related Work

There have been a large number of publications using DHRC or DVFS for energy and performance tradeoff in recent years. Pillai and Shin proposed off-line and on-line DVFS algorithms under fixed-priority and earliest deadline first (EDF) scheduling policy, respectively [1]. Kim et al. evaluated various existing DVFS algorithms including both fixed-priority and EDF scheduling algorithm for hard real-time systems [2]. Saewong and Rajkumar proposed several off-line and on-line DVFS algorithms for fixed-priority real-time systems corresponding to processors with large or little DVFS overhead [3]. Mochocki et al. introduced off-line DVFS algorithms for EDF scheduling policy considering time and energy overhead of DVFS [25]. Cho et al. first proposed DVFS algorithms considering measured energy performance relations for different applications [4]. In contrast to the above inter-task approaches; Choi et al. presented a fine-grained intra-task DVFS algorithm for memory-bound application using performance counter for runtime measurement [5]. In [6], Shin and Kim also proposed intra-task DVFS algorithm using control flow information for hard real-time systems. Recently, Yuan et al. proposed cross-layer adaptation DVFS algorithm combining both inter-task and intra-task scaling for energy savings in a soft real-time application [7].

As far as DHRC is concerned, Albonesi proposed selective cache ways by using off-line program profiling and runtime program phase-based configuration [11]. Banerjee et al. proposed completely dynamic cache ways configuration using hardware-based program phase detector [12]. Both the above approaches utilize temporality-based program phase information to switch the configurations. On the contrary, Huang et al. proposed position-based (subroutine) hardware configuration approach including off-line and on-line algorithms [9]. Note that all these DHRC approaches performed fine-granularity configuration and not targeted for hard real-time systems, which is different from the proposed approach. Albonesi et al. summarized recent dynamically tuning processor resources approaches in [8].

Although the two technologies are effective for energy savings, there are few papers considering the combination of them due to the reasons discussed in Section 1. Huang et al. first proposed the combination of DVFS and hardware resource configuration for energy and temperature management in which an on-line interval-based algorithm was presented to select the most energy-saving configuration subject to a given slowdown factor [26]. While this work targets for single-task application with given slowdown factor, our approach target for multi-task hard real-time application with given period and deadline. Recently, Nacul and Givargis proposed combination of DVFS and cache

reconfiguration for low power [14].Their approach used an on-line algorithm for selecting the Pareto-optimal configuration that best fill the slack for the next task to be executed, which is different from our off-line optimal global exploration algorithm for all tasks. Moreover, our generalized framework can adopt various DHRC schemes, and not limited to cache reconfiguration.

3 Proposed DEPS Framework

3.1 System Model

This work focuses on embedded system and assumes a DHRC and DVFS enabled embedded processor. The DVFS can operate at a finite set of supply voltage levels, each with an associated speed.

We consider hard real-time applications consisting of a set of independent n periodic real-time tasks, represented as $\Gamma = \{\tau_1, \tau_2 ..., \tau_n\}$. Each task τ_i has a period P_i and relative deadline D_i that is equal to P_i. A task τ_i has m_i candidate DEPS configurations $\{C_{i1}, C_{i2}, \ldots, C_{imi}\}$ consisting of both DHRC configuration and DVFS parameters. Each DEPS configuration C_{ij} is associated with a specific energy time (performance) relation, which can be represented by a pair of values (T_{ij}, E_{ij}) where T_{ij} is its worst-case execution time under this DEPS configuration, and E_{ij} is its energy consumption corresponding to the T_{ij}.

Note that we employ measurement to obtain this application-dependent energy time relation for each DEPS configuration. There are two reasons for this. First, as described in Section 1, the only way for prediction of energy and time relation after DHRC configuration change is measurement. Second, although most DVFS papers use calculation to predict energy and time relation after voltage/frequency scaling, recent research reveals application-specific energy time relation through actual measurements, which can be exploited to further save energy over normal DVFS application [4, 5]. These application-specific power characteristics include memory or I/O access behaviors as well as leakage power consumption, etc., which is generally neglected by simple calculation.

3.2 Problem Formulation for Static Application of DEPS

We assume the overhead for task switching and DEPS configuration is negligible for simplicity, and denote $hyperperiod = LCM(P_1, P_2, \ldots, P_n)$, i.e., the least common multiple of all task periods. The problem is to determine the set of optimal DEPS configurations that minimize the energy consumption over a hyperperiod while meeting the deadline constraints. This problem can be formulated as follows:

Minimize energy:

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{HyperPeriod}{P_i} (E_{ij} - T_{ij} W_{idle}) C_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i}\frac{T_{ij}}{P_i}C_{ij} \le n(2^{1/n}-1) \tag{2}$$

and

$$\sum_{j=1}^{m_i}C_{ij}=1 \quad i=1,2,\cdots,n \tag{3}$$

where $C_{ij} \in \{0,1\}, \forall i, j$.

In the above formulation, W_{idle} denotes the idle power of processor. The constraint (2) represents utilization-based schedulability test for RM scheduling [16]. Note that more complex schedulability test such as response time analysis (RTA) [17] can also be used for fixed-priority based scheduling at the expense of higher computational complexity. Although we only give the schedulability test for fixed-priority based scheduling, it is straightforward to extend it to EDF based scheduling. Constraint (3) indicates that for one task, only one DEPS configuration can be selected where C_{ij} = 1 denotes that the configuration C_{ij} has been selected for task τ_i in DEPS framework, otherwise C_{ij} = 0.

It is clear that the problem for selecting the optimal DEPS configuration is a typically multiple choice 0/1 knapsack problem, which is known as a NP-hard problem [15]. Although there is no polynomial-time exact method for this problem, we can use common dynamic programming or mixed-integer linear programming method for solving any reasonable size by off-line computation.

Note that although we do not consider the configuration overhead in the above formulation for simplicity, they can be incorporated easily. This is because in one hyperperiod, the occurred number of hardware configuration and DVFS settings is known. Thus, if the DEPS overhead in terms of time latency and energy consumption for once hardware configuration and DVFS setting is also known, their influences can be incorporated into formula one and two. A detailed discussion on the overhead of DHRC and DVFS configuration can be found in [9] and [25], respectively.

3.3 Decision Algorithm for Selecting Candidate DEPS Configurations

Actually, a processor may have many DEPS configurations consisting of different DHRC and DVFS parameters. To reduce the computational complexity we only select some of them as candidates in the above optimal computation. As discussed in Section 1, because DVFS is effective for any applications, we retain all DVFS parameters as candidates directly. And then, to select effective DHRC configuration under the same DVFS parameters, first, we conduct measurement to obtain energy time relation for all possible DEPS configurations. Second, the maximal energy consumption E_{max} and the minimal execution time T_{min} from the above results are selected as comparative objects. Third, different DHRC configurations C_{ij} (T_{ij}, E_{ij}) with the same DVFS parameters are compared with each other by calculating its energy improvement over performance

degradation, which is represented by $(E_{max}-E_{ij})/(T_{ij}-T_{min}+1)$. Finally, the DHRC configurations with higher energy improvement rate will be selected as candidates in the optimal computation.

3.4 Implementation of Static DEPS

The implementation procedure of static DEPS mainly includes the following steps:

1. Obtain application-dependent energy time relation under all possible DEPS configurations by simulation or actual measurement.
2. Select candidate DEPS configurations for the optimal computation as the proposed decision algorithm.
3. Solve the energy optimal problem using the above formulation and obtain the optimal DEPS configuration for each task.
4. Store the optimal DEPS configuration including corresponding hardware parameters into a static configuration table.
5. OS scheduler sets corresponding DEPS configuration based on the static configuration table for next task to run at every context switch.

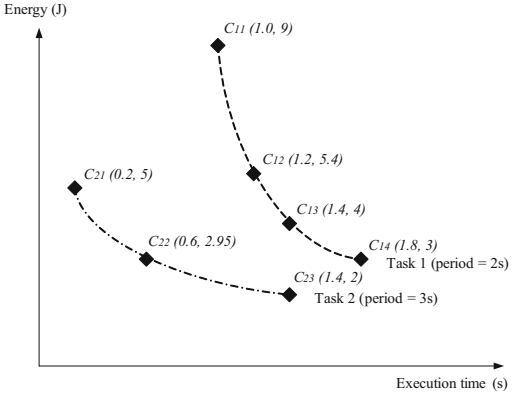


Fig. 1. An example for DEPS including two tasks and 7 selected DEPS configurations

We use the following example to illustrate the application of DEPS. This simple example includes two periodic tasks and 7 candidate DEPS configurations as shown in Fig.1, where $C_{11}(1.0, 9)$ indicates that for DEPS configuration C_{11} of task1, its corresponding worse case execution time and energy consumption are 1.0s and 9J, respectively. The idle power of processor is assumed to be 1 W. As the above formulation, the objective of DEPS is to find the optimal configuration combination for two tasks that can achieve the minimal energy and meet the deadline constraints simultaneously. The DEPS results for one hyperperiod scheduling are given in Fig.2. As can be seen, RTA-based method has more potential on energy savings, and considering idle power in the formulation can lead to more energy savings than without consideration of idle power.

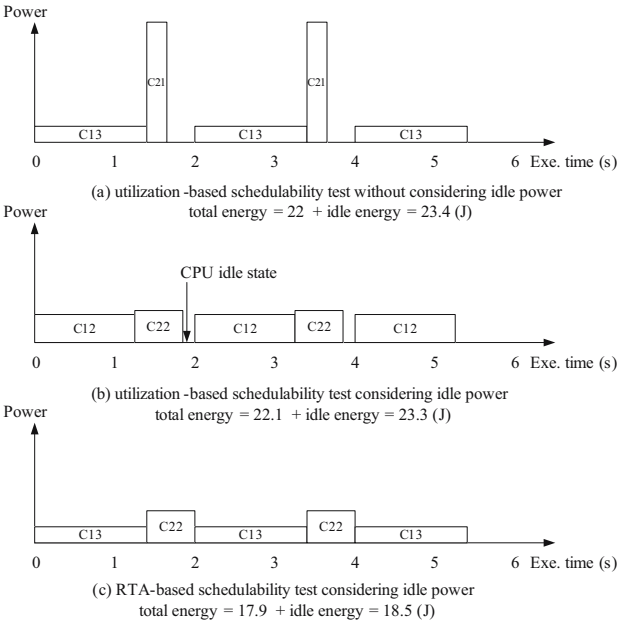


Fig. 2. DEPS results using different schedulability test methods with and without considering idle power

4 A Case Study

As mentioned earlier, because DEPS can adopt various DHRC and DVFS techniques, the achievable energy savings of DEPS are highly dependent on the employed DHRC and DVFS. Therefore, it is difficult to evaluate the absolute energy savings of general DEPS. For this reason, we demonstrate the effectiveness of DEPS through a case study.

We choose a 4-level voltage DVFS and the selective cache way (SCW) [11] as DHRC for our DEPS framework in this case study. In [3], it is shown that limited voltage/frequency level will result in more energy consumption for DVFS applications. However, while most general-purpose commercial DVFS processors can provide more voltage levels, embedded processors typically have less ones due to its relatively low running frequency. For example, the evaluation board of TMS320C5509 only provides 3-voltage levels [22]. The reason for selecting SCW is due to its easy implementation and low configuration overhead. SCW exploits the subarray partitioning of set associative caches in order to provide the capability to disable ways of the cache during periods where full cache functionality is not required to achieve energy savings. The detail implementations of SCW, configuration overhead, as well as method for keeping data coherency can be found in [11] and [12]. Note that our DEPS framework is general

and independent of the employed DHRC and DVFS technologies. We simple choose the above technologies as an example of DEPS.

4.1 Simulation Environment Setup

As we focus on embedded systems, a SimpleScalar/ARM [18] based Sim-Panalyzer [19] power simulator is employed to run the power simulation for our experiments. Sim-Panalyzer is an infrastructure for microarchitectural power simulation considering both dynamic and leakage power. The ARM configuration for SimpleScalar is listed in Table 1. Note that we only implement the SCW on instruction cache to further reduce the configuration overhead associated with writing cache operations. The possible configurations for SCW on L1 instruction cache are summarized in Table 2. In addition to the above configurations for SimpleScalar, Sim-Panalyzer uses its default configuration. Furthermore, we incorporate the DVFS capability into the Sim-Panalyzer as shown in Table 3. Some benchmark programs from Mibench [20] and Powerstone [21] that have distinct power characteristics are selected for this evaluation. A task set including these benchmark programs is assumed to run on this ARM simulator using fixed-priority scheduling with specified periods in Table 4.

Table 1. Configuration for SimpleScalar/ARM

Fetch queue	8
Branch Predictor	Not-taken
Fetch & Decode width	1
Issue width	1 (in-order)
Functional units	1 int ALU, 1 int Multiplier 1FP ALU, 1 FP Multiplier
Instruction L1 Cache	selective cache way (SCW)
Data L1 Cache	Size: 8KB; sets: 64; block size: 32-byte; 4-way
L2 Cache	none
Memory bus width	4-byte

Table 2. Instruction L1 cache SCW configurations

Parameters	Config. 1	Config. 2	Config. 3
Cache size (KB)	8	4	2
Num. of sets	64	64	64
Block size	32	32	32
Associativity	4	2	1
Replacement policy	LRU	LRU	LRU

Table 3. Configuration for DVFS

Processor frequency (MHz)	280	220	160	100
Processor voltage (V)	2.0	1.8	1.6	1.4

Table 4. Task set for experiments

No.	Task name	Period (ms)	WCET(ms) : 280MHz; HRC config.1	Total CPU utilization
1	sha	400	64.9	59%
2	v42	200	36.7	
3	engine	100	8.7	
4	g3fax	100	15.6	

4.2 Experimental Results

According to the above Table 2 and 3, there are 3 configurations for DHRC and 4 configurations for DVFS. Therefore, this framework can provide total 12 possible DEPS configurations for each task. Each benchmark is simulated 12 times using Sim-Panalyzer, which corresponds to 12 DEPS configurations. The simulation results are summarized in Table 5, in which the HRC denotes the hardware resource configuration as shown in Table 2, and VF denotes the voltage frequency parameters as shown in Table.3. As these results show, DVFS can provide an identical energy performance tradeoff for all benchmarks. That is, lowering processor frequency and voltage leads to longer execution time and less energy consumption. However, for DHRC, the energy performance tradeoff is highly dependent on program behaviors. For example, while the large instruction cache (HRC config.1: 8KB) can achieve better energy performance results for v42 benchmark; small instruction cache (HRC config. 3: 2KB) is the better choice for g3fax benchmark because it leads to negligible variation of execution time but with less energy consumption.

Table 5. Simulation Results for Benchmarks

Name	VF / HRC	280 MHz		220MHz		160 MHz		100MHz	
		E(mJ)	T(ms)	E(mJ)	T(ms)	E(mJ)	T(ms)	E(mJ)	T(ms)
sha	config.1	24.34	64.88	19.93	82.60	15.94	113.16	12.62	180.91
	config.2	**20.35**	**64.90**	**16.69**	**82.63**	**13.37**	**113.19**	**10.64**	**180.95**
	config.3	**19.37**	**66.92**	**16.09**	**84.98**	**12.93**	**115.40**	**10.48**	**184.01**
v42	config.1	**13.40**	**36.72**	**11.10**	**46.35**	**8.93**	**61.94**	**7.23**	**98.24**
	config.2	14.66	44.48	12.60	55.37	10.28	70.43	8.79	109.95
	config.3	25.82	72.90	23.54	88.36	19.71	101.28	18.15	152.42
engine	config.1	3.22	8.69	2.63	11.05	2.11	15.17	1.67	24.25
	config.2	**2.72**	**8.69**	**2.22**	**11.05**	**1.78**	**15.17**	**1.42**	**24.26**
	config.3	4.70	14.10	4.17	17.33	3.36	21.03	2.99	32.30
g3fax	config.1	6.00	15.56	4.90	19.80	3.92	27.18	3.10	43.48
	config.2	5.13	15.56	4.20	19.80	3.36	27.19	2.66	43.48
	config.3	**4.71**	**15.58**	**3.85**	**19.82**	**3.09**	**27.20**	**2.46**	**43.50**

The selected candidate DEPS configurations for each benchmark as the proposed decision algorithm are denoted in boldface in the table. In this case study, we use LPSolve tool [27], a free mixed integer linear programming solver, to solve the energy optimal problem as described in Section 3.2. DEPS results corresponding to different schedulablity test methods are reported in Table 6 and 7. It is clear that DEPS can

Table 6. DEPS results for fixed-priority scheduling using utilization-based schedulability test

No.	Task name	DEPS results: total energy 62.57 mJ	
		HRC	VF
1	sha	config. 3	220 MHz
2	v42	config. 1	220 MHz
3	engine	config. 2	220 MHz
4	g3fax	config. 3	220 MHz

Table 7. DEPS results for fixed-priority scheduling using RTA-based schedulability test

No.	Task name	DEPS results: total energy 52.03 mJ	
		HRC	VF
1	sha	config. 3	160 MHz
2	v42	config. 1	160 MHz
3	engine	config. 2	220 MHz
4	g3fax	config. 3	160 MHz

Table 8. Comparison with other methods

Task name & Results	SVFS [1][3]		Opt-clock [3]		Static DHRC		DEPS	
	HRC	VF	HRC	VF	HRC	VF	HRC	VF
sha	config.1	220	config.1	160	config.3	280	config.3	160
v42	config.1	220	config.1	160	config.1	280	config.1	160
engine	config.1	220	config.1	220	config.2	280	config.2	220
g3fax	config.1	220	config.1	160	config.3	280	config.3	160
Energy of hyperperiod	72.3 mJ		60.0 mJ		75.9 mJ		**52.0 mJ**	
Ave. power	180.6 mW		150.0 mW		189.7 mW		**130.1 mW**	
Power red.	53.1 %		61.0 %		50.7 %		**66.2 %**	

achieve the minimal energy consumption and meet the deadline simultaneously by selecting the optimal DEPS configuration.

Table 8 compares the DEPS with other power saving methods. Note that because the proposed DEPS is an inter-task based static method, we also select the inter-task based static application of DVFS and DHRC for fair comparison. In addition, we assume that static application of DVFS utilizes full hardware resource, and static application of DHRC utilizes the highest processor performance. In Table 8, the column denoted as SVFS represents the static voltage frequency scaling methods proposed in [1] and [3], in which identical speed is assigned to all tasks to reduce the energy loss caused by large DVFS overhead. The column denoted as Opt-clock represents the optimal speed assignment method proposed in [3]. This method statically assigns different speed for different tasks to achieve the maximal energy savings. Because the absolute energy consumption depends on the run time of application, we compare the average power of various methods to the maximal power consumption in this ARM-based simulator, i.e., 385 mW when running g3fax at 280 MHz on HRC config. 1. As can be seen from Table 8, the DEPS can achieve 66.2% power reduction and a 5%-15% improvement over previous methods when original task set has a total CPU utilization of 59%.

To verify the relation of the CPU utilization and power reduction rate, we extend the periods of sha and v42 in Table 4, to 600 and 300 ms, respectively, which means a lower CPU utilization, i.e., 47%. And then, the above experiments are conducted again, and results show a 75.7% reduction in power consumption, which is a significant improvement over the case of 59% CPU utilization.

5 Conclusion

We proposed a generalized software framework, i.e., DEPS: dynamic energy performance scaling for energy savings targeting for hard real-time embedded systems. It integrates two existing energy performance tradeoff technologies, i.e., dynamic hardware resource configuration and dynamic voltage frequency scaling into this framework. We formulate the problem of selecting the optimal DEPS configuration to achieve maximal energy savings and meet the deadline constraint simultaneously. As a first step, we propose static task-level application of DEPS. Through a case study, DEPS shows 66% power reduction and a 5%-15% improvement over previous methods in the case of 59% CPU utilization.

Acknowledgments. The authors would like to thank Professor Tohru Ishihara at System LSI Research Center of Kyushu University for his valuable comments. This work is supported by the Core Research for Evolutional Science and Technology (CREST) from Japan Science and Technology Agency.

References

1. Pillai, P., Shin, K.G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In: Proc. ACM Symposium Operating Systems Principles, pp. 89–102 (2001)
2. Kim, W., Shin, D., Yun, H., Kim, J., Min, S.L.: Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In: Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 219–228 (2002)
3. Saewong, S., Rajkumar, R.: Practical Voltage Scaling for Fixed-Priority RT-Systems. In: Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 106–114 (2003)
4. Cho, Y., Chang, N., Chakrabarti, C., Vrudhula, S.: High-Level Power Management of Embedded Systems with Application-Specific Energy Cost Functions. In: Proc. Design Automation Conference (DAC), pp. 568–573 (2006)
5. Choi, K., Soma, R., Pedram, M.: Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff Based on the Ratio of Off-Chip Access to On-Chip Computation Times. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 24(1), 18–28 (2005)
6. Shin, D., Kim, J.: Intra-Task Voltage Scheduling on DVS-Enabled Hard Real-Time Systems. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 24(10), 1530–1549 (2005)
7. Yuan, W., Nahrstedt, K., Adve, S.V., Jones, D.L., Kravets, R.H.: GRACE-1: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. IEEE Trans. Mobile Computing 5(7), 799–815 (2006)

8. Albonesi, D.H., Balasubramonian, R., Dropsbo, S.G., et al.: Dynamically Tuning Processor Resources with Adaptive Processing. IEEE Computer, 49–58 (2003)
9. Huang, M., Renau, J., Torrellas, J.: Positional Adaptation of Processors: Application to Energy Reduction. In: Proc. IEEE International Symposium Computer Architecture, pp. 157–168 (2003)
10. Chaver, D., Pinuel, L., Prieto, M., Tirado, F., Huang, M.: Branch Prediction on Demand: An Energy-Efficient Solution. In: Proc. International Symposium on Low-Power Electronics and Design, pp. 390–395 (2003)
11. Albonesi, D.H.: Selective Cache Ways: On-Demand Cache Resource Allocation. In: Proc. International Symposium on Microarchitecture, pp. 248–259 (1999)
12. Banerjee, S., Nandy, G.S., Program, S.K.: Phase Directed Dynamic Cache Way Reconfiguration for Power Efficiency. In: Proc. Asia and South Pacific Design Automation Conference (ASPDAC), pp. 884–889 (2007)
13. Buyuktosunoglu, A., et al.: A Circuit-Level Implementation of an Adaptive-Issue Queue for Power-Aware Microprocessors. In: Proc. Great Lakes Symp. VLSI, pp. 73–78. ACM Press, New York (2001)
14. Nacul, A., Givargis, T.: Dynamic Voltage and Cache Reconfiguration for Low Power. In: Proc. Design Automation and Test in Europe (DATE), pp. 1376–1377 (2004)
15. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. Wiley, Chichester (1990)
16. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM 20(1), 40–61 (1973)
17. Lehoczky, J.P., Sha, L., Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In: Proc. IEEE Real Time Systems Symposium (RTSS), pp. 166–171 (1989)
18. SimpleScalar Tools, http://www.simplescalar.com/
19. Sim-Panalyzer Project, http://www.eecs.umich.edu/ panalyzer/
20. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In: IEEE Annual Workshop on Workload Characterization (2001)
21. Scott, J., Lee, L., Arends, J., Moyer, B.: Designing the Low-Power M*CORE Architecture. In: Proc. International Symposium on Computer Architecture Power Driven Microarchitecture Workshop, pp. 145–150 (1998)
22. Texas Instruments, Application Report, SPRA848A: Using the Power Scaling Library (2004)
23. Texas Instruments, Application Report, SPRAA19A: Power Management in an RF5 Audio Streaming Application Using DSP/BIOS (August 2005)
24. Lee, S., Sakurai, T.: Run-Time Voltage Hopping for Low-Power Real-Time Systems. In: Proc. Design Automation Conference (DAC), pp. 806–809 (2000)
25. Mochocki, B.C., Hu, X.S., Quan, G.: A Unified Approach to Variable Voltage Scheduling for Nonideal DVS Processors. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 23(9), 1370–1377 (2004)
26. Huang, M., Renau, J., Yoo, S.M., Torrellas, J.: A Framework for Dynamic Energy Efficiency and Temperature Management. In: Proc. International Symposium on Microarchitecture (MICRO), pp. 202–213 (2000)
27. LPSolve tool: http://sourceforge.net/projects/lpsolve/

A Shortest Time First Scheduling Mechanism for Reducing the Total Power Consumptions of an IEEE 802.11 Multiple Rate Ad Hoc Network

Weikuo Chu[1,2] and Yu-Chee Tseng[1]

[1] Department of Computer Science
National Chiao-Tung University, Hsin-Chu, Taiwan
[2] Department of Information Management
St. John's University, Tamsui, Taipei, Taiwan

Abstract. Power management is one of the most important issues in mobile communications. Much research has been done in reducing wireless station's power consumptions. IEEE 802.11 addresses this issue by adopting a MAC layer active-doze Power Saving Mechanism. In an 802.11 ad hoc network, this Power Saving Mechanism works as follows. Any wireless station with data to send must first announce its traffic and then contends for the channel with other stations for data transmissions, all based on the DCF protocol. Stations not involved in any data transmissions can go to the doze mode to conserve energy. In this paper, we first show that this mechanism has the problem of power management inefficiency when used in a multiple rate ad hoc network. We then propose a novel scheduling mechanism, STFS, to reduce the total power consumptions of the wireless stations in the network. Simulation results show that the proposed scheduling mechanism does have better performance than that of 802.11 PSM.

1 Introduction

Wireless LAN or WLAN is the fastest growing field in mobile communications. By now, the majority of notebook computers and an increasing number of PDAs are equipped with wireless technology. Among the many wireless technologies, the family of IEEE 802.11 protocols is the most widely used access method for WLAN. In IEEE's proposed protocols for WLAN, there are two different ways to configure a network: ad hoc and infrastructure. In the ad hoc configuration, wireless stations (STAs) are brought together to form a network "on the fly". There is no structure to the network; there are no fixed points; and usually every STA is within the communication range of every other STA in the network. When configured in infrastructure mode, the WLAN consists of at least one access point (AP) connected to the wired network and a number of wireless STAs. The AP provides a local relay function for the network. All STAs in the network communicate with the AP and no longer communicate with each other directly.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 25–32, 2007.

In WLAN, battery power is an unavoidable issue that must be dealt with. In order to save power, 802.11 defines a MAC-layer Power Saving Mechanism (802.11 PSM) that allows a wireless STA to go from the active state to doze or power-saving state when the STA is not involved in any data transmissions [1]. In the infrastructure configuration of a WLAN, the AP will keep track of all STAs that are in power-saving state and buffer frames addressed to these STAs. These frames are kept until the STAs request them to be sent or discarded if they are not requested for a certain period of time. While in the case of ad hoc configuration, time is divided into Beacon Intervals and each Beacon Interval contains an ATIM (Ad Hoc Traffic Indication Message) Window followed by the Data Transmission Phase. The ATIM Window is used as the common awake period for all participating STAs to announce their traffic through ATIM frame transmissions. After the ATIM Window finishes, STAs that successfully send or receive ATIM frames must remain in the active state, and STAs can switch to power-saving state if they are not involved in any traffic announcements till the beginning of next ATIM Window. Actual data transfers occur in the Data Transmission Phase, and the normal DCF (Distributed Coordination Function) access procedure is used while sharing the transmission medium among the active STAs. Any STA that completes the ATIM frame transmission in the ATIM Window but fails to send data packet in the Data Transmission Phase will try to initiate another traffic announcement in the next ATIM Window. In addition to the 802.11 PSM, a number of power saving methods [2, 3] covering all protocol layers from Physical to the Application layer have also been proposed in the literature, and a system-level power-saving methodology for heterogeneous wireless networks is in [4].

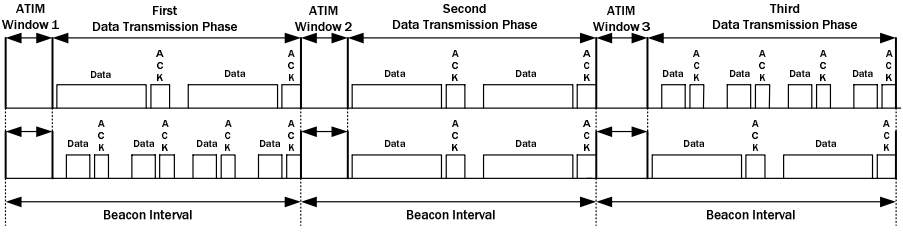


Fig. 1. The worst-case and best-case scenarios of power management in an 802.11 multiple rate ad hoc network

Because of signal fading, interference, shadowing, and path loss, etc., wireless channels have time varying characteristics. As a result, different wireless STAs may perceive different channel qualities at the same time. In order to obtain optimum throughput, STAs in the network need to use different transmission rates for different channel qualities [6]. But when 802.11 PSM is enabled in such a multiple rate ad hoc environment, we observe a problem of power management inefficiency which can be exemplified in Fig. 1. In this example, we assume there

are 16 STAs in the network, 8 of which are transmitters[1], and 8 of which are receivers. Each transmitter has only one packet to send to its receiver and all data packets are equal in length. In those transmitters, 4 of them are fast STAs, and the other 4 are slow STAs. Since fast (slow) STAs will use less (more) time in sending packets, the packets transmitted by fast (slow) STAs are represented by narrow (wide) rectangles in Fig. 1. According to the operations of 802.11 PSM, these transmitters must first announce their traffic in the ATIM Window and then use DCF to contend for the channel in the Data Transmission Phase. In the worst case, it may happen that all slow transmitters win the channel contentions before any fast transmitter has a chance to send data packet. Therefore as shown in the upper half of Fig. 1, the numbers of STAs that must stay in the active/power-saving state in the first, second, and third Data Transmission Phases are 16/0, 12/4, and 8/8, respectively. That is, 4 of the 16 STAs must stay in the active state for 2 Beacon Intervals, and 8 STAs must remain active for all of the 3 Beacon Intervals. In order to save power, we will propose a scheduling mechanism called STFS (Shortest Time First Scheduling) in this paper so that the packets transmitted on the channel can be as shown in the lower half of Fig. 1. This scheduling mechanism has the characteristic that it will schedule all fast transmissions or transmissions using less time to proceed before any of the slow STAs is allowed to send packet in every Data Transmission Phase. By scheduling in this way, more STAs can complete their data transmissions earlier and then go to power-saving state to conserve energy. Now the numbers of active/power-saving STAs are only 16/0, 8/8, and 4/12 in Data Transmission Phases 1, 2, and 3, respectively, the total power consumptions of these STAs are thus minimized.

In the above example, we assume each transmitter only has a specified number of data packets to send, therefore after a transmitter completes all its data transmissions, it will go to the doze mode; that is, the number of active transmitters in each Beacon Interval may decrease over time. By scheduling fast transmissions to proceed first, STFS can make this decrease more significant, so more power can be saved.

The rest of the paper is organized as follows. Section 2 describes the operations of the proposed scheduling mechanism. The performance of the STFS is investigated in section 3 and conclusions are given in section 4.

2 The Shortest Time First Scheduling

In STFS, we assume: (1) The WLAN is configured in its ad hoc mode; (2) An ideal channel condition without packet losses is considered; (3) The Beacon Intervals begin and end approximately at the same time at all STAs, so the problem of time synchronization is not considered; (4) Each STA in the network can support k data rates, $r_1 > r_2 > \cdots > r_k$, and has implemented an automatic rate selection protocol such as the RBAR in [5] which enables a receiver to select the most appropriate rate for its sender to use in the Data Transmission Phase;

[1] In this paper, a transmitter is a wireless STA that only transmit, not receive data packets.

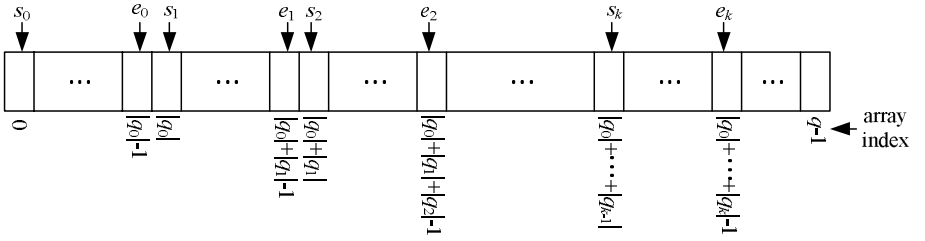
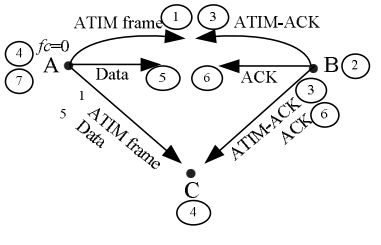


Fig. 2. The configuration of $k + 1$ queues in the scheduling array

and (5) The promiscuous mode of the wireless interface is enabled so that the interface can intercept and read each network packet that arrives in its entirety.

As we mentioned earlier, STFS will schedule all fast transmissions before any of the slow transmissions in every Beacon Interval. A major problem with this scheduling mechanism is starvation; that is, some of the slow STAs may have no chances to send packets when Data Transmission Phase can not accommodate all active transmissions. In order to achieve the goals of shortest time first and starvation prevention, we modify the packet formats of 2 control frames as follows: (1) The ATIM frame is extended with a 1-byte *aging* field; and (2) The ATIM-ACK is modified to include 2 additional 1-byte fields, *aging* and *rate*. The uses of these fields will be described in the following paragraph.

In addition to the above modifications, each STA in the network needs to maintain a local counter, fc. This counter has an initial value of 0. Whenever an STA has made a traffic announcement in an ATIM Window but fails to initiate transmission in the following Data Transmission Phase, fc is incremented by 1,



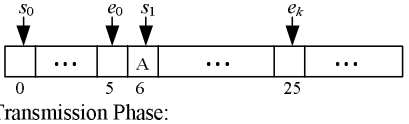
In ATIM Window:

Step1: A sends an ATIM frame with *aging* = 0 to B. This frame will also be received by C.

Step2: Based on the ATIM-frame' s signal strength, B then selects the most appropriate rate, e.g. r_1, for A to use.

Step3: B sends the ATIM-ACK with *DA* = 'A', *rate* = r_1, and *aging* = 0 back to A. This ATIM-ACK will also be heard by C.

Step4: A and C extract the contents from *DA, aging,* and *rate* fields of the ATIM-ACK and use these information to update their respective scheduling arrays. After the updates, the arrays at A and C may be like the following:



In Data Transmission Phase:

Step5: After 6 idle time slots, A sends a data packet to B.

Step6: B responds with an ACK.

Step7: A resets its backoff counter to $e_k + 1 = 25 + 1 = 26$.

Fig. 3. A simple STFS scheduling example

otherwise fc is reset to 0. Before an ATIM frame is sent, the transmitter will copy the value of fc to the *aging* field of the frame. After an ATIM frame is received, the rate selected by the receiver is sent back to it's transmitter through the *rate* field of the ATIM-ACK. The contents of the field *aging* in ATIM-ACK are coming from the same field of the received ATIM frame.

For the purpose of deciding packet transmission order in every Data Transmission Phase, a scheduling array of size q and a number of $2 \times (k+1)$[2] indexes, s_0, e_0, s_1, e_1, $\cdots$, s_k, e_k, also need to be maintained by each STA in the network. The size of this array is such that it can accommodate at least $k+1$ non-overlapping queues, q_0, q_1, $\cdots$, and q_k; that is: $|q_0| + |q_1| + \cdots + |q_k| \leq q$. The two ends, front and rear, of each q_i are pointed to by s_i and e_i, $0 \leq i \leq k$, respectively. The configuration of these queues in the array is shown in Fig. 2. Whenever an STA receives an ATIM-ACK, the STA will use the DA[3], *rate*, and *aging* fields of the frame to update its scheduling array as follows: (1) If $aging > 0$, the contents of DA will be put into q_0; and (2) If $aging = 0$ and $rate = r_i$, the contents of DA will be put into q_i, $1 \leq i \leq k$; that is, the addresses of all STAs with the local counter $fc = 0$ and using the same data rate will be put into the same queue in the scheduling array. The order of the station addresses in queue q_i, $1 \leq i \leq k$, is decided by the order of ATIM-ACK receptions, while the order in q_0 is determined as follows: The address in DA of ATIM-ACK$_1$ will have a smaller index value in q_0 than that in DA of ATIM-ACK$_2$ if (1) *aging* of ATIM-ACK$_1$ is larger than that of ATIM-ACK$_2$ or (2) *aging* of ATIM-ACK$_1$ is equal to that of ATIM-ACK$_2$ and *rate* of ATIM-ACK$_1$ is higher than that of ATIM-ACK$_2$ or (3) Both *aging* and *rate* of ATIM-ACK$_1$ are equal to those of ATIM-ACK$_2$ and ATIM-ACK$_1$ is received earlier than ATIM-ACK$_2$. For example, suppose an STA X receives 4 ATIM-ACKs with DA='A', *aging*=0, and *rate*=r_2 at time t, DA='B', *aging*=0, and *rate*=r_2 at time $t+1$, DA='C', *aging*=1, and *rate*=r_1 at time $t+2$, and DA='D', *aging*=2, and *rate*=r_2 at time $t+3$. Then, in the scheduling array of STA X, the address of STA A will have a smaller index value in q_2 than that of STA B, and the address of STA D will have a smaller index value in q_0 than that of STA C. When ATIM Window finishes, the array index values will be used by those STAs whose addresses are recorded in the scheduling array to setup the backoff counters to be used in data transmissions. Therefore all STAs whose addresses are in q_0 are permitted to send packets first, followed by the transmitters in q_1, and so on. Since the STAs whose addresses are in q_i will use a higher transmission rate than those whose addresses are in q_j, $1 \leq i < j \leq k$, the goal of shortest time first is achieved. Any STAs that had completed traffic announcements but failed to transmit data in the previous Beacon Interval(s) are recorded in q_0, so the starvation problem mentioned above is also solved. After a transmitter completes its data transmission, it will reset its backoff counter value to $e_k + 1$. This will give that transmitter chances to send multiple packets in the same Data Transmission Phase. After the current

[2] Recall that k is the number of different rates supported by STAs in the network.

[3] The Destination Address field, which now contains the address of the STA that transmitted the ATIM frame.

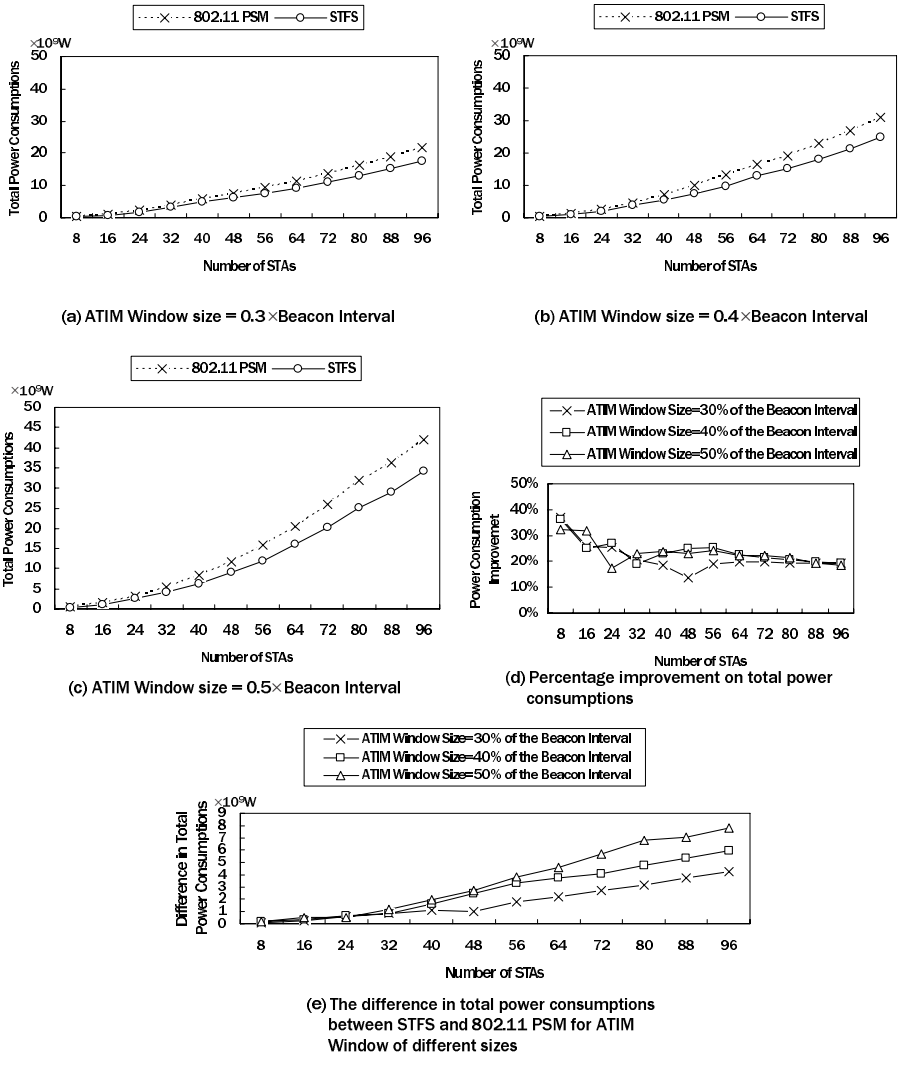


Fig. 4. Power consumption performance of STFS and 802.11 PSM with each transmitter having 1000 data packets to send

Beacon Interval terminates, the contents of the scheduling arrays maintained at all STAs are flushed to ensure the correct scheduling in the next Beacon Interval. A simple scheduling example of the STFS is shown in Fig. 3.

3 Performance of the STFS

We have developed a C++ based simulator to investigate the power consumptions of both STFS and 802.11 PSM. Since the ATIM Window size will

significantly affect the performance of 802.11 PSM [10, 11], we will vary that size to be 30%, 40%, and 50% of the Beacon Interval in each set of the simulations to see its effect on the performance of STFS. In this paper, we assume an STA will never be both a transmitter and a receiver at the same time. An 802.11b-based ad hoc network is particularly considered in our simulations, so the STAs in the network can support $k = 4$ different data rates, with $r_1 = 11.0$ Mbps, $r_2 = 5.5$ Mbps, $r_3 = 2.0$ Mbps, and $r_4 = 1.0$ Mbps. The rate used to send all control frames is 1 Mbps. In all simulations, we assume the numbers of transmitters that will use rate r_i, $1 \leq i \leq 4$, for data transmissions are equally distributed among all transmitters in the network. The size of the scheduling queue maintained at each STA is set to $q = 63$. The packet size at the MAC layer is fixed at 1024 bytes, and the lengths of the Beacon, ATIM, and ATIM-ACK frames for 802.11 PSM are 50, 28, and 14 bytes, respectively. The Beacon Interval is set to be 100 ms. For the energy model, a wireless STA will consume 1.65 W, 1.4 W, 1.15 W, and 0.045 W in the transmit, receive, idle, and the power-saving states, respectively [7, 8]. As in [9], the energy consumption for switching between awake and power-saving states is not considered in this paper. All simulation results are averages over 30 runs.

In our simulations, we measure the total power consumptions of all STAs for the case in which one half of the STAs are transmitters and each transmitter has 1000 data packets to send to its receiver. The results are shown in Fig. 4(a)∼(c). As we can see from the results, the total power consumed by all STAs in the network is less in STFS than in 802.11 PSM for all situations. The percentage improvement on total power consumptions, defined as $\frac{\text{TotalPowerConsumption}_{802.11PSM} - \text{TotalPowerConsumption}_{STFS}}{\text{TotalPowerConsumption}_{802.11PSM}}$, is shown in Fig. 4(d). We find a 20% to nearly 40% saving on energy is achieved by STFS. Finally, the results in Fig. 4(e) show that the savings on power consumption are more significant when the number of STAs in the network gets higher or the ATIM Window size becomes larger[4]. When these situations occur, more STAs will remain active in the same Data Transmission Phase, so the less chance they all can complete data transmissions. By scheduling fast transmissions first, STFS can send more packets in every Data Transmission Phase, therefore more STAs can complete their transmissions earlier and then go to power saving mode to conserve energy.

4 Conclusions

WLANs are usually designed for mobile applications. In mobile applications, battery power is one of the critical issues that must be dealt with. Due to limited battery power, various energy efficient protocols have been proposed to reduce wireless station's power consumptions in the literature. 802.11 addresses this power issue by allowing wireless stations to go into power-saving state at

[4] When ATIM Window size gets larger, the Data Transmission Phase will become shorter for Beacon Intervals with fixed length.

appropriate times to save power. However, this Power Saving Mechanism proposed by 802.11 has the problem of power management inefficiency when used in a multiple rate ad hoc network.

In this paper, a novel scheduling mechanism, STFS, is proposed to solve the above problem. The main idea of STFS is to schedule as many wireless stations to send packets as possible in every Beacon Interval so that they can complete their data transmissions earlier and then go to power-saving state to conserve energy. Simulation results show that the improvements made by STFS are significant and obvious in all situations.

References

[1] IEEE Std 802.11-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (1999)
[2] Karl, H.: An Overview of Energy-Efficiency Techniques for Mobile Communication Systems, Technical Report TKN-03-017, Telecommunication Networks Group, Technische University, Berlin (September 2003)
[3] Jones, C.E., Sivalingam, K.M., Agrawal, P., Chen, J.C.: A Survey of Energy Efficient Network Protocols for Wireless Networks, Wireless Networks, pp. 343–358 (July 2001)
[4] Simunic, T.: Power Saving Techniques for Wireless LANs. In: Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition, vol. 3, pp. 96–97 (2005)
[5] Holland, G., Vaidya, N., Bahl, P.: A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks. In: Proceeding ACM MOBICOM, pp. 236–251 (July 2001)
[6] Ci, S., Sharif, H.: A Variable Data Rate Scheme to Enhance Throughput Performance of Wireless LANs. In: Proc. IEEE Int. Symp. on Communication Systems, Networks and Digital Signal Processing, pp. 160–164 (2002)
[7] Lucent, IEEE802.11 WaveLAN PC Card - Users Guide, p. A-1
[8] Stemm, M., Katz, R.H.: Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. IEICE Transactions on Communications, special Issue on Mobile Computing, 1125–1131 (1997)
[9] Kim, D.-Y., Choi, C.-H.: Adaptive Power Management for IEEE 802.11-Based Ad Hoc Networks. In: Procceedings of the 5th World Wireless Congress, San Francisco (May 2004)
[10] Woesner, H., Ebert, J.-P., Schlager, M., Wolisz, A.: Power- Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective. IEEE Personal Communications 5(1), 40–48 (1998)
[11] Jung, E.-S., Vaidya, N.H.: An Energy Efficient MAC Protocol for Wireless LANs. In: INFOCOM, pp. 1103–1112 (2002)

Energy Efficient Scheduling for Real-Time Systems with Mixed Workload*

Jheng-Ming Chen, Kuochen Wang, and Ming-Ham Lin

Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
jmchen@cs.nctu.edu.tw, kwang@cs.nctu.edu.tw,
travelaman958@cs.95g.nctu.edu.tw

Abstract. In spite of numerous inter-task dynamic voltage scaling (DVS) algorithms of real-time systems with either periodic tasks or aperiodic tasks, few of them were aimed at the mixed workload of both kind of tasks. A DVS algorithm for mixed workload real-time systems should not only focus on energy saving, but also should consider low response time of aperiodic tasks. In this paper, we develop an on-line energy efficient scheduling, called *Slack Stealing for DVS* (SS-DVS), to reduce CPU energy consumption for mixed workload real-time systems under the earliest deadline first (EDF) scheduling policy. The SS-DVS is based on the concept of slack stealing to serve aperiodic tasks and to save energy by using the dynamic reclaiming algorithm (DRA). Unlike other existing approaches, the SS-DVS does not need to know the workload and the worst case execution time of aperiodic tasks in advance. Experimental results show that the proposed SS-DVS obtains better energy reduction (17% ~ 22%) while maintaining the same response time compared to existing approaches.

Keywords: mixed workload real-time system, inter-task dynamic voltage scaling, slack time, actual workload, worst case-execution time.

1 Introduction

In order to conserve energy for battery-powered real-time systems, some techniques were proposed in the past. Such as shutting down systems parts while they are not in use is one of the techniques for portable devices. However, restarting the hardware takes time and increases the response time. It's not effortless to determine when and which device should be shut down and woken up [8]. Another approach, called dynamic voltage scaling (DVS), to conserve power is by scaling down the processor voltage and frequency when some unused idle periods exist in the schedule at run time. The voltage scheduler determines which voltage to use by analyzing the state of the system. That is, the voltage scheduler of the real-time system supplies the lowest

* This work was supported by the NCTU EECS-MediaTek Research Center under Grant Q583 and the National Science Council under Grant NSC96-2219-E-009-012.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 33–44, 2007.

possible level voltage without affecting the system performance. Several commercially available processors provide the DVS feature, including Intel Xscale [11] and Xeon [12], Transmeta Crusoe [13], AMD Mobile Athlon [14], and IBM PowerPC 405LP [15].

It is known that the energy consumption E of a CMOS circuit is dominated by its dynamic supply voltage and is proportional to the square of its supply voltage, which is defined as $E = C_{eff} \cdot V_{dd}^2 \cdot C$ [10], where C_{eff} is the effective switched capacitance, V_{dd} is the supply voltage, and C is the number of execution cycles. Degrading the supply voltage also drops the maximum operating frequency proportionally ($V_{dd} \propto f$). Thus E could be approximated as being proportional to the operating frequency squared ($E \propto f^2$). Therefore, lowering operating frequency and according supply voltage is an effective technique for reducing energy consumption. However, reduction of the operating frequency leads to long service time. For this reason, applying DVS algorithms for real-time tasks to reduce energy consumption should still meet all requirements of real-time systems. For hard real-time tasks, DVS algorithms which lower operating frequency have to ensure no task missing its deadline. In the same way, DVS algorithms have to ensure reasonable response time of soft real-time tasks while reducing the operating frequency.

Despite several obvious advantages by using DVS algorithms, it also causes more preemptions which increase energy consumption in memory subsystems, and extra energy consumption and extra time for voltage transitions. However, these overheads have been generally ignored because the overhead can be included into the worst case execution time (WCET) of a task [3] [16]. Thus, a DVS algorithm can be used without modifications for real variable-voltage processors [3]. Additionally, [17] provides a technique that takes the task preemption into account while adjusting the supply voltage using the delayed preemption technique. The DVS algorithms have been proposed in growing numbers to minimize energy consumption in the past decade. In [9], it classifies existing DVS algorithms for real-time systems into two categories. One is *intra-task DVS* algorithms, which uses the slack time when a task is predicted to complete before its WCET. The other is *inter-task DVS* algorithms, which allocates the slack time between the current task and the following tasks. The basic difference between them is that intra-task algorithms adjust the supply voltage during an individual task boundary, while inter-task algorithms adjust the supply voltage task by task.

In this paper, we consider inter-task DVS scheduling. Most of the existing inter-task DVS algorithms were targeted at periodic tasks, and could get all tasks information in advance including arrival time, deadline, and WCET at the maximum processor speed. However, practical real-time applications involve both periodic and aperiodic tasks. For instance, in multimedia applications such as MPEG players, some tasks such as decoding frames periodically have stringent periodic performance, and some aperiodic user requests (e.g., volume control) should be with reasonable response times [3]. Periodic tasks are time driven with absolute hard deadlines, in general, and aperiodic tasks are event driven with soft deadline. Moreover, a portion of aperiodic tasks could not know the actual workload in advance [7].

2 Related Work

2.1 Aperiodic Real-Time Task Scheduling Schemes

Two schemes to serve aperiodic and periodic tasks in real-time systems, which were used in the proposed approach, are described:

(1). Bandwidth Preserving Server [18][19][20][21]: It is similar to the polling server. The concept of the bandwidth preserving server is creating a server with execution budget which is a time amount for executing aperiodic tasks. It is also characterized by an ordered pair (Q_s, T_s). Q_s/T_s is the server utilization. The difference from the polling serve is that it serves aperiodic tasks anytime while the budget isn't zero. It will execute aperiodic tasks according to the budget. If the budget is exhausted, it will stop or delay serving the aperiodic tasks.

(2). Slack Stealing [5]: The slack stealing is executing aperiodic tasks by using the available slack times of periodic tasks. If there is available slack time from periodic tasks, aperiodic tasks could be serviced first without causing any deadline miss of period tasks. In this scheme, the response time of aperiodic tasks is the lowest, but the complexity of such a real-time system is the highest among the possible approaches.

2.2 On-Line Inter-task DVS Strategies for Period Tasks

Two on-line inter-task DVS strategies for periodic tasks in real-time systems, which were used in the proposed approach, are depicted [9]:

(1). Stretching-to-NTA [16]: This strategy is based on that the scheduler already knows the next task arrival time (NTA) of periodic tasks. The scheduler will stretch the execution time to the NTA, if it doesn't cause deadline miss in this way. Therefore, the operating frequency and supply voltage can be decreased.

(2). Priority-Based Slack Stealing [2]: Because not all the execution time of tasks are in the worst cases, the slack time remains on the schedule if high priority tasks complete earlier than their WCETs. Consequently, the allowed execution time of low priority tasks can be extended.

Note that most DVS algorithms used these strategies for real-time systems with periodic tasks only, and directly using these algorithms for mixed workload real-time systems is not appropriate. If we directly use the stretching to NTA for mixed workload real-time systems, it is hard to know the next arrival time of an aperiodic task. As a result, there will be deadline miss of hard real-time periodic tasks when high priority aperiodic tasks abruptly arrive during the stretching period.

In the priority-based slack stealing strategy [2] and the utilization updating [22] strategy, although getting the slack time of a periodic task is easy, it's not easy to decide the slack time of an aperiodic task. Especially, we even don't know the arrival time and the WCET of each aperiodic task ahead. If utilizing the slack time from an aperiodic task is too aggressive or the actual workload of aperiodic tasks is higher than the predicted processor utilization, the deadline miss will occur just like that occurs in the stretching to NTA. Therefore, for mixed workload real-time systems, we

should adapt these strategies to satisfy the timing constraints of periodic tasks and the short response time requirement of aperiodic tasks. That is, we need to modify the on-line DVS algorithms for periodic tasks and integrated them with the previous mentioned aperiodic real-time tasks scheduling schemes.

2.3 Dynamic Reclaiming Algorithm

The Dynamic Reclaiming Algorithm (DRA) [2] is a kind of priority-based slack stealing technique, which is based on detecting early completions and adjusting the speeds of periodic tasks in order to provide additional power saving while still meeting the deadlines of periodic tasks. The DRA is the basis of most existing DVS algorithms for mixed workload real time systems. First of all, the optimal constant speed $\overline{S}$ could be calculated [2][22], at which speed every instance completes its worst case before the deadline. However, if the task finishes earlier than the worst case, the amount of remaining CPU time the dispatched task can safely use to reduce its speed. The DRA keeps and updates a data structure, α-queue, to calculate the earliness belonging to the executing tasks at run time. The information of the α-queue includes the identity, arrival time, deadline and remaining execution time rem_i of each periodic task T_i. The rem_i field of the head of the α-queue decreases with a rate equal to that of the passage of time. The DRA is under the EDF* policy. EDF* is almost the same as EDF. The difference is that in EDF* among the tasks whose deadlines are the same, the task with the earliest arrival time has the highest priority (a FIFO policy). Among the tasks whose deadlines and arrival times are the same, the task with the lowest index has the highest priority. The key notation for the DRA is as follows [2]:

- S^{can} : The canonical schedule in which each periodic task finishes before its deadline
- $\hat{S}_i$: The nominal speed of periodic task T_i
- $rem_i(t)$: The remaining execution time of periodic task T_i at time t in S^{can}
- $w_i^S(t)$: The remaining WCET of periodic task T_i under speed S at time t in the actual schedule
- $\varepsilon_i(t)$: The earliness of periodic task T_i at time t in the actual schedule, defined as :

$$\varepsilon_i(t) = \sum_{j|D_j^* < D_i^*} rem_j(t) + rem_i(t) - w_i^{S_i}(t)$$
$$= \sum_{j|D_j^* \le D_i^*} rem_j(t) - w_i^{S_i}(t)$$

where D_i is the deadline of T_i

Note that the nominal speed is the default speed it has whenever it is dispatched by the operating system prior to any dynamic adjustment.

2.4 Existing Inter-task DVS Algorithm for Mixed Workload Real-Time Systems

Recently, several researchers proposed DVS algorithms for mixed workload real-time systems. Under the EDF (or EDF*) scheduling policy, most of these algorithms integrate the bandwidth preserving server and priority-based slack stealing strategies. Doh et al. [6] proposed an approach which leads to proper allocation of energy budgets for hard periodic and soft aperiodic real-time tasks. Given an energy budget, it computes a proper voltage setting for attaining an improved performance for aperiodic tasks while meeting the deadline requirements of periodic tasks. It used TBS (total bandwidth server) [18], which is a kind of bandwidth preserving servers, and only focused on the off-line static scheduling problem. Aydin et al. proposed three separate on-line schemes with mixed workload under a power consumption constraint. It also used TBS and DRA [2] under the EDF* scheduling policy. In the Basic Reclaiming Scheme (BRS) [1] the earliness of aperiodic tasks is only used for reclaiming the coming aperiodic tasks, and the earliness of periodic tasks is only used for reclaiming the coming periodic tasks. The Mutual Reclaiming Scheme (MRS) [1] was developed from BRS. The main difference between MRS and BRS is that in the MRS both periodic and aperiodic tasks can mutually reclaim their unused computation times. The Bandwidth Sharing Scheme (BSS) [1] is to solve the problem of the actual aperiodic workload that is relatively lower than the predicted aperiodic workload. In BSS when TBS is idle, the algorithm will create a ghost job *J* to produce more earliness to aggressively reduce the operating speed. But it will increase the response time of aperiodic tasks if an actual aperiodic task arrives right after creating the ghost job.

In [4], Shin et al. merged the TBS and two DVS algorithms, lppsEDF [16] and DRA, respectively, under the EDF* scheduling policy. They also proposed an enhanced approach called Workload-based Slack Estimation (WSE) [3], which integrates CBS [19] and DRA. The WSE is almost the same as the MRS (as indicated in [3]), except that it uses C_{slack} to stretch the execution time of periodic tasks by using the slack time from CBS and it guarantees the constrained response time. But C_{slack} decreases to zero rapidly as aperiodic tasks arrive. In this case, the slack time of CBS is wasted. Note that both the MRS and WSE need to know the workload of aperiodic tasks in advance. In addition, the MRS needs to know the WCETs of aperiodic tasks. However, in most cases the workload of aperiodic tasks of some real-time systems is with large variance or unpredictable [7]. If the given budget of the bandwidth preserving server is not suitable for the current schedule point, it will waste unnecessary energy consumption or result in long response time. Another problem is that a periodic task with the highest priority may run slowly even if there are some aperiodic tasks waiting in the queue.

3 System Model, Assumptions and Notations

The target processor can change its supply voltage (*V*) and operating speed (*S*) (or frequency) continuously within its operational ranges, $[V_{min}, V_{max}]$ and $[S_{min}, S_{max}]$. There are two components of mixed workload real-time systems: a set of $T = \{ T_1 \ldots T_n \}$ of *n* periodic tasks with hard deadlines, and a set of *J* aperiodic tasks

arriving randomly with soft deadlines. Based on related work [1][3][4][5][18][19][20][21], the arrival time and the worst case requirements of periodic tasks are known in advance, but those of aperiodic tasks are made available only when they arrive. The relative deadline of each periodic task instance is assumed equal to its period. All tasks are assumed to be independent. We used the following notation:

- $T_{i,j}$: The j^{th} instance of T_i
- C_i : The worst case CPU execution cycles per instance of T_i
- D_i : The deadline of T_i
- J_i : The i^{th} aperiodic task
- r_i : The arrival time of J_i
- d_i : The deadline of J_i
- f_i : The finish time of J_i
- E_a : The average CPU execution cycles per aperiodic task
- N_a : The number of aperiodic tasks in the queue
- A_i : The largest amount of aperiodic processing possible of J_i

4 Proposed SS-DVS Algorithm

In this paper, we propose an on-line DVS algorithm for mixed workload real-time systems, named *Slack Stealing for DVS* (SS-DVS). The SS-DVS doesn't need to know the workload and the WCET of aperiodic tasks. It used the concept of slack stealing, which was originally used in mixed workload real time systems without DVS, to service aperiodic tasks. It used the DRA to serve aperiodic tasks and to reclaim the operating speed. In SS-DVS, all allowed execution time belongs to periodic tasks. In this way, the periodic tasks can execute slowly to save energy. It will increase the operating frequency once aperiodic tasks arrive and serve the aperiodic tasks as soon as possible. Although the slack stealing approach was considered to have high computation overhead to derive the slack time of the schedule to service aperiodic tasks, the proposed SS-DVS has low computation overhead compared to other approaches. The reason is that the SS-DVS obtains slack time from the periodic tasks completed earlier or generates slack time by raising the operation speed of the current task. That is, the SS-DVS not only collects the slack time for reclaiming the speed to save energy, but also for servicing aperiodic tasks to reduce their response time.

The basic idea of the proposed SS-DVS is using the slack time to reduce the operating speed of periodic and aperiodic tasks and also using the slack time to service aperiodic tasks based on the DRA. If no aperiodic task arrives, obtaining low energy consumption is to operate periodic tasks with low speed. And if any aperiodic task arrives, it uses the collected slack time to service aperiodic tasks. First we have to construct a data structure call *α'-queue* which is almost the same as α-queue with an additional flag, *ModeFlag*, in the α'-queue. The *ModeFlag* records the nominal speed of each periodic task. If *ModeFlag* is set to *H*, it means the nominal speed is S_H and

if *ModeFlag* is set to *L*, it means the nominal speed is S_L. There are four operations in our algorithm.

(1). Setup: Compute the static optimal speed $\overline{S}$. Then set S_L, to a value large than or equal to $\overline{S}$, and set S_H to a value between $S_{\max}$ and S_L. Then, initialize the α'-queue to the empty list and set the nominal speed of periodic tasks to S_L.

(2). Basic stealing: Using the slack time to service aperiodic tasks is the basic idea of SS-DVS. First of all, when an aperiodic task arrives at the active state, it means that there are pending aperiodic tasks in the queue at time t and there already exists at least one task J_i in the queue such that $r_i \le t < f_i$. Otherwise, it is at the idle state. When a task J_y arrives at the active state, the task is placed in a queue of pending tasks according to the FIFO, and the allowed execution time and the deadline of J_y are set according to the preceding task J_{y-1}. When a task J_y arrives at the idle state and no periodic tasks being serviced or in the queue, the deadline of J_y is set to NTA, and the allowed execution time of J_y is set to $NTA - t$. As a task J_y arrives at the idle state and some periodic tasks are being serviced or in the queue, the deadline of J_y depends on the earliness of periodic task T_x with the highest priority. Hence the deadline of J_y is $t + \varepsilon_x(t)$, where $\varepsilon_x(t)$ is the earliness of periodic task T_x at time t ,and A_x is set to $\varepsilon_x(t)$. For example, in Fig. 1(a), when a task J_y arrives at the idle state and there are no other periodic tasks in the queue, the A_x of J_y is $NTA - t$. And in Fig. 1(b), when a task J_y arrives at the idle state and periodic task T_x is executing, the A_x of J_y is set to $\varepsilon_x(t)$.

(3). Aggressive stealing: The speed S_L is the nominal speed of periodic tasks. However, if we execute a periodic task at S_L and the actual execution time of the periodic task is in its worst case, there will be no earliness left. Consequently, the speed is raised to S_H to create more slack time for serving aperiodic tasks. In Fig. 1(c), J_y arrives at t_1, but there is no earliness of T_x left. Therefore, the operating speed is changed to S_H to create more slack time. As we can see, T_x finishes at t_2 and the slack time is available

(4). Swapping stealing: In the aggressive stealing, a periodic task executes at high speed S_H and produces some slack time. However, we still have to service periodic tasks first. Therefore, we could more aggressively service aperiodic tasks first to reduce the response time without causing any deadline miss of hard real-time periodic tasks. Swapping the order of periodic and aperiodic tasks not only can reduce the response time, but also could lower the speed of periodic tasks if aperiodic tasks completes early than expects. As shown in Fig. 1(d), we change the order of execution between T_x and J_y.

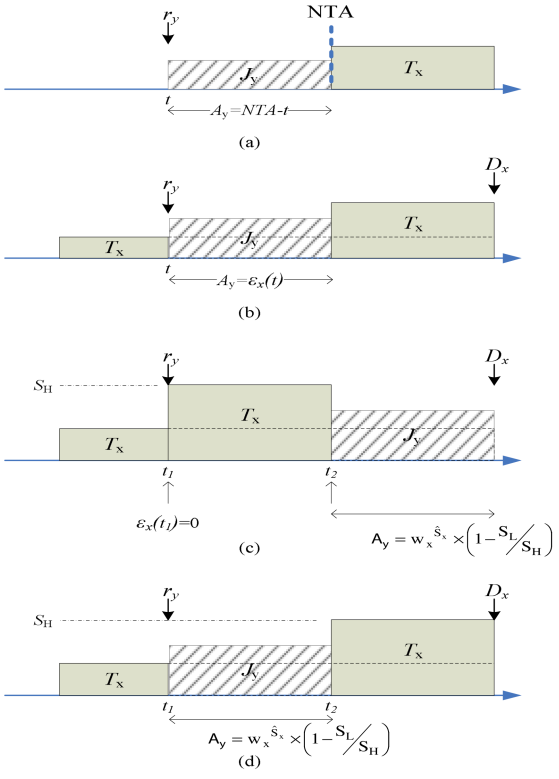


Fig. 1. The SS-DVS scheme (a) the aperiodic task arrives with no other periodic task in the queue (b) the aperiodic task arrives with some periodic tasks in the queue, and the highest priority task has some earliness (c) aggressive stealing(d) swapping stealing

5 Simulation Results

5.1 Simulation Model

We use the same simulation environment as that of WSE [3]. Aperiodic tasks were generated by the exponential distribution using with inter arrival time (1/λ) and service time (1/μ) with parameters λ and μ. We used a fixed value μ and varied λ to control the workload (ρ = λ/μ) of aperiodic tasks under a fixed utilization U_p of periodic tasks [3]. There are four periodic tasks in Table 1. The period of each task is 6, 8, 14, and 18, respectively, and the WCET of each task is 0.5, 1.0, 2.1, and 3.1, respectively. The utilization U_p of periodic tasks is 0.4 ((0.5 / 6) + (1.0 / 8) + (2.1 / 14) + (3.1 / 18)).

The actual execution time of each periodic task instance was generated by a normal distribution function in the range of [BCET, WCET], where BCET is the best-case

Table 1. Periodic task set descriptin

Task Set (millisecond)		
Task	Period	WCET
T_1	6	0.5
T_2	8	1.0
T_3	14	2.1
T_4	18	3.1
U_p	0.4	

execution time. The mean and the standard deviation were set to (WCET+BCET)/2 and (WCET-BCET)/6, respectively [1]. In the experiments, the voltage scaling overhead was assumed negligible both in the time delay and power consumption. For easiness to compare with other approaches, we used the fixed S_L and S_H in this experiments. S_L is 60% of the maximal speed (must be larger than or equal to the static optimal speed) and S_H is the maximal speed.

In order to experimentally evaluate the performance of the proposed algorithm, SS-DVS, we implemented the following schemes for performance evaluation: (1) PD/CBS [3]: Aperiodic tasks are serviced by CBS (constant bandwidth servers). It was assumed that if the system is idle, it enters into the power-down mode (PD). The power consumption in the PD mode is assumed to be zero. (2) Mutual Reclaiming Scheme (MRS) [1]: We set S_a (the nominal speed of aperiodic tasks) equal to S_p (the nominal speed of periodic tasks). (3) Workload-based Slack Estimation scheme (WSE) [3]. (4) Bandwidth Sharing Scheme (BSS) [1]: We set S_a equal to S_p.

In the following, the average energy consumption and response time are normalized to those of PD/CBS, under different conditions.

5.2 Effect of the Workload of Aperiodic Tasks on Energy Consumption and Response Time

BCET is assumed to be 10% of WCET, and ρ is ranging from 0.05 to 0.25 ($\lambda = 0.05 \sim 0.25$ and $\mu = 1.0$) [3]. As shown in Fig. 2, by different workload of aperiodic tasks, SS-DVS is compared with others approaches under varied server utilization. Fig. 2(a) shows that SS-DVS has lower normalized energy consumption than that of the other three existing algorithms, and it also has comparable normalized response time with MRS and WSE, as shown in Fig. 2(b). Although BSS has less energy consumption than SS-DVS when the server utilization is close to the workload of aperiodic tasks, BSS has the highest response time. The normalized energy * response time [1] is a performance metric that combines the two important dimensions, energy consumption and response time, of the mixed workload real-time systems. As Fig. 2 (c) shown, SS-DVS has better performance than the three other approaches in terms of normalized energy * response time.

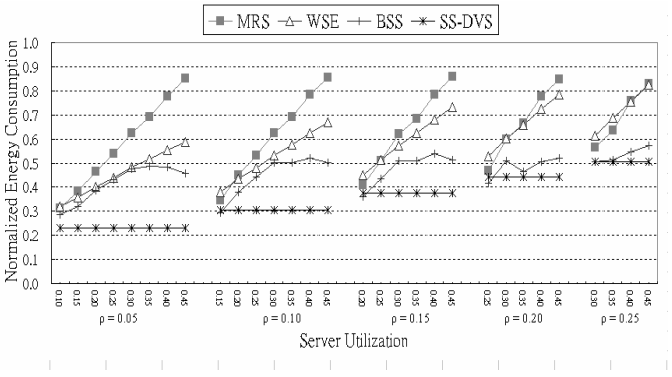


Fig. 2(a)**.** Normalized energy consumption

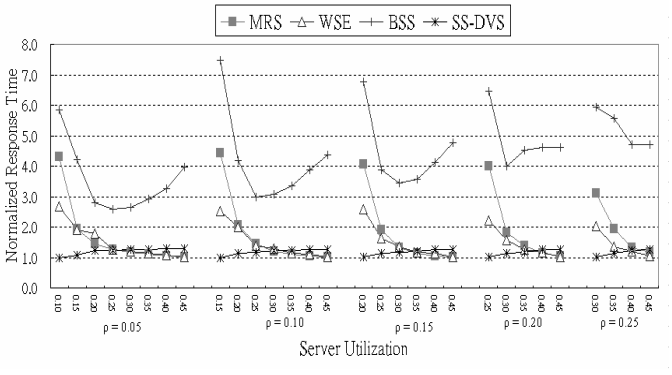


Fig. 2(b)**.** Normalized response time

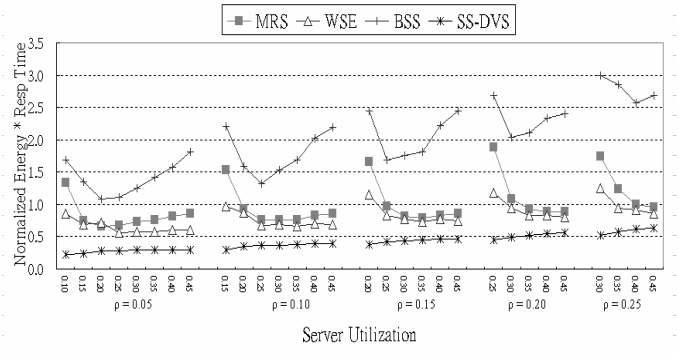


Fig. 2(c)**.** Normalized energy * response time

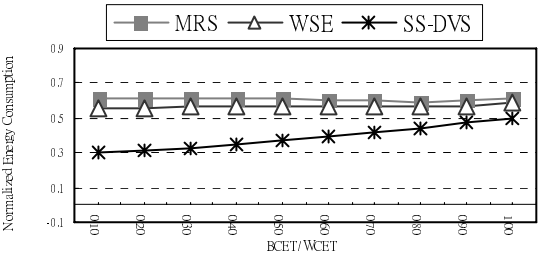


Fig. 3. The normalized energy consumption base on same response time with different BCET/WCET

5.3 Effect of BCET/WCET Ratio of Periodic Tasks on Energy Consumption

The workload of aperiodic tasks was set to 0.1 (λ = 0.1 and μ = 1.0), and the BCET/WCET ratio was from 0.1 to 1.0 with an increment of 0.1 0. Fig. 3 shows the normalized energy consumption based on the same response time of all algorithms with different BCET/WCET ratios. We have the following observations:

- The normalized energy consumption of SS-DVS increases as BCET/WCET increases. This is because aperiodic tasks are served by the slack time from periodic tasks. The less slack time from periodic tasks may cause the periodic and aperiodic tasks to run at a higher speed.
- SS-DVS reduces the energy consumption by an average of 21% and 17% compared with the MRS and WSE algorithms, respectively.

6 Conclusion

In this paper, we have presented an on-line dynamic voltage scaling (DVS) algorithm, called SS-DVS, for mixed workload real-time systems. SS-DVS not only addresses the energy consumption for mixed workload real-time systems, but also considers the response time of aperiodic tasks. SS-DVS integrates the slack time stealing concept to service aperiodic tasks and the dynamic reclaiming algorithm (DRA) to set a suitable operating speed. SS-DVS can use the slack time more efficiently than existing approaches, because it doesn't reserve any time for aperiodic workload. Simulation results have shown that SS-DVS can effectively reduce the average energy consumption by 48%, 22%, 18% compared with the PD/CBS, MRS, and WSE algorithms, respectively, under the same response time.

References

1. Aydin, H., Yang, Q.: Energy-responsiveness tradeoffs for real-time systems with mixed workload. In: Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 74–83 (2004)
2. Aydin, H., Melhem, R., Moose, D., Mejia-Alvarez, P.: Power-aware scheduling for periodic real-Time tasks. IEEE Transactions on Computers 53(5), 584–600 (2004)

3. Shin, D., Kim, J.: Dynamic voltage scaling of mixed task sets systems in priority-driven systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25(3), 438–453 (2006)
4. Shin, D., Kim, J.: Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In: Proceedings of Asia and South Pacific Design Automation Conference, pp. 653–658 (2004)
5. Lehoczky, J.P., Ramos-Thuel, S.: An optimal algorithm for scheduling soft-aperiodic tasks in fixed priority preemptive systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 110–123 (December 1992)
6. Doh, Y., Kim, D., Lee, Y.-H., Krishna, C.M.: Constrained energy allocation for mixed hard and soft real-time tasks. In: Proceedings of 9th International Conference on Real-Time and Embedded Computing Systems and Applications, pp. 533–550 (2003)
7. Rusu, C., Ruibin, X., Melhem, R., Mosse, D.: Energy-efficient policies for request-driven soft real-time systems. In: Proceedings of Euromicro Conference on Real-Time Systems, pp.175–183 (July 2004)
8. Benini, L., Bogliolo, A., De Micheli, G.: A survey of design techniques for system level dynamic power management. IEEE Transactions on Very Large Scale Integration Systems 8(3), 299–316 (2000)
9. Kim, W., Shin, D., Yun, H.S., Min, S.L., Kim, J.: Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In: Proceedings of the IEEE Real-Time and Embedded Technology and Application Symposium, pp. 219–228 (September 2002)
10. Moyer, B.: Low-power design for embedded processors. Proceedings of IEEE 89(11), 1576–1587 (2001)
11. Intel XScale® Technology, Intel. PXA270 processor electrical, mechanical, and thermal specification, http://www.intel.com/design/intelxscale/
12. Intel® Xeon® Processor, http://www.intel.com/products/processor/xeon/
13. Trasmeta Corporation, TN5400 processor specification, http://www.transmeta.com/crusoe/
14. Mobile AMD AthlonTM 64 processor, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_10220_10221,00.html
15. Carpenter, G.: Low power SOC for IBM's PowerPC information appliance platform, http://www.research.ibm.com/arl
16. Shin, Y., Choi, K.: Power conscious fixed priority scheduling for hard real-time systems. In: Proceedings of the Design Automation Conference, pp. 134–139 (1999)
17. Kim, W., Kim, J., Min, S.L.: Preemption-aware dynamic voltage scaling in hard real-time systems. In: Proceedings International Symposium Low Power Electronics and Design, pp. 393–398 (2004)
18. Spuri, M., Buttazzo, G.: Scheduling aperiodic tasks in dynamic priority systems. Journal of Real-Time Systems 10(2), 179–210 (1996)
19. Abeni, L., Buttazzo, G.: Integrating multimedia applications in hard real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 4–13 (1998)
20. Strosnider, J.K., Lehoczky, J.P., Sha, L.: The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Transactions on Computers 44(1), 73–91 (1995)
21. Sprunt, B., Sha, L., Lehoczky, J.P.: Aperiodic task scheduling for hard real-time systems. Journal of Real-Time Systems 1(1), 27–60 (1989)
22. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low power embedded operating systems. In: Proceedings of the ACM Symposium on Operating Systems Principles, pp. 89–102 (2001)

Function-Level Multitasking Interface Design in an Embedded Operating System with Reconfigurable Hardware

I-Hsuan Huang, Chih-Chun Wang, Shih-Min Chu, and Cheng-Zen Yang

Department of Computer Science and Engineering
Yuan Ze University, Taiwan, R.O.C.
{ihhuang,ken,csm,czyang}@syslab.cse.yzu.edu.tw

Abstract. Reconfigurable architecture provides a high performance computing paradigm. We can implement the compute-intensive functions into reconfigurable devices to optimize the application performance. In current reconfigurable hardware designs, the function-level reconfigurable hardware has high reusability and low maintenance cost. However, the sharing mechanism and the function invocation interface are still unknown. In this paper, we propose a function-level multitasking interface design to support reconfigurable component sharing in a multitasking embedded operating system. The reconfigurable hardware functions are managed and scheduled by the operating system. Applications can use any needed hardware function via invocation APIs. To study the performance impacts, we implemented a prototype on Altera SOPC development board. We modified μC/OS-II RTOS and evaluated the prototype with prime number search programs and loop programs. The experimental results show the management overhead is acceptable.

Keywords: Reconfigurable computing, multitasking, hardware function, FPGA-based computer, μC/OS.

1 Introduction

Reconfigurable computing provides a high performance computing paradigm [3,4,9]. In the current development, a general reconfigurable computer comprises one or several traditional microprocessors, and reconfigurable hardware devices. With hardware/software co-design, the reconfigurable computer may execute the compute-intensive tasks on the specific programmable devices. As the reconfigurable hardware can accelerate the task execution, system performance can be highly improved (e.g., [1,4,5]).

Adopting reconfigurable hardware has two most prominent benefits. First, overall system productivity can be highly promoted because traditional software-coded functions are accelerated with the reconfigurable computing hardware. Tasks can be thus parallelized with multiple computing engines. Second, the acceleration engine can be flexibly customized due to the reconfigurability. Therefore, the reconfigurable computer can adapt to different computation requirements with high performance.

From the aspect of accelerating granularity, the reconfigurable computing engines can be classified into three categories. The category of the finest accelerating granularity

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 45–54, 2007.

consists of instruction-level processing engines, such as the 2D-VLIW approach [7]. The reconfigurable device operates as a co-processor to execute the task instruction-by-instruction. The instruction-level processing engine approach thus benefits system performance with the expanded instruction set. This approach, however, may incur huge amount of data synchronization overhead between the reconfigurable devices and the general-purpose processors.

The acceleration granularity employed in the second category focuses on the task level [2,11]. In this task-level acceleration approach, tasks can be either software-coded or implemented as hardware units. Therefore, a software task unit can even have its own correspondent reconfigurable hardware version. While task execution is initiated, the system dynamically decides whether a hardware unit or a software unit is invoked. Since the granularity is at task level, the synchronization overhead between programmable devices and general-purpose processors is highly reduced. Nevertheless, the reusability of hardware task units is very low due to the functional specificity of each unit. Besides, the hardware space efficiency is low because the programmable device needs to maintain all hardware task units in its limited space. The management of hardware task units incurs extra overhead.

The function-level acceleration schemes fall into the third category [6,10]. The function-level processing engine maintains a consistent hardware interface as the programming interface of software functions. Application developer can follow the programming conventions to use the hardware functions. Due to the high modularity at the function level, hardware function units favor high reusability and low maintenance cost. Although the function-level acceleration approach can fully exploit the high-performance and flexibility of reconfigurable computing architecture, two main issues need to be further discussed: the sharing mechanism of function units and the multitasking interface design. To the best of our survey, previous studies on function-level processing engines mainly focus on performance optimization of specific functions, and rarely discuss the sharing and multitasking issues [6,10]. Current embedded systems, however, are mostly multitasking systems in which multiple tasks cooperate. To further improve system performance with consideration of the limited space of FPGA, a multitasking interface design providing hardware unit sharing is very crucial.

To support reconfigurable component sharing in a multitasking environment, the enhancements can be practiced in three possible layers: applications, operating system kernel, or reconfigurable hardware. For the following two reasons, we argue that OS kernel support is more superior to other two enhancements. First, if applications take the responsibility to maintain hardware function multitasking and sharing, they need to manage the control registers of hardware functions and maintain function invocations from other applications. Consequently, a task execution may be interfered with other task executions. Application design becomes more complicated and error-prone. Second, if the multitasking mechanism is implemented in reconfigurable hardware, it will occupy a large amount hardware space due to many bookkeeping data structures. Since the space resource is very precious in reconfigurable hardware, this approach incurs high cost/performance ratio in reconfigurable hardware utilization. Implementation of the multitasking mechanism in the OS layer can avoid both the error-prone development problem and the low hardware utilization problem. Although the OS layer enhancement

cannot be benefitted from hardware acceleration, its overhead of software-coded execution is comparatively small in the whole system. Accordingly, we propose a function-level multitasking interface design implemented in the OS layer to take advantage of high performance of reconfigurable hardware.

This paper presents the multitasking interface design in an embedded operating system with reconfigurable hardware. The proposed approach has three main design features. First, multiple tasks can coherently share the reconfigurable accelerator hardware with the OS support of the multitasking mechanism. Second, the invocation interface of the hardware units is consistent with the software library to ease application development. Third, if there are numerous tasks waiting for the same shared reconfigurable hardware, OS can dynamically direct the function invocation to the software-coded library. With these features, the system keeps the flexibility to process the application function calls with high performance.

The design of multitasking support for reconfigurable hardware is not straightforward because hardware unit sharing is very different with software library sharing. Two main issues need to be considered: parameter passing flow and data consistency. To maintain data consistency, a management module in OS is designed to deal with multiple invocations, and the OS has a specific job queue to schedule these invocations. In addition, each hardware function unit has its own invocation API in OS to pass parameters. Programmers can replace the compute-intensive function calls in the applications with the corresponding hardware function invocations. When the hardware function unit completes the job, the results are returned via an interrupt service routine. These two issues complicate the design of the multitasking interface.

We have implemented the proposed multitasking interface in an embedded OS μC/OS-II [12] to study the performance impacts. The prototype is based on an Altera SOPC (system-on-programmable-chip) development board [13]. Several hardware functions have been implemented in FPGA to verify the functionality of the multitasking interface. We also conducted preliminary experiments to evaluate the prototype. Although the current benchmark set is primitive, the experimental results show that the management overhead is acceptable and the application performance can be highly improved.

The rest of the paper is organized as follows. Section 2 reviews previous reconfigurable computer studies of different acceleration granularity. Section 3 elaborates the proposed function-level multitasking hardware interface. Section 4 presents the Altera SOPC prototype implementation, and the evaluation results in the experiments. Section 5 concludes the paper.

2 Related Work

The hardware accelerators on FPGA chips are also known as processing engines. From the aspect of accelerating granularity, the processing engines can be classified into three categories, the instruction-level, the task-level, and the function-level. The instruction-level processing engines are usually implemented as co-processors. For example, Santos, Azevedo, and Araujo presented an instruction-level reconfigurable architecture called the 2D-VLIW [7]. In the 2D-VLIW project, the processing engines are

controlled by 2D-VLIW instructions which are composed of multiple single operations. The processing engine needs one 2D-VLIW instruction for each execution cycle. Since processing engines are commanded according to 2D-VLIW instructions, the application developer needs to understand the details of the processing engine and to write the 2D-VLIW instructions in the application. Although the processing engine in the 2D-VLIW can be beneficial to applications, the reconfigurable computer needs to synchronize the processing engines and the general purpose processor frequently.

The task-level acceleration approach presents the second category. For example, Andrews et al. proposed the *hthreads* task-level reconfigurable architecture [2]. The *hthreads* is a multithreaded RTOS, which supports the software and the hardware threads it its thread model. Each hardware thread obtains an exclusive processing engine kept in the FPGA chip. The *hthreads* scheduler is responsible for managing the software threads and the hardware threads. Since data of hardware threads are private, the data synchronization can be largely reduced. However, the hardware thread maintenance are cumbersome. Besides, since the hardware thread is customized, the hardware thread can hardly be reused. Accordingly, the utilization of hardware threads is seriously degraded. Another example is the SHUM-μC/OS project proposed by Zhou et al. [11]. Zhou et al. modified the μC/OS-II RTOS and defined their hardware thread model. They implemented a hardware thread control block in the μC/OS-II to keep the data structures of hardware threads. The hardware threads in SHUM-μC/OS are also very difficult to be reused by other applications.

The function-level processing engines are implemented according to the basic algorithmic function blocks of applications. The function-level reconfigurable architecture provides a reusable, elegant, and high maintainability programming paradigm. For example, Rullmann, Siegel, and Merker proposed an application partitioner [6]. The application partitioner extracts compute-intensive algorithmic blocks. Then the compute-intensive algorithmic blocks are implemented as FPGA processing engines. Thus, the system performance can be improved. However, the processing engines are managed by specific applications and cannot be directly used by other applications. Another example is the ReConfigME function-level reconfigurable architecture proposed by Wigley, Kearney, and Jasiunas [10]. Since the processing engine still managed by specific applications, they are very difficult to be reused by other applications. Shibamura et al. also proposed a function-level reconfigurable platform called EXPRESS-1 in 2004 [8]. The major difference between EXPRESS-1 and our system is that EXPRESS-1 focused on the design of the reconfiguration procedure and our system focused on the hardware/software interface design.

3 Functional-Level Multitasking Interface Design

The idea of the function-level multitasking interface is to support reconfigurable component sharing in a multitasking reconfigurable computer. Since the processing engines of previous function-level reconfigurable architecture are managed by respective applications, the hardware functions are difficult to be reused by other applications. With the OS supported function-level multitasking interfaces, all applications can request to use public hardware functions.

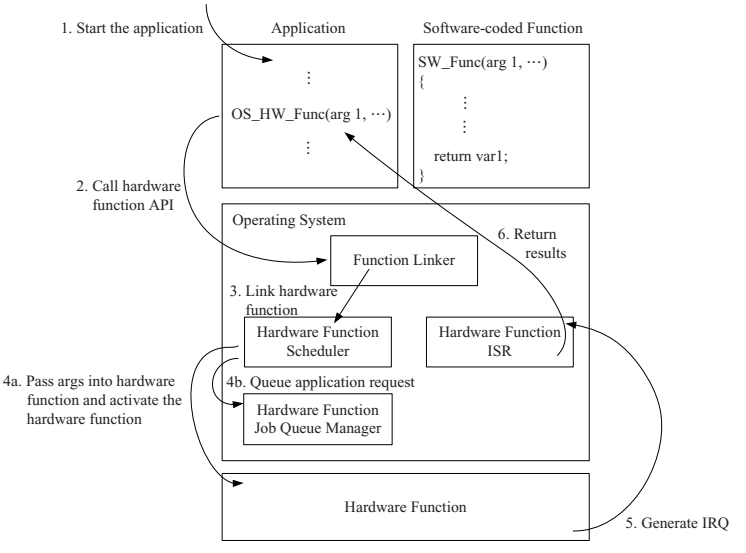


Fig. 1. Control flow of the hardware function invocation

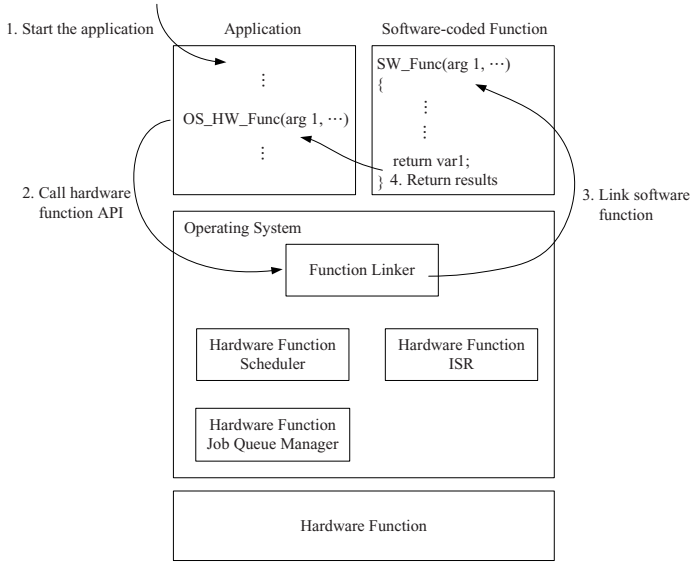


Fig. 2. Control flow when the hardware function job queue is full

Our proposed function-level multitasking architecture modifies four components in a traditional reconfigurable computer: (1) Operating system: We move the management responsibility of hardware functions to the operating systems. We develop a multitasking hardware function manager in the OS. The manager includes a function linker and a

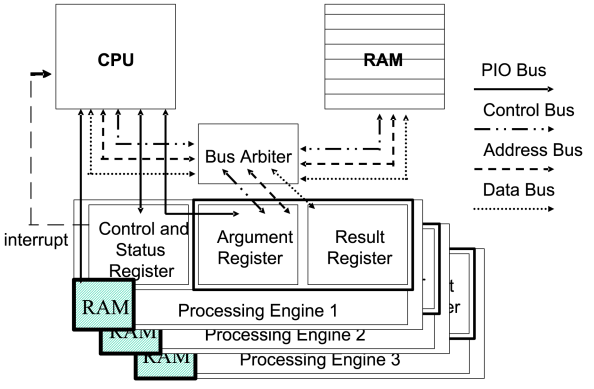


Fig. 3. Hardware function structure

scheduler. Besides, we develop a job queue for each hardware function. We provide a multitasking invocation API for each hardware function. (2) Task: The compute-intensive functions can be replaced with the multitasking invocation API. The interface of function parameters and the return results is consistent with the software library to ease application development. (3) Processing engine: In the beginning, each hardware function has to register in the OS. The OS creates the job queue, the function-level multitasking interface, and the interrupt service routine (ISR) for each hardware function. When the hardware function is executing, the hardware function selects new job from the job queue in the OS. When the hardware function finishes a job, it issues an IRQ to the general purpose processor. (4) Software-coded function: Our architecture keeps the software-coded functions. OS can dynamically direct the function invocation to the software-coded library.

Figure 1 shows the control flow of the proposed function-level multitasking architecture. Suppose the hardware function has already been registered. Since the SW_Func() is a compute-intensive function, we implement the hardware version function in FPGA chip. At the same time, we replace the original function call with the hardware function invocation interface called OS_HW_Func(). The control flow is as follows. First, the OS executes the application. Meanwhile, the application is executed by the general purpose processor. Second, the application invokes the hardware function. After the application calls the interface, the OS forces the application enter the waiting queue. Third, After the application passes the parameters to the OS, if the job queue still has free spaces, the function linker forwards the parameters to the hardware function scheduler. Fourth, the hardware function scheduler checks the status of the requested hardware function. If the hardware function is available to be executed, the hardware function scheduler passes the parameters into the hardware function. If the hardware function is busy, the hardware function scheduler saves the hardware function request into the job queue. Fifth, after the hardware function completes one job, the hardware function issues an IRQ. The general purpose processor then jumps to the hardware function ISR to grab computation results of the hardware function. Besides, the ISR configures the hardware

function to select next job from the job queue. Sixth, the ISR passes the results to the application. The OS then forces the application enter the ready queue.

Figure 2 presents the control flow when OS dynamically direct the function invocation to the software-coded library. The difference is the third step. When the OS receives the hardware function request, OS jumps to the function pointer of the software-coded function. The parameters are also passed to the software-coded function. Fourth, after the software-coded function finishes the computation, it returns the results to the application. In this case, all operations are executed by the general purpose processor.

Figure 3 shows the hardware function structure used in our architecture. The hardware function is implemented in hardware description language like VHDL or Verilog. In the figure, we denote the hardware function as the processing engine. Each processing engine contains at least four registers, the control register, the status register, the argument register, and the result register. The size of each register is varied according to the application demands. The OS enables and disables each processing engine via the control register. Since the OS has to manage the processing engine, it monitor the condition of processing engine through the status register. The argument register records the parameters of function request. The result register keeps the computation results of the processing engine. Since the parameters and the results may be a pointer variable or a data structure, the processing engine can directly connect to the memory bus. If the parameter is a pointer variable, the processing engine directly access the memory address to capture the variable value. If the processing engine has to access the global variable, it also directly access the memory address of the global variable. When the processing engine completes one job, it issues an interrupt to the general purpose processor. The general purpose processor then executes the ISR to retrieve the computation results. Sometimes, the processing engine needs to call an external function. To solve this issue, the processing engine keeps a private memory space. The memory space contains the function call instructions. When the processing engine wants to call the external function, it first issues an IRQ. The general purpose processor then jump to the ISR. Meanwhile, since the status register shows that the processing engine wants to call an external function, the ISR then creates a new task and jumps to the address of the private memory of the processing engine. After the called function returns the result, rest instructions in the private memory resumes the execution of the processing engine.

4 Prototype Implementation and Experimental Results

We implemented a prototype on the Altera SOPC (system-on-programmable-chip) development board [13] to study the performance impacts. The Altera SOPC development board adopts an FPGA chip to be its core. We programmed the Altera Nios soft IP core processor into the FPGA chip. We choose the μC/OS-II [12] to be our operating system.

To study the performance, we implemented the square root hardware function and the loop hardware function. Both the two functions are compute-intensive. The hardware engines are implemented in VHDL and are configured as memory-mapped I/O devices. Each hardware engine needs to be registered in the OS. It also needs to register its interrupt service routine (ISR). For each hardware function, the operating system provides a multitasking interface. Figure 4 shows the implementation of the hardware

```
01 if((in_use->np_piodata & 0x2)){
02
03     ptcb = OSTCBPrioTbl[OSPrioCur];
04     OSTCBCur->OSTCBStat   |= OS_STAT_HDF;
05     OSTCBCur->OSTCBPendTO  = FALSE;
06     hdsqrt_wait++;
07     OS_EventTaskWait(hdsqrt);
08     OS_Sched();
09  }
10  in_use->np_piodata |= 0x2;
11
12  hdsqrt_owner = OSPrioCur; pointer = &hdsqrt_out->np_piodata;
13  *pointer = *parameter; pointer2 = &hdsqrt_in->np_piodata;
14
15  y                      = OSTCBCur->OSTCBY;
16  OSRdyTbl[y]           &= ~OSTCBCur->OSTCBBitX;
17  if (OSRdyTbl[y] == 0) {
18      OSRdyGrp &= ~OSTCBCur->OSTCBBitY;
19  }
20
21  launch->np_piodata |= 0x2;
22
23  OS_Sched();
```

Fig. 4. Implementation of the hardware function scheduler

function scheduler. Taking the square root hardware function as an example, line 1–9 presents the case when the hardware function is busy. We did not really implement a new queue data structure in our prototype, on the contrasty, we adopted the ready table structure in the μC/OS-II kernel. We used the table structure to record applications which is requesting the hardware function. Line 1 checks if the hardware function is busy. Line 3–6 increases the length of waiting queue of the square root hardware function. Line 7–8 sets the application into waiting state and forces the OS execute next application. Line 10–23 presents the parameter passing procedure if the square root hardware engine is available. Line 10 sets the status register of the hardware function. Line 12–13 passes the parameters. Line 15–19 maintains the application state. Line 21 activates the hardware function. Finally, line 23 forces the OS execute next application.

To evaluate our prototype, we run two experiments. In the first experiment, we developed three tasks to search prime numbers. Each task uses the square root hardware function to confirm the prime number. Hence, the square root hardware function is shared by three tasks at the same time. Figure 5 shows the result. The x-axis presents the amount of searched integers. The y-axis presents the processing time. The search workload are shared by three tasks fairly. For example, if we want to find out all prime numbers between 1 to 30000, the task 1 searches 1 to 10000, the task 2 searches 10001 to 20000, and the task 3 searches 20001 to 30000. We compared the processing time of using square root hardware function with the processing time of pure software condition. The processing time gains an obvious degradation in 15%. Accordingly, the management overhead of hardware function is little and acceptable.

In the second experiment, we developed a loop program to study the operation overhead of hardware functions. The loop program simply runs "for loop" instructions. Figure 6 shows the result. The x-axis presents the variable n. In each test case, we ran n^3 times loop instructions. The y-axis presents the processing time. The processing

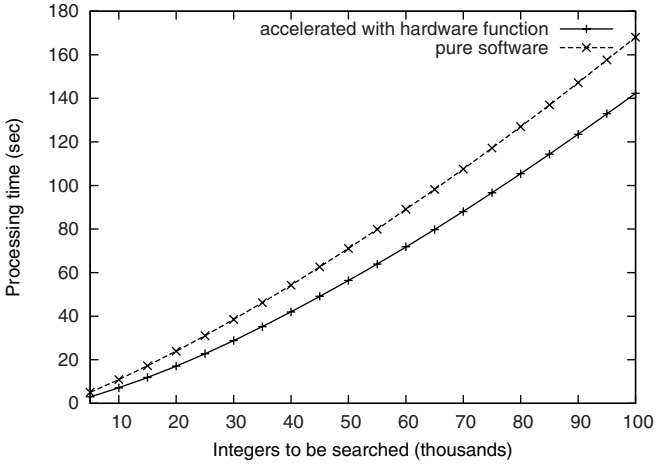


Fig. 5. Performance of the prime number search program

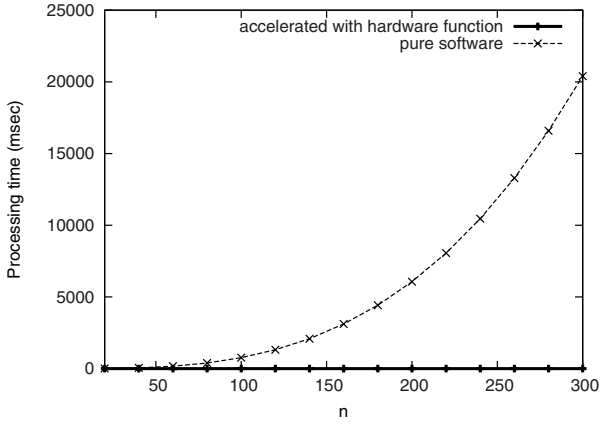


Fig. 6. Performance of the loop program

time of hardware accelerated configuration is 3–4ms. Hence, the operation overhead of hardware functions is very little and can be ignored.

5 Conclusions

Reconfigurable computing grabs the public attention recently because it provides a high performance computing paradigm. In this paper, we propose a function-level multitasking interface design in an embedded OS to exploit the high-performance benefit of reconfigurable hardware. The proposed mechanism has the following three distinguishing features: (1) multiple tasks can coherently share the reconfigurable accelerator hardware

without data inconsistency; (2) the invocation interface of the hardware units is consistent with the API of the software library; (3) OS can dynamically decide whether it resolves the invocation with the hardware unit or with the software library. A prototype implemented with an Altera SOPC development board and μC/OS-II shows its prominent performance improvement in our preliminary experiments. Although the benchmark is still primitive, the positive experimental results convince us of the feasibility in the future development.

In our future plan, the reconfigurable mechanism will be integrated in our prototype. Besides, a more comprehensive benchmark will be implemented to get complete performance characteristics. We also plan to port the proposed multitasking scheme to other famous embedded operating systems, such as uClinux. Improvements on other OS components then will be under investigation to fully take the high-performance benefit of reconfigurable hardware.

References

1. Alam, S.R., et al.: Using FPGA Devices to Accelerate Biomolecular Simulations. IEEE Computer 40(3), 66–73 (2007)
2. Andrews, D., et al.: hthreads: A Hardware/Software Co-Designed Multithreaded RTOS Kernel. In: Proc. IEEE ETFA 2005, pp. 331–338 (September 2005)
3. Buell, D., et al.: Guest Editors' Introduction: High-Performance Reconfigurable Computing. IEEE Computer 40(3), 23–27 (2007)
4. Garcia, P., et al.: An Overview of Reconfigurable Hardware in Embedded Systems. Eurasip Journal of Embedded Systems 2006, 1–19 (2006)
5. Prasanna, V.K., Morris, G.R.: Sparse Matrix Computations on Reconfigurable Hardware. IEEE Computer 40(3), 58–64 (2007)
6. Rullmann, M., Siegel, S., Merker, R.: Optimization of Reconfiguration Overhead by Algorithmic Transformations and Hardware Matching. In: Proc. IEEE IPDPS 2005 (April 2005)
7. Santos, R., Azevedo, R., Araujo, G.: Exploiting Dynamic Reconfiguration Techniques: The 2D-VLIW Approach. In: Proc. IEEE IPDPS 2006 (April 2006)
8. Shibamura, H., et al.: EXPRESS-1: A Dynamically Reconfigurable Platform using Embedded Processor FPGA. In: Proc. IEEE ICFPT 2004 (December 2004)
9. Todman, T.J., et al.: Reconfigurable Computing: Architectures and Design Methods. IEE Proceedings: Computers and Digital Techniques 152(2), 193–207 (2005)
10. Wigley, G., Kearney, D., Jasiunas, M.: ReConfigME: A Detailed Implementation of an Operating System for Reconfigurable Computing. In: Proc. IEEE IPDPS 2006 (April 2006)
11. Zhou, B., et al.: Reduce SW/HW Migration Efforts by a RTOS in Multi-FPGA Systems. In: Shen, W.-m., Chao, K.-M., Lin, Z., Barthès, J.-P.A., James, A. (eds.) CSCWD 2005. LNCS, vol. 3865, pp. 636–645. Springer, Heidelberg (2006)
12. Labrosse, J.: MicroC/OS-II, CMP Books (June 2002)
13. Altera corp.: SOPC Builder. http://www.altera.com/products/software/products/sopc/

Task Scheduling for Context Minimization in Dynamically Reconfigurable Platforms

Nei-Chiung Perng and Shih-Hao Hung

Department of Computer Science and Information Engineering
National Taiwan Univeristy
106 Taipei, Taiwan
{d90011,hungsh}@csie.ntu.edu.tw

Abstract. Dynamically reconfigurable hardware provides useful means to reduce the time-to-prototype and even the time-to-market in product designs. It also offers a good alternative in reconfiguring hardware logics to optimize the system performance. This paper targets an essential issue in reconfigurable computing, i.e., the minimization of configuration contexts. We explore different constraints on the CONTEXT MINIMIZATION problem. When the resulting subproblems are polynomial-time solvable, optimal algorithms are presented.

1 Introduction

Dynamically reconfigurable hardware (DRH) allows partial reconfigurations to provide different functionalities over a limited number of hardware logics, compared to the popular Application-Specific Integrated Circuit (ASIC) approach. It was recently raised by many researchers that the DRH technology is very suitable to deal with the dynamism of multimedia applications [1,2]. In such applications, instructions might be partitioned into coarse-grained tasks with a partial order and loaded onto dynamically reconfigurable devices for executions! Reconfigurability has recently become an important issue in the research community of embedded systems especially for FPGAs [3,4,5,6]. An FPGA configuration context, also referred to as a context, is the basic element to load a hardware description of a task. A multi-context system is a system with more than one FPGA chips or an FPGA chip with its configurable logic blocks being partitioned into several (equal-sized) areas. Although multi-context systems allow simultaneous task executions on different contexts, many implementations only allow the loading of one task at a time in reality [7].

One way to avoid the waiting time of task loadings is to pre-load proper hardware descriptions before the run time. In particular, Hauck [8] presented the concept of configuration prefetching in which the loading duration was overlapped with computations to reduce the overheads. Harkin et al. [9] evaluated nine approaches of hardware/software partitioning to provide insights in the implementation methodology for reconfiguration hardware. A genetic algorithm (GA) was also presented by the same authors for run-time reconfiguration [10]. Yuh et al. [11] developed a tree-based data structure, called T-tree, for a temporal floorplanning to schedule all the reconfigurable tasks with a simulated-annealing-based algorithm. Ghiasi et al. [12] proposed an efficient optimal algorithm to minimize the run-time reconfiguration delay in the executions of applications

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 55–63, 2007.

on a dynamically adaptable system under assumptions on several restricted implementation constraints. Noguera and Badia [2] introduced a two-version dynamic scheduling algorithm for reconfigurable architectures with or without a prefetching unit. Resano et al. [13] proposed a way to revise a given task schedule by considering reconfiguration to minimize the latency overheads.

This paper targets one essential implementation issue in the reconfigurable computing: the minimization of the required number of FPGA configuration contexts, where the deadline and precedence constraints of an application are given. We consider the optimization problem without a given schedule (that comes with fixed execution intervals). The NP-completeness of the CONTEXT MINIMIZATION problem is first proved. Several additional constraints on the loading time and the execution time of a task and the precedence constraints of tasks (and their resulting subproblems) are explored. Optimal algorithms are presented for subproblems that are polynomial-time solvable. The objective is provide insights on how and why difficult the CONTEXT MINIMIZATION problem is.

The rest of this paper is organized as follows: Section 2 defines the CONTEXT MINIMIZATION problem. Section 3 explores several subproblems under several additional constraints. Optimal algorithms for subproblems that are solvable in polynomial time are presented. Section 4 is the conclusion.

2 Problem Definition

In this paper, we are interested in the derivation of a *reconfiguration plan* $\Re$ with the objective to minimize the number of required FPGA configuration contexts in a multi-context FPGA platform. The reconfiguration plan should be derived based on a given task set T, a partial order of task precedences $\prec$, and a common deadline D. Each task τ_i in a task set T is denoted as $\tau_i = (e_i, l_i)$, where e_i is the required execution time, and l_i is the loading (configuration) duration to load task τ_i onto a context. A precedence constraint $\tau_i \prec \tau_j$ in the partial order $\prec$ requires that task τ_j can only start its execution after the completion of task τ_i. A precedence constraint might exist because the latter task needs to read from the output of the former task. We are interested in the minimization of the maximum time span of the task execution in T. The problem is modeled as a performance requirement, i.e., the common deadline D. Any solution to the targeted problem is a reconfiguration plan $\Re$, in which we have a loading time $\Re_{\mathbb{T}}(\tau_i)$, an execution starting time $\Re_{\mathbb{S}}(\tau_i)$, and a configuration context ID $\Re_{\mathbb{C}}(\tau_i)$ for each task τ_i. A solution should also satisfy the given partial order of task executions and the common deadline. The problem is formally defined as follows.

Problem 1 (CONTEXT MINIMIZATION). Given a set T of n tasks ($\tau_i = (e_i, l_i) \in T, 1 \leq i \leq n$) with a partial order $\prec$ and a common deadline D, the problem is to find a reconfiguration plan $\Re$ with the minimum number of required FPGA configuration contexts without violating the partial order of task executions and the common deadline.

A reconfiguration plan is *feasible* if and only if the following three conditions are satisfied: The first condition requires each FPGA context being loaded in time. The second condition requires that any two tasks should not use the same context in any overlapped

time interval. The third condition requires the loading of contexts should be done one by one. The three conditions are defined formally as follows:

Condition 1 (In-Time Loading). $\forall \tau_i$, $\Re_{\mathbb{T}}(\tau_i) + l_i \leq \Re_{\mathbb{S}}(\tau_i)$ *and* $\Re_{\mathbb{S}}(\tau_i) + e_i \leq D$.

Condition 2 (Non-Overlapping Configuration Contexts). $\forall\ (\tau_i, \tau_j)$ *pair,* $\Re_{\mathbb{C}}(\tau_i) \neq \Re_{\mathbb{C}}(\tau_j)$ *if any two time intervals* $(\Re_{\mathbb{T}}(\tau_i), \Re_{\mathbb{S}}(\tau_i) + e_i]$ *and* $(\Re_{\mathbb{T}}(\tau_j), \Re_{\mathbb{S}}(\tau_j) + e_j]$ *have a non-null intersection.*

Condition 3 (Mutual Exclusion on Loading). $\forall\ (\tau_i, \tau_j)$ *pair,* $\Re_{\mathbb{T}}(\tau_i) + l_i \leq \Re_{\mathbb{T}}(\tau_j)$ *or* $\Re_{\mathbb{T}}(\tau_j) + l_j \leq \Re_{\mathbb{T}}(\tau_i)$.

A reconfiguration plan is *optimal* if it is feasible, and the number of its required contexts, i.e., the largest ID of the assigned contexts, is equal to the minimum number of required contexts of all feasible reconfiguration plans. We shall show later that this optimization problem is $\mathcal{NP}$-complete.

3 Problem Properties

In this section, we prove the $\mathcal{NP}$-completeness of the CONTEXT MINIMIZATION problem and later explore subproblems in which polynomial-time solutions exist. For the sake of clarity, we transform this optimization problem into an equivalent decision problem by providing a bound on the number of FPGA configuration contexts. The decision version of the CONTEXT MINIMIZATION problem is to find a solution with the number of required contexts no larger than a given number M.

3.1 $\mathcal{NP}$-Complete Subproblems

We shall show the $\mathcal{NP}$-completeness of two subprograms of the CONTEXT MINIMIZATION problem under two constraints: (1) The execution time of each task is identical, i.e., $\forall i$, $e_i = E$. (2) The loading duration of each task is negligible, i.e., $\forall i$, $l_i = 0$. Before we show the $\mathcal{NP}$-completeness of the two subproblems, we shall first define the PRECEDENCE CONSTRAINED SCHEDULING (SS9 in [14]), that is $\mathcal{NP}$-complete:

Given a set T of tasks, the execution time e_i of every task $\tau_i \in T$ is 1, a given number $M \in \mathcal{Z}^+$ of processors, a partial order $\prec$ of tasks $\in T$, and a deadline $D \in \mathcal{Z}^+$, the problem is to derive a schedule σ over the M processors so that the deadline and the partial order of task executions are satisfied. In other words, $\forall \tau_i, \tau_j \in T$, $\tau_i \prec \tau_j$ implies $\sigma(\tau_j) \geq \sigma(\tau_i) + e_i$, where $\sigma(\tau_i)$ denotes the starting time of task τ_i.

Theorem 1. *The CONTEXT MINIMIZATION problem under the constraint* $\forall i$, $e_i = E$ *and* $l_i = 0$ *is* $\mathcal{NP}$*-complete, where* e_i *and* l_i *denote the execution time and the loading duration of task* τ_i*, respectively.*

Proof. This subproblem is indeed the PRECEDENCE CONSTRAINED SCHEDULING problem. □

Theorem 2. *The CONTEXT MINIMIZATION problem under the constraint* $\forall i,\ l_i = 0$ *is* $\mathcal{NP}$*-complete, where* l_i *denotes the loading duration of task* τ_i*.*

Proof. The correctness of this theorem follows directly from the fact that the problem stated in Theorem 1 is a special case of this problem. □

Theorem 3. *The CONTEXT MINIMIZATION problem is* $\mathcal{NP}$*-complete.*

Proof. The correctness of this theorem follows directly from the fact that the problem stated in Theorem 2 is a special case of this problem. □

3.2 Subproblems in $\mathcal{P}$

This section is meant to explore three subproblems of the CONTEXT MINIMIZATION problem that have polynomial-time solutions. It is to gain the insight on why and how difficult the problem is.

A Task Set of Independent Tasks. Although the CONTEXT MINIMIZATION problem is $\mathcal{NP}$-complete, the problem would become tractable under certain constraints. Suppose that all of the tasks are independent, i.e., $\prec = \emptyset$, and $\forall i,\ e_i = E$ and $l_i = L$ (i.e., the same execution time and loading duration for every task). The problem has optimal algorithms with a polynomial time complexity.

Algorithm 1

Input: A task set T ($\forall \tau_i \in T, \tau_i = (E, L)$), $\prec = \emptyset$, a deadline D, and M contexts
Output: A feasible reconfiguration plan $\Re$

1: $\Im = 0$.
2: **while** T is not empty **do**
3: Remove an arbitrary task τ_i from T.
4: Locate context M_j with the earliest idle time I_j.
5: $\Re_{\mathbb{T}}(\tau_i) = \max\{I_j, \Im\}$.
6: $\Re_{\mathbb{S}}(\tau_i) = \Re_{\mathbb{T}}(\tau_i) + L$.
7: $\Re_{\mathbb{C}}(\tau_i) = j$.
8: $\Im = \Re_{\mathbb{T}}(\tau_i) + l_i$.
9: **if** $\Re_{\mathbb{S}}(\tau_i) + E > D$ **then**
10: Return failure.
11: **end if**
12: **end while**

The subproblem is shown being $\mathcal{P}$-solvable by presenting a polynomial-time algorithm Algorithm 1. The input of the algorithm includes a task set, a common deadline, and an integer constant M that denotes the number of FPGA configuration contexts. Because of the mutual exclusion requirements on context loading (Condition 3), the algorithm maintains the earliest possible time for the next context loading, i.e., $\Im$. $\Im$ is initialized as 0 initially (Step 1). Each iteration of the loop between Step 2 and Step 12 is to schedule the loading of a task onto a context. The algorithm picks up a ready

task arbitrarily (Step 3) and load the task onto the context with the earliest available time (Step 4). The loading time, starting time, and context ID of the task are updated accordingly (Steps 5-7). The earliest possible time for the next context loading is then updated (Please see Step 8 and Condition 3). The algorithm reports a failure if any task misses the deadline (Steps 9-11). The time complexity is $O(n \times \log M)$.

Theorem 4. *Algorithm 1 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The correctness of this theorem follows directly from the fact that all tasks are of the same execution time and loading duration and share the common deadline. □

Figure 3.2 shows four optimal reconfiguration plans of four tasks, where the number M of FPGA configuration contexts ranges from 1 to 4. An interesting packing of tasks is shown in the figures, and the impacts of the mutual exclusion constraint, i.e., Condition 3, are clearly illustrated. Note that the shaded rectangles denote the loadings of tasks onto contexts, and white rectangles denote task executions.

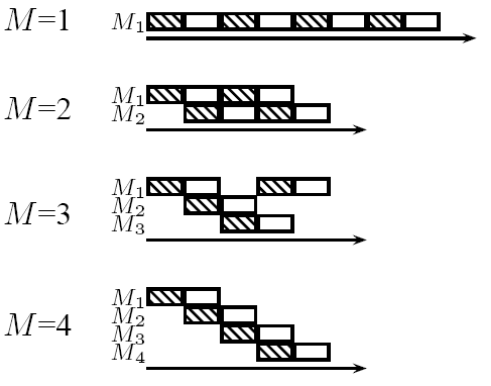


Fig. 1. Four reconfiguration plans over four different numbers of contexts

Lemma 1. *Algorithm 1 needs no more than* $M_B = \max\{n, \lceil \frac{E}{L} \rceil + 1\}$ *contexts to derive a feasible reconfiguration plan, where* n *is the number of tasks.*

Proof. The correctness of this lemma follows directly from the facts that loading durations can not be overlapped with each another, and a loading duration can be overlapped with any execution time as long as the three feasibility conditions are satisfied. □

Lemma 1 provides an upper bound on the maximum number of contexts over that Algorithm 1 could derive a feasible reconfiguration plan. Figure 2 shows reconfiguration plans for two task sets, i.e., one with $E \leq L$ and the other with $E > L$, where different numbers of FPGA configuration contexts are tried. Note that when $n \times (L + E) \leq D$, only one context is needed to derive a feasible reconfiguration plan.

Although we show that the CONTEXT MINIMIZATION problem becomes tractable when $\prec = \emptyset$, and $\forall i$, $e_i = E$ and $l_i = L$, one question remains. That is how difficult

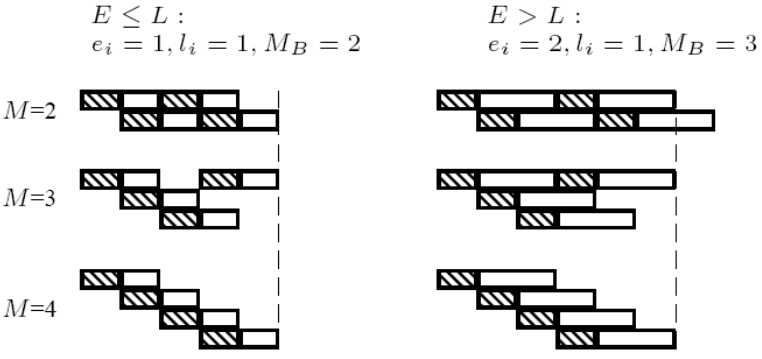


Fig. 2. Reconfiguration plans of two task sets over different numbers of contexts

the problem is if we relax some of the above constraints! Suppose that we still have tasks being independent, i.e., $\prec = \emptyset$, but every task might have a different execution time, i.e., $e_i \neq e_j$ for some $\tau_i, \tau_j \in T$. Even if we let $\forall i$, $l_i = 0$, the CONTEXT MINIMIZATION problem is intractable because the problem is indeed the MINIMUM MULTIPROCESSOR SCHEDULING problem, which is a well-known $\mathcal{NP}$-complete problem [14].

A Task Set with a Tree Partial Order. As shown in Theorem 1, the CONTEXT MINIMIZATION problem is $\mathcal{NP}$-complete when $\forall i$, $e_i = E$ and $l_i = 0$. However, the CONTEXT MINIMIZATION problem becomes tractable when all tasks are independent, i.e., $\prec = \emptyset$. In this section, we shall show that the CONTEXT MINIMIZATION problem remains tractable when $\prec$ is of a tree, and $\forall i$, $e_i = E$ and $l_i = 0$, regardless of whether it is an intree or an outtree (Figure 3).

We shall present an optimal algorithm based on the Critical Path (CP) rule [15]. A *critical path* of a partial order is defined as a path with the maximum number of tasks in a chain that follow the precedence constraints of the order. Note that when the completion time of a task on a critical path is delayed, the latest completion time of the tasks in the task set is delayed. Figure 3 shows an intree and an outtree. The path from the top-left node to the sink node of the intree in Figure 3 is an example critical path,

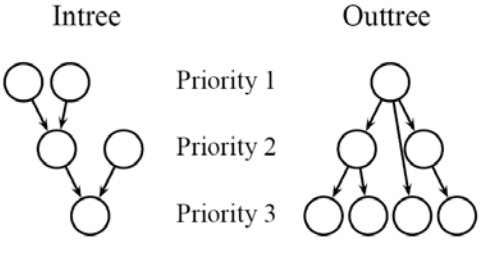


Fig. 3. An intree/outtree for some precedence constraints

and there are two critical paths in the intree. The CP rule provides a way to assign tasks priorities, where a task in the front of a critical path is assigned a higher priority, as shown in Figure 3. In fact, the priority assignment of tasks follows a topological order. Tie-breaking is done in an arbitrary way.

Algorithm 2

Input: A task set T ($\forall \tau_i \in T, \tau_i = (E, 0)$), $\prec$ as a tree, a deadline D, and M contexts
Output: A feasible reconfiguration plan $\Re$

1: Assign priorities to tasks according to the CP rule.
2: $S = \{ \tau_i : \tau_i \in T$ does not have any predecessor $\}$.
3: **while** S is not empty **do**
4: Remove the task τ_i with the highest priority from S.
5: Locate context M_j with the earliest idle time I_j.
6: $\Re_{\mathrm{T}}(\tau_i) = \Re_{\mathrm{S}}(\tau_i) = I_j$.
7: $\Re_{\mathrm{C}}(\tau_i) = j$.
8: $S = S \cup \{\tau_j\}$, where $\tau_i \prec \tau_j$ if all of the predecessors of τ_j have been scheduled.
9: **if** $\Re_{\mathrm{S}}(\tau_i) + E > D$ **then**
10: Return failure.
11: **end if**
12: **end while**

Algorithm 2 derives a feasible reconfiguration plan whenever possible: Tasks are first assigned priorities according to the CP rule (Step 1). Initially, S is set as the set of ready tasks in T (Step 2), where a ready task is a task with all of its preceding tasks in the partial order complete. Each iteration of the loop between Step 3 and Step 12 is to load a task onto a proper context. The ready task with the highest priority is loaded onto the context with the earliest possible idle time. The loading time and the starting time of the task is set as the earliest possible idle time of the context (Step 6), where $\forall i$, $l_i = 0$. The context ID of the task is then set (Step 7). After the task is scheduled, any ready task resulted from the scheduling join the ready task pool (Step 8). If the deadline is violated, then the algorithm reports a failure (Steps 9 and 10). The time complexity of Algorithm 2 is $O(n^2)$.

Theorem 5. *Algorithm 2 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The optimality of the algorithm is based on the proof of the CP rule in [15]. □

A Task Set with $E \leq L$. As shown in the previous section, the CONTEXT MINIMIZATION problem becomes more tractable when a partial order is restricted in a tree fashion, compared to that shown in Theorem 1. Another question is whether we could trade the partial-order constraint with any other constraint to keep the CONTEXT MINIMIZATION problem being tractable. In this section, we shall show that the CONTEXT MINIMIZATION problem remains tractable by the constraint $\forall i$, $e_i = E$, $l_i = L$, and $E \leq L$. In such a case, the partial order among tasks could be arbitrary.

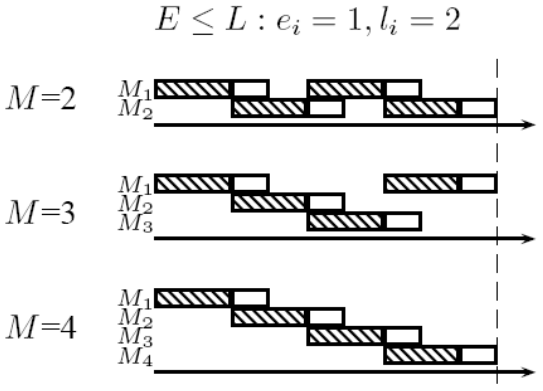


Fig. 4. Reconfiguration plans of a task set with $E \leq L$, where different numbers of contexts are used

An optimal algorithm for this problem is as the same as Algorithm 1 except two minor modifications: (1) Step 3: Remove any arbitrary ready task $\tau_i \in T$, and (2) Step 4: Use the context M_j which is idle in the last iteration. The algorithm is referred to as Algorithm 3. The time complexity of Algorithm 3 is $O(n)$.

Theorem 6. *Algorithm 3 is optimal in the sense that it always derives a solution if any feasible solution exists.*

Proof. The optimality of the algorithm is based on the fact that the minimum number of the required contexts must be 1 or 2, unless there is no feasible solution. If the summation of loading durations and execution times of all tasks is less than the common deadline, i.e., $n \times (L + E) \leq D$, the answer is 1; Otherwise, the answer is 2. □

Figure 4 illustrates the idea of the proposed algorithm and provides further explanation of the above theorem. As shown in the figure, there are three reconfiguration plans of a task set derived by Algorithm 3, where the number of contexts ranges from 2 to 4. Because of the mutual exclusion requirements on context loadings (i.e., Condition 3), only two contexts are needed to load tasks in turn. The execution time of each task on one context is contained in the loading duration of another task on the other context, where these two tasks are scheduled one after another. We shall point out that the CONTEXT MINIMIZATION problem would become much more intractable when constraints of Section 3 are relaxed.

4 Conclusion

In this paper, we explore different constraints on the CONTEXT MINIMIZATION problem, and provide the complexity proofs of $\mathcal{NP}$-complete subproblems and the optimal algorithms of subproblems in $\mathcal{P}$. We are currently investigating heuristic-based greedy algorithms for the CONTEXT MINIMIZATION problem. We will further explore the problem on the reconfigurable platforms allowing one-dimension allocation.

References

[1] Kneip, J., Schmale, B., Moller, H.: Applying and implementing the mpeg-4 multimedia standard. IEEE Micro 19(6), 64–74 (1999)
[2] Noguera, J., Badia, R.M.: Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling. ACM Transactions on Embedded Computing Systems 3(2), 385–406 (2004)
[3] De Micheli, G., Gupta, R.K.: Hardware/software co-design. Proceedings of the IEEE 85(3), 349–365 (1997)
[4] De Hon, A., Wawrzynek, J.: Reconfigurable computing: What, why, and implicatios for design automation. In: Proceedings of the 36th ACM/IEEE Conference on Design Automation, pp. 610–615 (1999)
[5] Hauck, S.: The roles of FPGA's in reprogrammable systems. Proceedings of IEEE 86(4) (1998)
[6] Wolf, W.: FPGA-Based System Design. Prentice-Hall, Englewood Cliffs (2004)
[7] Xilinx Inc.: XAPP151 Virtex Series Configuration Architecture User Guide (v1.7) edn. (2004)
[8] Hauck, S.: Configuration prefetch for single context reconfigurable coprocessors. In: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (1998)
[9] Harkin, J., McGinnity, T.M., Maguire, L.P.: Partitioning methodology for dynamically reconfigurable embedded systems. IEE Proceedings on Computers and Digital Techniques 147(6), 391–396 (2000)
[10] Harkin, J., McGinnity, T.M., Maguire, L.P.: Modeling and optimizing run-time reconfiguration using evolutionary computation. ACM Transactions on Embedded Computing Systems 3(4), 661–685 (2004)
[11] Yuh, P.H., Yang, C.L., Chang, Y.W.: Temporal floorplanning using the T-tree formulation. In: Proceedings of ACM/IEEE International Conference on Computer-Aided Design (2004)
[12] Ghiasi, S., Nahapetian, A., Sarrafzadeh, M.: An optimal algorithm for minimizing run-time reconfiguration delay. ACM Transactions on Embedded Computing Systems 3(2), 237–256 (2004)
[13] Resano, J., Mozos, D., Catthoor, F.: A reconfigurable manager for dynamically reconfigurable hardware. IEEE Design and Test of Computers 22(5) (2005)
[14] Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company, New York (1979)
[15] Pinedo, M.: Scheduling Theory, Algorithms, and Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (2002)

Compiler Support for Dynamic Pipeline Scaling

Kuan-Wei Cheng, Tzong-Yen Lin, and Rong-Guey Chang

Department of Computer Science
National Chung Cheng University
Chia-Yi, Taiwan
{why93,lty93,rgchang}@cs.ccu.edu.tw

Abstract. Low power has played an increasingly important role for embedded systems. To save power, lowering voltage and frequency is very straightforward and effective; therefore dynamic voltage scaling (DVS) has become a prevalent low-power technique. However, DVS makes no effect on power saving when the voltage reaches a lower bound. Fortunately, a technique called dynamic pipeline scaling (DPS) can overcome this limitation by switching pipeline modes at low-voltage level. Approaches proposed in previous work on DPS were based on hardware support. From viewpoint of compiler, little has been addressed on this issue. This paper presents a DPS optimization technique at compiler time to reduce power dissipation. The useful information of an application is exploited to devise an analytical model to assess the cost of enabling DPS mechanism. As a consequence we can determine the switching timing between pipeline modes at compiler time without causing significant run-time overhead. The experimental result shows that our approach is effective in reducing energy consumption.

1 Introduction

Since most embedded systems are portable, reducing energy consumption to extend the lifetime of batteries has become a crucial issue. In recent years, many techniques have been proposed to address this issue. DVS is the famous one, which has been demonstrated by much work to be very effective [8, 2, 12]. It adjusts dynamically voltage and frequency to save power, as indicated in Equation 1.

$$E \propto f \times C \times V^2, \tag{1}$$

where A $\propto$ B means A is in direct ratio to B. However, DVS has no effect on energy saving when the voltage reaches its low bound because it becomes a constant [10]. Fortunately, with reference to Equation 2, energy is in direct ratio not only to the clock frequency and the square of the voltage, but also to instruction-per-cycle (IPC).

$$E \propto f \times V^2 \times t \propto f \times V^2 \times \frac{I_t}{f \times IPC} \propto \frac{V^2}{IPC} \tag{2}$$

Thus, we can reduce energy dissipation at low-voltage level based on IPC. Equation 1 and Equation 2 reveal that IPC is the key to power dissipation at low-voltage level. This fact shows that power will increase in the opposite direction of IPC and motivates our low-power idea to devise a DPS technique to evaluate the IPC and determine

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 64–74, 2007.

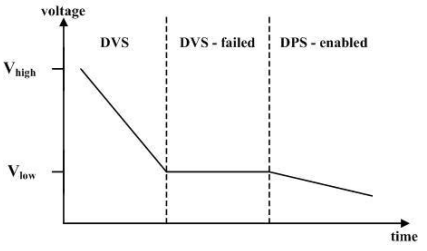


Fig. 1. Voltage characteristic

the switching timing between pipeline modes at compiler time. In DPS, the pipeline consists of deep mode and shallow mode. The deep mode is the default pipeline mode and the shallow mode is designed by dynamically merging adjacent pipeline stages of deep mode, where the latches between pipeline stages are made transparent and the corresponding feedback paths are disabled. In theory IPC is in inverse ratio to the pipeline depth, the IPC of deep mode may be smaller than that of shallow mode [6]. Therefore, executing applications in shallow mode will lead to the reduction of power dissipation. But this statement is not always true. In reality, many factors in deep pipeline mode will influence IPC [15,13].

Hence, if we want to apply DPS to save power at low-voltage level, we must decide when the pipeline enter deep mode or shallow mode depending upon the IPC. Consider the voltage characteristic shown in Figure 1. In the first stage, DVS is applied to save power at high-voltage level. Although reducing voltage is very effective to low power, DVS fails in the second stage when the voltage reaches its lower bound. At the final stage, we can enable DPS to switch the pipeline modes based on IPC. Since IPC is affected by some factors, we can consider their impact on IPC to determine the switching timing between pipeline modes to save energy.

Previous work on DPS was proposed by architects with architectural support [10, 7, 4, 14, 3]. However, the research about how to solve this issue with compilation techniques remains open. In this paper, we present an optimization technique to enable DPS with respect to IPC at compile time. We first partition an application into many regions and then calculate the IPC of each region to determine the switching timing between pipeline modes based on our evaluating model. Since our work is performed at compiler time, the run-time overhead will be small and the hardware cost and complexity will be as minimal as possible. The experimental results prove that the energy reduction really benefit from our work.

The rest of this paper is organized as follows. Section 2 we study the previous work on DPS. Section 3 gives the overview of our work and then presents our approach in detail. The experimental results are shown in Section 4. Finally, we conclude our paper in Section 5.

2 Related Work

Although many researchers still focus on devising new DVS techniques, other low-power techniques that are irrelevant to DVS are valuable to be exploited. DPS is a

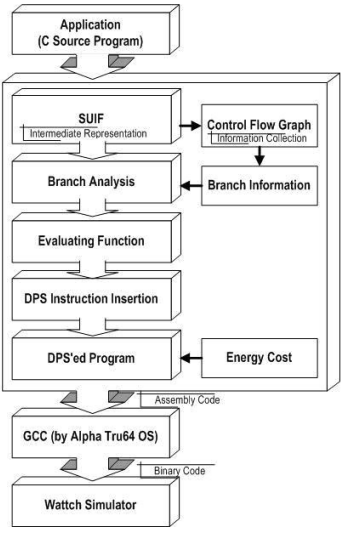


Fig. 2. The proposed DPS compilation system

typical example. Previous work on DPS has concentrated on hardware techniques, but how to solve this issue using software techniques remains open. For example, Koppanalil et al. devised a DPS technique by switching the pipeline between deep mode and shallow mode at run time with respect to high and low frequencies respectively [10]. Hiraki et al. proposed a method that skipped several pipeline stages and then used a decoded buffer to replace the functionality of the original pipelines for low-power support [7]. Ernst et al. speculated on the timing information by monitoring the error rate of pipeline processing to switch pipeline modes at circuit level [4]. Manne et al. applied pipeline gating to reduce energy consumption by stopping wrong-path instructions from entering the pipeline when a branch misses its prediction result [14]. Efthymiou et al. [3] applied a hardware-controlling method to reduce energy dissipation by adapting the depth of pipeline stages.

3 The Proposed Approach

In this section, we focus on how our DPS approach is applied to applications to save energy at compiler time. We first introduce our basic idea in Section 3.1. In section 3.2, we depict the method to partition a code into regions and then present the evaluating function to decide the switching between pipeline modes. The mechanism to enable DPS is given in Section 3.3.

3.1 Basic Idea

Figure 2 shows our compilation system composed of SUIF framework [16], our proposed engine, and Wattch simulator. First, an application is compiled by SUIF as a

control flow graph (CFG) and a data flow graph (DFG). Then the CFG and DFG are analyzed to identify the loop regions and collect information for our evaluating model and identify loop regions. In our work, a code will have another type of regions, non-loop regions, except loop regions. Indeed, a region is an union of basic blocks as a unit that our DPS can manipulate. The evaluation model has two goals: one is to partition the remaining part of the loop regions into non-loop regions and the other is enable each region to enter a suitable pipeline mode. The details of partitioning scheme is described in Section 3.2. To activate DPS, for each region we will insert the DPS-enable function in its entrance at compile time so that it can be executed in proper pipeline mode to save energy based on its IPC. Since the switching between pipeline modes best is very hard to decide, we propose an evaluation model to decide the timing during execution. The evaluation model is presented in detail in Section 3.3. In this way, the code will be switched between different pipeline modes at run time. The experiment is performed on the Wattch simulator [1] with DSPstone and Mediabench benchmark suites.

3.2 Evaluation Model for Switching Pipeline Modes

As mentioned in Section 1, the IPC of deep pipeline mode is not always larger than that of shallow mode. As a consequence, the shallow mode may have better power saving than the deep mode according to Equation 2. Thus to reduce power reduction, each region can enter deep mode or shallow mode based on the IPC during execution. To achieve this objective, we conduct an evaluation model to decide the switching timing between deep mode and shallow mode at compiler time. Since the calculation of IPC closely relates to the size of a region, how to partition a code into regions becomes very important to our work. On one hand, if the region size is too large, we may lose the chances to take advantage of switching pipeline modes to save energy. On the other hand, although the small region size can allow us to apply the DPS optimization to a code, it possibly generates severe switching overheads. However what the size of a region is the optimal solution for our approach is very hard to decide, thus we attempt to seek for a principle to guide our selection in this section. With our observations, since the loops usually dominate execution time and power consumption of a code, they are the key to our decision. This result motivates us to use major loop as a guideline to classify a code into regions to perform our code partitioning.

To use the loops to partition a code, they must be identified first and then be referred to divided the remaining part into non-loop regions. Below we present our partitioning approach and evaluation model. Given a code $G = (V, E)$, it is divided into two types of regions, Γ_1 and Γ_2, where $\Gamma_1, \Gamma_2 \subseteq V \times E, G = \Gamma_1 \bigcup \Gamma_2$, and $\Gamma_1 \bigcap \Gamma_2 = \emptyset$. Note $\emptyset$ represents the empty set; that is Γ_1 and Γ_2 are disjoint. We first define Γ_1 in case A and then use Γ_1 to define Γ_2 in case B.

Case A: Γ_1 is the set of regions, which are composed of loops. In other words, each region in Γ_1 only contains a loop.

After defining the loop regions, to classify the non-loop regions, we must present our evaluation model first. Then the evaluation model is applied to loop regions for categorizing the non-loop regions. Below we first give some assumptions and then present

how to use them to divided the non-loop part of a code into non-loop regions and finally formalize our evaluation model.

For a code $G = (V, E)$, where $V = \{R_1, R_2, ..., R_n\}$ is the set of regions in G and $E = \{(u, v) \mid u, v \in V \text{and } u \neq v\}$. That is, E is the set of edges between regions. For each region R_i, we assume:

- N_{R_i}: the number of instructions in region R_i, for $i = 1, 2, 3, \cdots$
- N_{b_i}: the number of branches in R_i
- B_{ij}: the jth branch in R_i
- $P_{B_{ij}}$: the probability that B_{ij} is taken
- C_i: the number of clock cycles that does not result from any branch in R_i
- CB_{ij}: the number of clock cycles that results from that B_{ij} is taken
- $\overline{CB}_{ij}$: the number of clock cycles that results from that B_{ij} is untaken

According to Equation 1 and Equation 2, IPC predominate the determination of switching pipeline modes during execution. With the information collected previously and the above terminologies, we can present our evaluating model as follows.

$$\Omega_{R_i} = \sum_{j=1}^{N_{b_i}} [P_{B_{ij}} \times CB_{ij} + (1 - P_{B_{ij}}) \times \overline{CB}_{ij}] + C_i \tag{3}$$

$$\Theta_{R_i} = N_{R_i} / \Omega_{R_i} \tag{4}$$

Equation 3 estimates the clock cycles required for each region of the target code, which is also applied to classify the non-loop regions. Equation 4 calculates the IPC for each region and is the guideline to enable DPS. Since the loop regions very likely dominates power dissipation of a code, we use the following parameter Λ with the aid of the evaluation function of regions in Γ_1 to partition the non-loop part of a code. Λ is defined as the maximum of all Ω_{R_i} in Γ_1. Formally, it can be described as follows.

$$\Lambda = \{\Omega_R \mid \exists R \in \Gamma_1 \text{ and } \Omega_R \geq \Omega_{R_i}, \text{for } i = 1, \cdots, n\} \tag{5}$$

Although the loops usually consume the majority of power dissipation for an application, using λ to partition the non-loop part can be furthermore improved. Instead, we adapt Λ as the new parameter by timing a α to it, where α is a real number. Thus Γ_2 can be defined on the basis of $\alpha\Lambda$ in the following case B.

Case B: For a code we first identify the loop regions in Case A and the remaining part is classified into non-loop regions. This part consists of two types of regions. The first is a set of code segments existing between loops and the second type has two special parts. One is from the beginning of a code to the beginning of the first loop region and the other is from the end of the last loop region to the end of a code. For each non-loop part, on one hand if its evaluation value (Ω value) is smaller than $\alpha\Lambda$, then it is identified as a non-loop region. On the other hand, if Ω value is larger than $\alpha\Lambda$, then it will be categorized into many non-loop regions so that their Ω values are not larger than $\alpha\Lambda$.

To make our partitioning mechanism clear, the above steps are summarized in Figure 3.

For a given code $G = (V, E)$, divide G into two types of pipeline regions, Γ_1 and Γ_2, where $\Gamma_1, \Gamma_2 \subseteq V$, $V = \Gamma_1 \cup \Gamma_2$, and $\Gamma_1 \cap \Gamma_2 = \emptyset$;

1. Identify the loops of G as the first type of regions, Γ_1. Assume $\Gamma_1 = \{R_{a_1}, R_{a_2}, \cdots, R_{a_n}\}$, where $R_{a_i} \cap R_{a_j} = \emptyset$ for $i \neq j$.
2. Define $\Lambda = \{\Omega_{R_{a_i}} \mid \exists\, R_{a_i} \in \Gamma_1$ and $\Omega_{R_{a_i}} \geq \Omega_{R_{a_j}}$, for $j = 1, ..., n\}$
3. For the following non-loop parts:
   (a) The code segment from the beginning of G to the beginning of the first loop region.
   (b) The code segment from the end of the last loop region to the end of G.
   (c) The code segments between loops.

   Partition them into a set of regions $\Gamma_2 = \{R_{b_1}, R_{b_2}, ..., R_{b_m}\}$, where $R_{b_i} \cap R_{b_j} = \emptyset$ for $i \neq j$ and $\Omega_{R_{b_i}} \leq \alpha\Lambda$ for $i = 1, 2, \cdots, m$.

Fig. 3. Classification of regions

3.3 DPS Enabling

After the code partitioning has been done, to enable DPS, we insert a function DPS_enable () into its head of each region to make it executed in deep mode or shallow mode. The DPS_enable () is implemented as follows.

$$DPS\_enable() \begin{cases} Lda \quad \#\text{SYSCALL\_DEEP2SHAW} \\ Call\_Pal \quad \#131 \\ Lda \quad \#\text{SYSCALL\_SHAW2DEEP} \\ Call\_Pal \quad \#131 \end{cases}$$

DPS_enable() provides two functionalities to switch between pipeline modes with the system call of Alpha 21264 Call_Pal #131. #SYSCALL_DEEP2SHAW switches the pipeline from deep mode to shallow mode and #SYSCALL_SHAW2DEEP switches the pipeline from shallow mode to deep mode. In this way, we are able to determine the timing to switch pipeline modes. The Ω value of each region calculated by Equation 4 is used for DPS_enable() when the code is compiled by our system. Thus the code will dynamically enter the deep mode or shallow mode during execution after the DPS_enable() is inserted into it. Finally the optimized DPSed program is performed on the modified Wattch simulator.

4 Experimental Results

In Section 4.1, we introduce the system configuration of our work and present the experimental results in Section 4.2.

Table 1. Hardware configuration

Processor Core	
Pipeline length	4 cycles (shallow mode) 8 cycles (deep mode)
Fetch buffer	8 entries
Functional units	4 Int ALU, 2 FP ALU, 1 Int mult/div, 1 FP mult/div, 2 mem ports
Instruction window	RUU=80, LSQ=40
Issue width	6 instructions per cycle: 4 Int, 2 FP
Memory Hierarchy	
L1 D-cache size	64KB, 2-way, 32B blocks,
L1 I-cache size	64KB, 2-way, 32B blocks,
L1 latency	1 cycle
L2	Unified, 2MB, 4-way LRU 32B blocks, 11-cycle latency
Memory latency	100 cycles
TLB size	128-entry, fully-associative, 30-cycle miss

4.1 System Configuration

The underlying hardware is the Alpha 21264 processor, which contains one fetch buffer, four integer ALUs, two floating-point ALUs, one integer multiplier/divider, and one floating-point multiplier/divider, etc. In instruction window, RUU indicates register update unit and LSQ comprises load queue (LQ) and store queue (SQ). Its main features are summarized in Table 1. To perform our proposed approach, we extend the pipeline mode from one mode to two modes. We assume that the original pipelining mode is shallow mode and the new mode is deep mode by constructed by adding extra four stages to shallow pipeline. It is designed to dynamically disable one of each pair of stages by making the latches between pipeline stages transparent so that the processor can switch between these two pipeline modes. The software configuration is shown in Table 2. The SUIF compiler infrastructure is the front end of our system and generate CFG and branch information. The operating system is Tru64 UNIX for 64-bit instruction set architecture. The Wattch simulator is an architectural simulator that provides cycle-by-cycle simulation and detailed out-of-order issue with multi-level memory system [9]. For keeping consistence with our DPS approach, it has been modified to support shallow pipeline mode and deep pipeline mode.

4.2 Experimental Results

In our experiment, the deep mode is the default pipeline mode and the shallow mode is chosen during execution if necessary. The energy reduction benefits by the switching between deep mode and shallow mode depending on the IPC of a region, which is calculated by equation 4. The experiment is performed on the Wattch simulator with

Table 2. Software Configuration

OS and Software Configuration	
Profiler	SUIF
Compiler	MachSUIF
OS	Tru64 UNIX
Simulator	Wattch v.1.02 with DPS

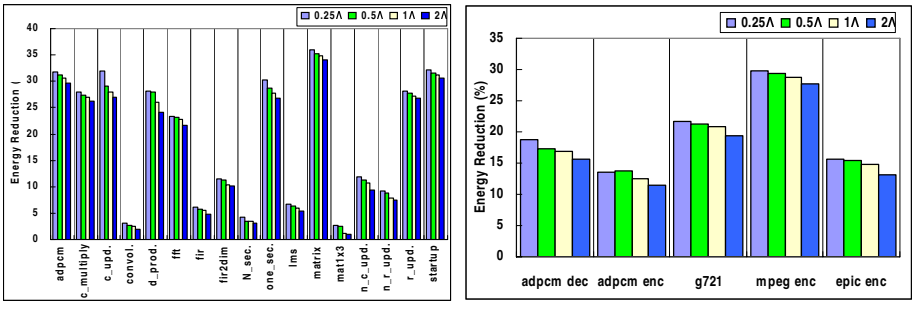


Fig. 4. Energy reduction of various λ of profile-based DPS for DSPstone and Mediabench

DSPstone and Mediabench. For each program, the baseline is its original energy dissipation and the optimized energy is measured by performing our DPS approach. This benchmark is compiled by the Alpha compiler with default settings and linked with the intrinsic library on Tru64 UNIX operating system.

Figure 4a shows the energy reduction by comparing the baseline energy and the optimized energy for DSPstone. In this experiment, we let $\alpha = 0.25, 0.5, 1.0, \text{and}, 2.0$ to measure the effects of various partitioning sizes of non-loop regions. The energy saving ranges from 2% to 35%, with a mean of reduction 17.8%. As the partitioning size of non-loop region λ becomes larger, the energy saving decreases slowly. With our observation, the large-size region eliminates some chances to switch the pipeline modes based on IPC and thus slightly increases energy consumption. Nine of these programs including adpcm, complex_multiply, complex_update, dot_product, fft, iir_biquad_one_sections, matrix, real_update, and startup, have better energy saving about from 22% to 35%. The reason is that there are many branches in them and thus our DPS approach can take advantage of them to save energy depending upon the contribution of branch penalty to IPC. With our experiences, our approach works better for the codes with many loops and larger loops. Note that many programs have large outer loops such as event-driven programs or programs with GUI, which may include almost the entire programs. In this experiment, the typical example for above discussion is matrix testbench, and it has the best energy saving about 35%.

Figure 4b shows energy reduction by performing our profile-based DPS with Mediabench benchmarks. Since these programs are loop-intensive, they at least contain a nested loop. For each benchmark, the Ω value of its loop region is large and thus the

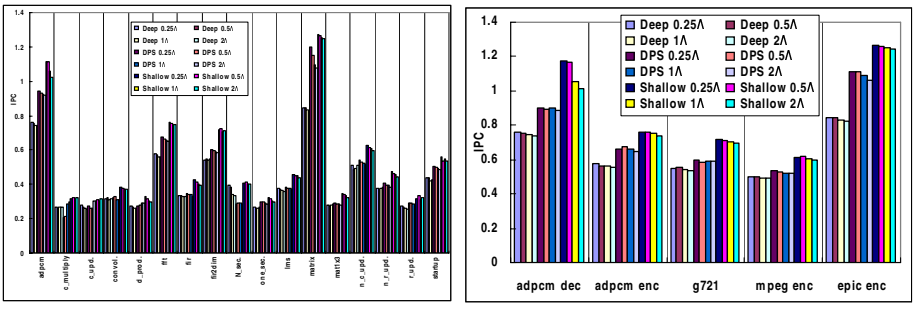


Fig. 5. Chang in IPC for three cases using DSPstone and Mediabench as benchmark

small $\alpha\lambda$ can have better energy reduction than larger ones. From Figure 4b, the bar of 0.25λ has the best energy saving and 2λ is the worst. Compared to the baseline version, our DPS approach can save energy by 12% to 28% and an average of 18.2%. Notice that if a program has a very large loop, it may contain only one non-loop region or two very small non-loop regions. In this case, the value of $\alpha\lambda$ will not influence our experimental result.

Figure 5 demonstrates the effect on IPC for three cases using DSPstone and Mediabench as the metric. Deep and Shallow represent the results of executing a program in deep and shallow pipeline modes respectively; DPS indicates the result of applying our DPS approach to a program. They are still measured for various partitioning sizes of non-loop regions $0.25\lambda, 0.5\lambda, \lambda,$ and, 2λ. For DSPstone, the average IPCs of Deep case and Shallow case are 0.4 and 0.52. In DPS case, the average IPC is 0.45, which is between those of deep mode and shallow mode. For Mediabench, the average IPCs of Deep case, Shallow case, and DPS are 0.63, 0.87, and 0.75. Normally, the IPC of deep mode is larger than that of shallow mode, but this contradicts with the results shown in Figure 5. The reason comes from a key observation that branch penalty is closely related to IPC. Since shallow mode has smaller branch penalty than deep mode, as a result its IPC become larger than that of deep mode. In DSPstone, the IPCs of adpcm, fft, fir2dim, and matrix are larger. This is because since they are loop-intensive applications and the loops in them contribute a lot to the increase of IPC. In Mediabench, the IPCs of adpcm decoder, adpcm encoder, g721, mpeg encoder and epic encoder are larger. This is because the five chosen programs are loop-intensive applications and the loops in them contribute a lot to the increase of IPC.

Figure 6 show the effect of our DPS approach on performance for DSPstone and Mediabench. The latency between pipelining stages is designed to be equivalent to increase performance and achieve resource sharing at each clock. In theory, the performance is in direct ratio to the number of pipelining stages and thus the longer pipeline will lead to the performance. Thus, the processor will result in slowdown when executing in shallow mode. The performance will be degraded if the pipelining stages are merged into shallow mode. In reality, the performance may be degraded due to many factors such as pipelining hazards, branch penalty, or switching overhead between pipeline modes. For each benchmark, the performance of deep mode with λ is the baseline to compare those

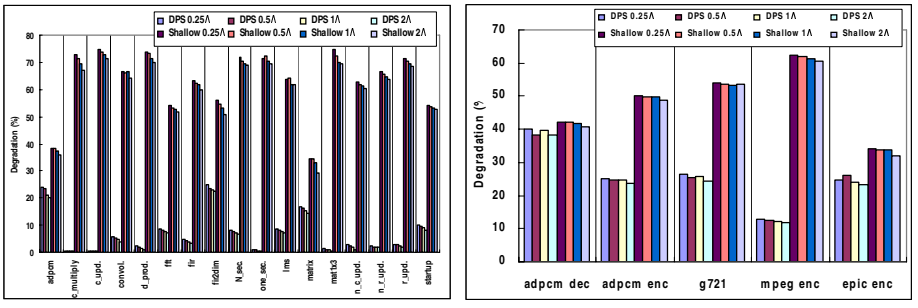


Fig. 6. Relative performance of various Λs for DSPstone & Mediabench

of deep mode and our DPS method for various λs. For the performance of DSPstone in Figure 6a, on average, our DPS approach leads to 6.23% degradation in performance. By contrast, the performance degradation of shallow pipeline mode is 61.62%, which is almost ten times larger than the above one. Although the DPS switches the pipelining modes based on the IPC to save energy, the switching slows down the processor compared to the high-speed execution in deep mode. In addition, larger λ has a better performance than smaller ones since it causes the pipeline to enter the shallow mode more infrequently. For Mediabench, on average, the shallow mode and profile-based DPS have 48.02% and 25.23% performance degradations.

5 Conclusions

DVS has been proven be very effective in low power optimizations, but it cannot further save energy when the voltage reaches its lower bound. Fortunately, DPS can overcome this limitation by adjusting pipeline modes based on IPC. Previous work resolved this issue with hardware techniques and thus increased hardware cost and design complexity. In this paper, we present a DPS technique to reduce power dissipation by proposing an evaluating model so that they can decide the timing of entering the proper pipeline mode. In contrast, our work can eliminate hardware overhead and reduce energy consumption according to the code behavior at compiler time. To investigate the effect of our approach, we perform the experiment with various criteria for DSPstone and Medieabench. In summary, the results show that smaller partitioning sizes of non-loop regions can create optimization space and loop-intensive applications provide more chances to optimize code to save energy.

References

[1] Brooks, D., Tiwari, V., Martonosi, M.: Wattch: A framework for architectural level power analysis and optimizations. In: International Symposium on Computer Architecture (2000)
[2] Burd, T., Brodersen, R.: Design issues for dynamic voltage scaling. In: International Symposium on Low Power Electronics and Design (2000)

[3] Efthymiou, A., Garside, J.D.: Adaptive pipeline depth control for processor power-management. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors (2002)
[4] Ernst, D., Kim, N.S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., Mudge, T.: Razor: A low-power pipeline based on circuit-level timing speculation. In: The 36th International Symposium on Microarchitecture (2003)
[5] Gerndt, M.: Automatic parallelization for distributed-memory multiprocessing systems. In: Phd Thesis (1989)
[6] Hartstein, A., Puzak, T.R.: The optimum pipeline depth for a microprocessor. In: ACM/IEEE International Symposium on Computer Architecture (2002)
[7] Hiraki, M., Bajwa, R.S., Kojima, H., Corny, D.J., Nitta, K., Shridhar, A., Sasaki, K., Seki, K.: Stage-skip pipeline: A low power processor architecture using a decoded instruction buffer. In: International Symposium on Low Power Electronics and Design (1996)
[8] Hsu, C., Kremer, U.: The design, implementation, and evaluation of a com-piler algorithm for cpu power reduction. In: The ACM SIGPLAN Conference on Programming Languages Design and Implementation (2003)
[9] Kessler, R., McLellan, E., Webb, D.: The alpha 21264 microprocessor architecture. In: Intl. Conf. Computer Design (1998)
[10] Koppanalil, J., Ramrakhyani, P., Desai, S., Vaidyanathan, A., Rotenberg, E.: A case for dynamic pipeline scaling. In: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (2002)
[11] Kornerup, J.: Mapping powerlists onto hypercubes. In Masters Thesis (1994)
[12] Krishna, C., Lee, Y.-H.: Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In: The 6th Real Time Technology and Applications Symposium (2000)
[13] Lilia, D.: Reducing the branch penalty in pipelined processors. IEEE Computer 21(7), 47–55 (1988)
[14] Manne, S., Grunwald, D., Klauser, A.: Pipeline gating: Speculation control for power reduction. In: ACM/IEEE International Symposium on Computer Architecture (1998)
[15] Parikh, D., Skadron, K., Zhang, Y., Stan, M.: Power-aware branch prediction: Characterization and design. The IEEE Transactions on Computers (2004)
[16] Suif, G.: Stanford University Intermediate Format, http://suif.stanford.edu

Parallel Network Intrusion Detection on Reconfigurable Platforms*

Chun Jason Xue[1], Zili Shao[2], MeiLin Liu[3], QingFeng Zhuge[4], and Edwin H.-M. Sha[4]

[1] City University of Hong Kong, Kowloon, Hong Kong
jasonxue@cityu.edu.hk
[2] Hong Kong Polytechnic University, Hong Kong
cszlshao@comp.polyu.edu.hk
[3] Wright State University, Dayton, Ohio 45435, USA
meilin.liu@wright.edu
[4] University of Texas at Dallas, Richardson, Texas 75083, USA
{qingfeng,edsha}@utdallas.edu

Abstract. With the wide adoption of internet into our everyday lives, internet security becomes an important issue. Intrusion detection at the network level is an effective way of stopping malicious attacks at the source and preventing viruses and worms from wide spreading. The key component in a successful network intrusion detection system is a high performance pattern matching engine that can uncover the malicious activities in real time. In this paper, we propose a highly parallel, scalable hardware based network intrusion detection system, that can handle variable pattern length efficiently and effectively. Pattern matchings are completed in O(log M) time where M is the longest pattern length. Implementation is done on a standard off-the-shelf FPGA. Comparison with the other techniques shows promising results.

1 Introduction

Network Intrusion Detection System (NIDS) performs packet inspection to identify, prevent and inhibit malicious attacks over internet. It can effectively stop viruses, worms, and spams from wide spreading. Pattern matching is the key component in the network intrusion detection systems. Using modern reconfigurable platforms, like FPGA, design and implement a parallel, high performance pattern matching engine for network intrusion detection is the goal of this paper.

Traditionally, network intrusion detection systems are implemented completely in software. Snort [20] is a well-known open source software network intrusion detection system. It matches pattern database against each packet to identify malicious target connections. With the rapid growth of pattern database, and the rapid growth of network bandwidth, software only solution can not process the

* This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, USA and HK PolyU A-PH13, A-PA5X and A-PH41.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 75–86, 2007.

internet traffic in full network link speed. A natural approach will be to move the computation intensive pattern matching to hardware. The main idea is to use specialized hardware resources along with a conventional processor. In this way, the conventional CPU can process all the general-computing tasks and the specialized co-processor can deal with string pattern matching, where parallelism, regularity of computations can be exploit by custom hardware resources.

Extensive researches exist on general pattern matching algorithms. The Boyer-Moore algorithm [6] is widely used for its efficiency in single-pattern matching problems. However, the current implementation of Boyer-Moore in Snort is not efficient in seeking multiple patterns from given payloads [3]. Aho and Corasick [1] proposed an algorithm for concurrently matching multiple strings. Their algorithm uses the structure of a finite automation that accepts all strings in the set. Two implementations of the Aho-Corasick algorithm have been done for Snort, by Mike Fisk [10] and Marc Norton [17], respectively. Fisk and Varghese [11] presented a multiple-pattern search algorithm that combines the one-pass approach of Aho-Corasick with the skipping feature of Boyer-Moore as optimized for the average by Horspool. The work by Tuck, et al. [22] takes a different approach to optimizing Aho-Corasick by instead looking at bitmap compression and path compression to reduce the amount of memory needed. All these approaches are developed mainly for software implementation. To examine packets in real time with full network link speed, a hardware solution is more favorable.

There are two main groups of hardware solutions for fast pattern matching. The first group generally applies finite state machine (FSM) to process patterns in sets. Aldwairi et al. [2] designed a memory based accelerator based on the Aho-Corasick algorithm. In their work, rules are divided into smaller sets that generate separate FSMs which can run in parallel. Hence significantly reduces the size of state tables and increases the throughput. Liu et al. [15] designed and implemented a fast string matching algorithm on network processor. Baker and Prasanna [4] proposed a pipelined, buffered implementation of the Knuth-Morris-Pratt algorithm [13] on FPGA. Li et al. [14] implemented rule clustering for fast pattern matching based on [9] with FPGA platform. Pattern size is limited by the available hardware resources. Tan and Sherwood [21] designed special purpose architecture working in conjunction with string matching algorithms optimized for the architecture. Performance improvement in this first group is generally achieved by dividing patterns into smaller sets, and deeply pipelining the pattern matching process. However, these type of approaches all have the shortcoming in scalability, when the pattern database grows exponentially, these type of approaches will suffer from extensive resources consumption and not able to maintain the same level of performance. Deep pipelining also has the side effect of increased latency, which is detrimental to some internet traffic.

The second group of hardware solutions uses hash tables as the foundation for pattern matchings. Dharmapurikar et al. [8] used bloom filters [5] to perform string matching. The strings are compressed by calculating multiple hash functions over each string. The compressed set of strings is then stored into a small memory which is queried to find out whether a given string belongs to

the set. If a string is found to be a member of the bloom filter, it is declared as a possible match and a hash table or regular matching algorithm is needed to verify the membership. Song and Lockwood [19] proposed a new version of the implementation that eliminates the need of hash table for match verification. Both of these implementation are done on FPGAs. Since patterns have different length, a bloom filter is constructed for each pattern length. This is simple but not efficient in handling variable pattern length.

This paper propose a novel hardware solution for pattern matching in NIDS. Our approach uses hash tables. However, we handle variable pattern length naturally from the beginning. Basically, we will slice each pattern into substrings of length 2^i, where $0 <= i <= k$, $k = \lfloor \log(M) \rfloor$, and M is the maximum pattern length. A hash table will be constructed for each substring length. There will be a total of k number of hash tables. Input string is processed in an iterative fashion. First, all substrings of length 2^k of the input string is matched against the hash table for substring length 2^k. Then, all substrings of length 2^{k-1} of the input string is matched against the hash table for substring length 2^{k-1}. Until all substrings of length one is matched against hash table for substring length one. A match is declared when all substrings of a pattern are matched. An extremely simple example is shown in Figure 1 to illustrate our idea.

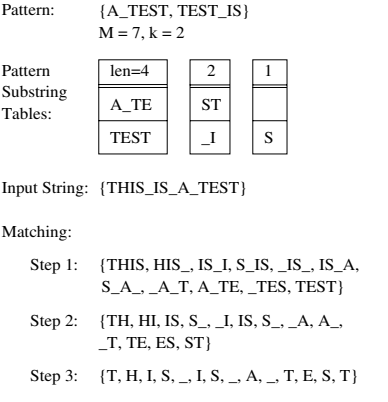


Fig. 1. An example

In this example, there are only two patterns to be matched, "A_TEST" and "TEST_IS". The maximum pattern length $M = 7$. Hence $k = \lfloor \log(7) \rfloor = 2$. We slice patterns into substrings of length 2^2, 2^1, and 2^0. Pattern substring tables are built for each substring length. In this example, we show the exact value of the substring for illustration purpose. In reality, these tables will be hash tables for speed matching. An example input string is also shown in Figure 1. Using our approach, the matching is done in 3 steps. In step 1, all substrings of length 4 of the input string are matched against the pattern substring table with length equals 4. In step 2, all substrings of length 2 of the input string are matched against the pattern substring table with length equals 2. In step 3, all substrings

of length 1 of the input string are matched against the pattern substring table with length equals 1. A match of pattern "A_TEST" is declared when both substring "A_TE" and substring "ST" are matched during the process. The detail of the matching process and the data structures used will be presented in the rest of this paper.

In this paper, we use M to represent the maximum pattern length, example value of M could be 256 or 512. We use N to represent the number of patterns. A typical N would be 2k, which can fit the current snort rule set [20]. The main research contributions of this paper are:

- Handles variable pattern length efficiently and effectively while using hash tables.
- Finishes matching in O(log M) steps, where M is the maximum pattern length.
- Excellent scalability. Pattern matching performance is not affected by the growth of pattern database.

The remainder of this paper is organized as follows. The architecture of our technique is shown in Section 2. The concepts and data structures used in our approach are introduced in Section 3. Section 4 presents the algorithms. Section 5 presents the implementation on a reconfigurable platform. Section 6 presents the concluding remarks.

2 Architecture

The block diagram of our proposed architecture is shown in Figure 2. In this architecture, the core elements are an array of PEs (Processing Element). The number of PEs equals to the size of the input string S. A PE processes a substring of the input against all the same length substrings of the patterns. The input string is processed in rounds of different substring length. Each PE will first process all the 2^k bytes substring of the input string, then 2^{k-1}, etc.

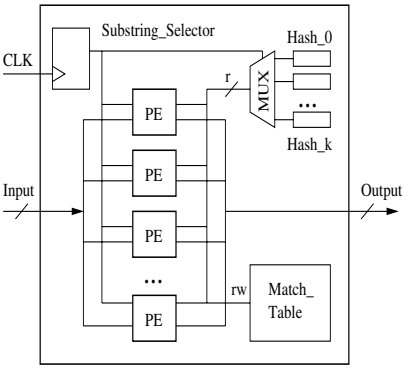


Fig. 2. Architecture Diagram

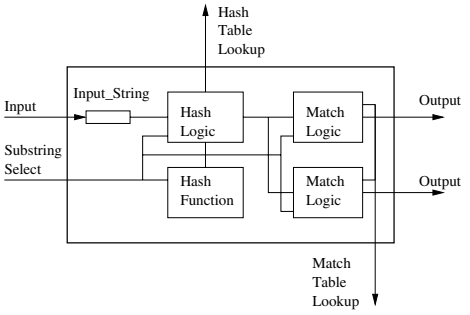


Fig. 3. A Processing Element (PE)

The design diagram of a PE is shown in Figure 3. The inputs of a PE are a substring and a substring select signal that determines the length of the substring that will be worked on. First the input string will be passed to the hash function block and a hashing value will be obtained. This hash value will be used to do a hash table lookup. The result of hash table lookup will be passed to the match logic block to determine if there is a match or not. The design of each PE is kept simple. Duplicated hardware is used for the Match Logic block to increase the performance.

3 Basic Concepts and Data Structures

In this section, we introduce the basic concepts which will be used in the later sections. The data structures used in our approaches are also presented in this section.

Let us first define the problem that we are trying to solve. Assuming a packet carries a string S of length L, and we know a set of N patterns, $p[1], p[2], ..., p[N]$, the goal of Network Intrusion Detection System (NIDS) is to determine if there is any exact matching between pattern $p[i]$ and a substring of S. Let M be the maximum pattern length, and let $k = \lfloor logM \rfloor$. The main idea of our approach is to slice each pattern into substrings of length 2^i, where $0 \leq i \leq k$. Input data string S is read in as a whole and processed in rounds of different substring length. First all substrings of length 2^k are processed, then all substrings of length 2^{k-1}, etc. The whole matching are completed in k steps.

After finding a match of a substring, we will first decide if all the previous substrings in the pattern are matched, If yes, then a partial match is identified. And then, we will see if this is the last substring in the partially matched pattern. If yes, then an potential exact match is declared and a red flag will be raised by the network intrusion detection system and processed accordingly by the host system.

Three sets of data structures are used in our approach, and we will introduce them one by one. The first data structure of interest is the [illegible] table. It is an array that stores each pattern's length and indexed by the pattern ID.

Pattern ID	Pattern Length	32	16	8	4	2	1
1	33	1	0	0	0	0	1
2	5	0	0	0	1	0	1
3	10	0	0	1	0	1	0
4	17	0	1	0	0	0	1

Fig. 4. Pattern_Length table

The binary representation of each pattern length shows what substrings that this pattern will be decomposed into. An example is shown in Figure 4. In this example, for the first pattern with pattern ID equals to 1 and length equals to 33, it will be sliced into a substring of length 32 and a substring of length 1, as depicted by its binary representation in Figure 4.

The second set of data structure of interest is a set of hash tables that stores the pre-processed information for each substrings of each patterns. For pattern substrings of length 1, since there can only be 256 values, no hashing is done. Instead, a table of 256 entries is created. Each entry contains three elements, the first element is the value of this entry, the second element is the starting pattern ID, and the third element is the number of patterns that have the same value from the starting pattern ID. An example is shown in Figure 5. In this example, there are three patterns with value "a" as the last byte. Hence, in the _ table, there is an entry with value equal to "a", starting pattern ID equal to 100, and number of consecutive patterns equal to 3.

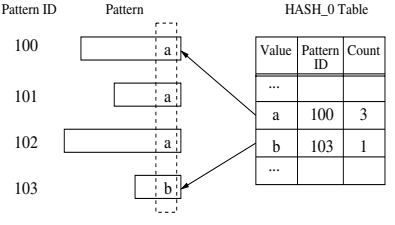
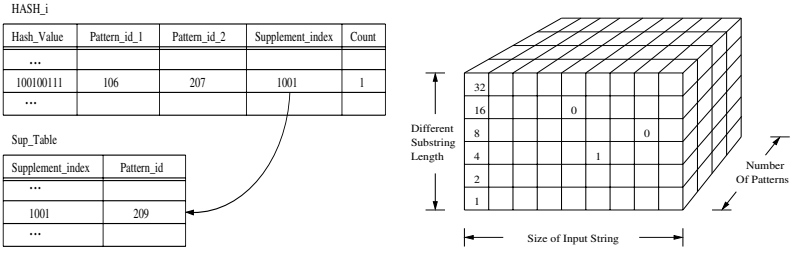


Fig. 5. HASH_0 table

For substring length greater than 1, a hash table is constructed for each substring length. Hash table _ correspond to substring length 2^i, where $i \neq 0$. Index of each hash table is the hashing value, and the entries in the hash tables are the pattern IDs. An example of hash table when substring length not equal to zero is shown in Figure 6(a). There are five columns in each hash table. Extra columns are used to handle hashing collisions. There are two sources of potential hashing collisions exist in our scheme. First, different substrings could be hashed to the same hash value. Second, different patterns could have the same substring. For example, pattern "hell" and pattern "hello" have the same 4 bytes substring

"hell". To handle hashing collisions efficiently, for each hash value, we reserve two space for pattern ID in column two and column three respectively. These two pattern ID will be read in the same clock cycle and processed by hardware simultaneously. When there are more than 2 substrings are hashed to the same value, a separate table called is used to record these values. is also shown in Figure 6(a). Column four of the table points to the starting Supplement_index, and column five identify the number of consecutive entries in the that have the same hash value. In the example shown in Figure 6(a), for hash value "100100111", there are three patterns total have this hash value, pattern 106, pattern 207 and pattern 209 as recorded in in entry 1001.



(a) HASH_i table and Sup_Table (b) Match_Table

Fig. 6. HASH Table and Match Table

The third data structure that we use is the , which is a three-dimensional bit array, with length equals to the input string length L, width equals to the number of patterns N, and the height equals to number of different substring length k. This table is used to record the substring matches found, which is in turn used for determining whole pattern match. For each substring match, a "1" will be recorded using the substring length, matched pattern id, and the position of the substring in the input string S. An example is showing in Figure 6(b). In this example, there are six different substring length, 1, 2, 4, 8, 16, and 32. Hence has a height of 6.

4 Algorithms

In this section, the algorithms of our approach are presented. An example is given at the end of this section to show how the algorithms work. There are two main algorithms in our approach. Algorithm shown in Algorithm 4.1 handles the initialization of all the necessary data structures. The second algorithm shown in Algorithm 4.2 processes the input strings for potential matchings.

In algorithm , first all the odd length patterns are sorted by the value of the last byte. This is necessary for building the lookup table for

Algorithm 4.1. Init_Matching

Require: A set of patterns p.
Ensure: Initialized data structures.

```
Sort all the odd length patterns by the value of the last byte;
for all pattern p[i] do
  Pattern_Length[i] = length(p[i]);
end for
for all pattern p[i] do
  for each substring s in p[i] do
    hashed_value = HASH(s);
    set HASH_j[hashed_value] = i;
  end for
end for
for j = 0 to 255 do
  Insert Starting Pattern ID and number of patterns into HASH_0 ;
end for
```

substring length 1. Then for each pattern, [illegible] table is populated with the length of the pattern. Afterward, we will hash each substring of each pattern, and store the pattern ID accordingly. Based on our [illegible] table, there are two spaces to store pattern ID. We will first try to store the pattern ID of a particular hash value in one of these two spaces. If both of these two spaces are occupied, we will then place the pattern ID in the [illegible] and update the last two columns of the [illegible] table accordingly. The last step of the [illegible] algorithm populates the [illegible] table with the sorted pattern information. Updating the pattern set when we need to add or remove a pattern can be done in the similar fashion of Algorithm 4.1.

The main algorithm that processes each input string for potential matching patterns is algorithm [illegible]. There are two functions notable used in Algorithm 4.2 , i.e., Pre_Substring(pl,i) and Post_Substring(pl,i), where pl is the pattern length and i is the current substring length. These two functions are used to determine if there is other substrings in the current pattern or not. If there are substrings before the current substring with length i in a pattern of length pl, Pre_Substring(pl,i) will return the previous substring length. Otherwise, Pre_Substring(pl,i) will return "0". Post_Substring(pl,i) will return "1" if there is any substring after the current substring with length i, and return "0" if the current substring is the last substring of the pattern. In [illegible] algorithm, for each substring length and each substring, we will first run the hash function to obtain a hash value. The hash value is used to lookup the corresponding hash table. If there are matches found in the hash table, for each matched pattern ID, we will examine its previous substrings and post substrings. If there is no previous substring or if there is a previous substring and it is also matched to the same pattern, we will mark "1" in the $Match\_Table$ for this input substring, at this substring length and this matched pattern. After we mark "1" in the Match_Table, if this substring also happens to be the last substring of the pattern, then we declare there is a potential match. Hash function

Algorithm 4.2. Pattern_Matching

Require: Input string S of length L.
Ensure: Yes/No. If there is a substring in the input string S that matches one pattern.

```
for all substring length i do
  for all substring starting at position j of S do
    hashed_value ← HASH(substring);
    for each match in HASH_i; do
      k ← matched pattern ID;
      /* Find the pattern length for pattern k */;
      pl ← Lookup the Pattern_Length table for pattern k;
      /* Find the previous substring for pattern k */;
      pre_s ← Pre_Substring(pl,i);
      if (pre_s = 0) or ( pre_s > 0 and Match_Table[j − pre_s][pre_s][k] = 1) then
        Match_Table[j][i][k] = 1;
        if Post_Substring(pl,i) = 0 then
          Return Match_found= 1;
        end if
      end if
    end for
  end for
end for
```

is used heavily in our approach. Implementing hashing in hardware is relatively inexpensive. A class of universal hash functions called H_3 described in [18] were found to be suitable for hardware implementation. Our implementation of hash function falls into this class.

An example of the matching process is shown in Figure 7. This is a continuation of the simple example shown in the introduction section. The detail of the $Match\_Table$ is shown in Figure 7. In this example, there are one input string S that has 14 bytes, two patterns to be matched, and three different substring

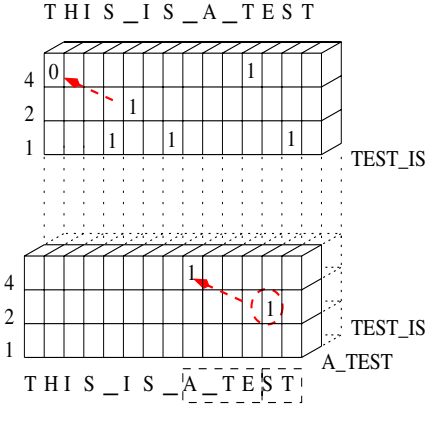


Fig. 7. Matching Example

Table 1. Comparison of throughput, unit size, and performance

Design	Throughput	Unit Size (Logic cells)	Performance (Mb/s/logic cell)
UTD	4.7Gb/s	232	20.3
USC(no pipelining) [4]	1.8Gb/s	92	19.6
USC(pipeline)[4]	2.4Gb/s	120	20.0
Los Alamos [12]	2.2Gb/s	243	9.1
Wash U. - DFA [16]	0.952Gb/s	260	3.7
Wash U. - Bloom [8]	0.8Gb/s	0.76	1058
UCLA [7]	2.88Gb/s	160	18.0

length. During the first round, where we match all substrings of length 4, there are two matches, one for "A_TE" and one for "TEST". Both matches lead to a marked "1" in the matching table. Moving on to the second round, where we match all substrings of length 2. Substring "ST" is matched, and since the previous substring of the same pattern is also marked as matched to the same pattern, "1" is marked for the "ST" substring match. Since substring "ST" has no substring after it, a match is declared. There is also a substring match of "_I" found, however, since the previous substring is not marked as matched, we do not mark "1" in the match table for the location where "_I" is matched.

5 Implementation

We have described our design in VHDL and targeted it to the Xilinx Virtex II architecture with -7 speed grade. We use the Xilinx ISE 7.1i and Mentor Graphic ModelSim 6.0 development tools. We have implemented a linear array of these PEs. Using a Virtex II XC2V6000, we are able to accommodate 128 PEs. This allows us to handle input string length of 128 bytes. The corresponding clock frequencies are 220 MHz. Since we need an average of six clock cycles to complete pattern matching for 128 bytes, hence our average throughput is 0.22x128/6 = 4.7 Gb/s.

Memory consideration in the implementation is very important in achieving high performance. In our design, [illegible] is the key in consolidating partial matches from substrings into full matches. Concurrent read/write accesses to the [illegible] could be the bottleneck of our performance. In the real implementation, we actually slice the match table into thin slices. As shown in Figure 8, we can assign one slice of match table per PE. Each PE will write to its own slice of match table, and read from other slices of the match tables if needed. So the memory design requirement of the [illegible] becomes single write, multiple read instead of multiple write, multiple read. For our implementation on Xilinx Virtex II, we mapped each slice of match table into one 18kb Block SelectRAM. There are 3.5 Mb of total memory constituted by these 18kb Block SelectRAM on a chip [23]. We can fit all 128 slices of match tables easily. Memory implementation for the hash tables can be optimized in the same fashion. We can duplicate multiple copies of the hash tables and distribute among

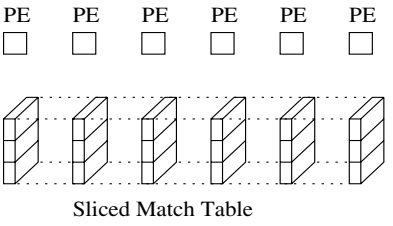


Fig. 8. Memory mapping for match_table

the PEs. Since we only need to read from the hash tables during the matching process, each copy of hash tables can be implemented using multi-port memory and shared among several PEs.

Performance of our system can be further improved with the availability of more hardware resources. There are two ways that this performance gain could take place. First, we could use a larger FPGA that can accommodate 6x128 PEs. In this way, input strings can be processed in a pipelined fashion. At every clock cycle, there will be a 128 bytes string input and a 128 bytes string output. Second, multiple copies of the current design can be used in parallel to process multiple input streams at the same time. Either way, scalability can be achieved easily with the addition of new hardware resources.

The throughput, unit size, and performance of our design is compared with several other designs in Table 1. While generating high throughput, our design works relatively well in the unit size and performance. The real strength of our design comes when the number of patterns grows significantly and the speed of network increases dramatically, we do not have to make huge change in our design, only increase in hardware resources will make our design scale as needed.

6 Conclusion

In this paper, we propose a new hardware solution for NIDS. Our solution can handle variable pattern length efficiently while using the hash function approach. Pattern matching is processed in O(log M) steps, where M is the maximum pattern length. Enabling fast pattern matching is the key component in successful network intrusion detection. As a next step, we plan to explore beyond exact pattern matching, identifying threats that are not exactly the same as the known patterns, but are variants of the known patterns.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search 18, 333–340 (1975)
2. Aldwairi, M., Conte, T., Franzon, P.: Configurable string matching hardware for speeding up intrusion detection, pp. 99–107 (2005)

3. Anagnostakis, K.G., Antonatos, S., Markatos, E., Polychronakis, M.: E2xb: A domain-specific string matching algorithm for intrusion detection (2003)
4. Baker, Z.K.: Time and area efficient pattern matching on fpgas, 223–232 (2004)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors 13, 422–426 (1970)
6. Boyer, R.S., Moore, J.S.: A fast string searching algorithm 20, 762–772 (1977)
7. Cho, Y., Navab, S., Mangione-Smith, W.: Specialized hardware for deep network packet filtering (September 2002)
8. Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J.: Deep packet inspection using parallel bloom filters (2003)
9. Kruegel, C., et al.: Automatic rule clustering for improved, signature based intrusion detection (2002)
10. Fisk, M., Varghese, G.: An analysis of fast string matching applied to content-based forwarding and intrusion detection (2002)
11. Fisk, M., Varghese, G.: Applying fast string matching to intrusion detection (2004)
12. Gokhake, M., Dubois, D., Dubois, A., Boorman, M., Poole, S., Hogsett, V.: Towards gigabit rate network intrusion detection (2002)
13. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in string (1977)
14. Li, S., Torresen, J., Soraasen, O.: Exploiting reconfigurable hardware for network security (2003)
15. Liu, R.-T., Huang, N.-F., Chen, C.-H., Kao, C.-N.: A fast string-matching algorithm for nerwork processor-based intrusion detection system, 614–633 (2004)
16. Moscola, J., Lockwood, J., Loui, R.P., Pachos, M.: Implementation of a content-scanning module for an interenet firewall (2003)
17. Norton, M., Roelker, D.: Snort 2.0: Detection revised (2002)
18. Ramakrishna, M., Fu, E., Bahcekapili, E.: A performance study of hashing functions for hardware applications, 1621–1636 (1994)
19. Song, H., Lockwood, J.: Multi-pattern signature matching for hardware network intrusion detection systems (2005)
20. Sourcefire. Snort: The Open Source Network Intrusion Detection System (2003)
21. Tan, L., Sherwood, T.: A high throughput string matching architecture for intrusion detection and prevention, 112–122 (2005)
22. Tuck, N., Sherwood, T., Calder, B., Varghese, G.: Deterministic memory-efficient string matching algorithms for intrusion detection (March 2004)
23. Xilinx, Inc. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet (2004)

Evaluating Mobility Support in ZigBee Networks*

Tony Sun[1], Nia-Chiang Liang[1], Ling-Jyh Chen[2], Ping-Chieh Chen[1], and Mario Gerla[1]

[1] Department of Computer Science, University of California at Los Angeles
[2] Institute of Information Science, Academia Sinica

Abstract. The deployment of ZigBee networks is expected to facilitate numerous applications, such as home healthcare, medical monitoring, consumer electronics, and environmental sensors. For many envisioned applications, device mobility is unavoidable and must be accommodated. Therefore, providing ubiquitous connection to/from a mobile ZigBee device is crucial for future ZigBee applications. In particular, knowledge of how nodal mobility affects ZigBee routing protocol is of significance. In this paper, our contributions are twofold. First, we dissect ZigBee routing and its support for device mobility, and we analyzed the current provisions in dealing with different mobility cases. Second, we performed a rich set of preliminary tests, illustrating the inefficacy of current standard. Our results indicate that ZigBee device type plays a significant role in determining the routing performance in most mobile scenarios.

Keywords: Mobility, Routing, ZigBee, IEEE 802.15.4, Simulation.

1 Introduction

With wireless networking technologies permeating into the very fabrics of our working and living environment, simple appliances and numerous traditional wired services can now be efficiently connected wirelessly. This provides simple yet effective control/monitoring conveniences, while allowing very interesting applications to be developed on top of these wireless network enabled gadgets. The ZigBee standard [2], designed to interconnect simple devices, is the latest attempt to realize this wireless network vision. In the context of a business environment, this wireless technology can facilitate better automated control/management of facilities and assets. Additionally, there are also many ZigBee applications for home-appliance networks,home healthcare, medical monitoring, consumer electronics, and environmental sensors.

For an environment richly connected with ZigBee devices, drastic topological changes can occur due to device failures, mobility, and other factors. For certain applications, device mobility is unavoidable. For example, a health monitoring application for the elderly described in [4] [3], where a ZigBee enabled health monitoring sensor alerts the hospital, through an adjacent network, when a health-related emergency has occurred. The consequence is disastrous if the message was not delivered as intended. Therefore, understanding the performance of ZigBee networks becomes

* This work was co-sponsored by the National Science Council and the National Science Foundation under grant numbers NSC 95-2218-E-002-072 and ANI-0335302.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 87–100, 2007.

important in determining the applicability of many applications. In particular, knowledge of how nodal mobility affects the workings of the ZigBee routing protocols is of significance.

Without a doubt, mobility support is important to the proper functioning of many envisioned ZigBee applications. Since mobility is anticipated and unavoidable, adequate mobility support is important in ensuring ubiquitous connection to/from the mobile devices. In this study, our contribution is twofold. First, we dissected ZigBee routing and its current support for device mobility. It is the goal of this study to identify the existing provisions in accommodating ZigBee device mobility, and to analyze the adequacy of these provisions in dealing with different mobility cases. Secondly, we ran a rich set of preliminary simulations, illustrating the inefficacy of current standard in handling mobility. Our results reveal that existing ZigBee provisions for mobility is inadequate, and mobility problem was not thoroughly considered by the standard. Moreover, we found that the current recovery mechanisms are not reliable, or responsive enough in all mobility cases. Finally, we found that the situation worsens when there are multiple instances of mobility in the ZigBee network, yet routing performance in ZigBee network is closely tied to the ZigBee node types used.

The rest of this paper is organized as follows. In section 2, we summarize the IEEE 802.15.4 and the ZigBee specifications. Section 3 discusses the routing and address allocation mechanism deployed in ZigBee mesh routing, and analyzes the response of the routing protocol in basic mobility cases. Section 4 is an equitable treatment to the ZigBee tree routing mechanism, where the behavior of tree routing is explained and analyzed for basic mobility scenarios. Section 5 presents a rich set of preliminary simulation results. Finally, in section 6, we discuss the tradeoff between the two routing mechanism in dealing with mobility and conclude the paper.

2 Overview

2.1 IEEE 802.15.4

Based on the PHY and MAC layers specified by IEEE 802.15.4 WPAN standard [1], the ZigBee specification establishes the framework for the Network and Application layers. Specifically,at the MAC layer, IEEE 802.15.4 controls access to the radio channel using the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) or the optional slotted CSMA/CA mechanism, as respectively utilized by the beaconless and beaconed modes.

Two device types are specified within the IEEE 802.15.4 framework: full function device (FFD) and reduced function device (RFD). An FFD generally have more responsibility in that they must maintain routing tables, participate in route discovery and repair, maintain beaconing framework, and handle node joins. Moreover, a FFD have the capability of communicating with any other devices within its transmission range. On the other hand, a RFD simply maintains the minimum amount of knowledge to stay on the network, and it does not participate in routing.

2.2 ZigBee Network Layer

The ZigBee network layer defines how the network formation is performed (i.e., either mesh topology or tree topology) and how the network address is assigned to each participating ZigBee node. Note that the assigned network address is the *only* address that is used for routing and data transmission in ZigBee networks. Three device types are defined in ZigBee: ZigBee coordinator, ZigBee routers, and ZigBee end devices. A RFD can only be a ZigBee end device; whereas a FFD can be either a ZigBee coordinator or ZigBee router. The ZigBee coordinator is responsible for starting a new network. ZigBee coordinator and routers are "routing capable", while the ZigBee end devices can't participate in routing and have to rely on their corresponding ZigBee parent routers for that functionality.

Every node in a ZigBee network has two addresses, namely a 16-bit short network address and a 64-bit IEEE extended address. The 16-bit network address is assigned to each node dynamically by its parent coordinator/router upon joining the network. *This address is the only address that is used for routing and data transmission*. It is analogous to the IP addresses that we use on the internet; whereas the extended address is similar to the MAC address, which is a unique identification of each device and is mostly fixed at the time the device is manufactured.

3 Mobility Support in ZigBee Mesh Topology

As pointed out in section 2, only coordinators/routers (FFDs) can actively participate in mesh routing, end devices (RFDs) must rely on their parent nodes to perform mesh routing on their behalves. Under the innate properties of IEEE 802.15.4 and ZigBee networks (i.e., the addressing structure and service assumptions), the performance bound of ZigBee mesh routing is expected different to the ones from previous AODV studies. We will see the effect of this recovery mechanism in more detail in section 5.

3.1 Mobile End Device

As mentioned earlier, ZigBee end devices are just simple devices without routing capabilities. Therefore, problems arise whenever these end devices moves out of the range of its parent router, and acquires a new network address from a new parent router, while the source node is still sending data to the mobile end device. Since this end device can no longer be found with its "old" address, data reception in this scenario will be halt completely, and can't be recovered from any available ZigBee mesh routing mechanisms. When the route cannot be found, a route error message will eventually be delivered to the source node, and trigger the Device Discovery primitive in the application layer. Once the source node discovers the new network address of the destination, the data transmission would resume (after another route discovery procedure). For this simple case, the data flow would only suffer the duration required for the source to receive the route error and complete its device discovery process.

For the case where the mobile end node acquires a new network address while it is sending data, data transmission will be temporally disrupted for the duration it takes for

the mobile end node to find a new parent router to associate itself with. If the data flow is two way, a route discovery and Device Discovery process would be triggered at the receiver, and the disruption would be compounded, yet recoverable assuming that the mobile end device doesn't move out of range again.

3.2 Mobile Router

ZigBee routers actively participate in mesh routing, and provide functionalities that maintain/repair routes whenever an existing route failed. With the built-in route recovery mechanism (via route discovery and route error), ZigBee routers remains robust to effects from most mobility cases regardless whether the node is sending or receiving data. Once the router is assigned an initial network address, this is no explicit need to change this address.

4 Mobility Support in ZigBee Tree Topology

For ZigBee tree topology, the network address is assigned based on a hierarchical tree structure. As the root of the cluster tree, the ZigBee coordinator is responsible in defining the number children node a parent may have $nwkMaxChildren(C_m)$, the maximum number of routers a parent may have as children $nwkMaxRouters(R_m)$, and the maximum depth of the network $nwkMaxDepth(L_m)$. For a network of depth d, the n^{th} network address is allocated according to Eq. 2 and 1.

$$C_{skip}(d) = \begin{cases} 1 + C_m(L_m - d - 1) & , R_m = 1 \\ \frac{1+C_m-R_m-C_m(R_m)^{L_m-d-1}}{1-R_m} & , else \end{cases} \tag{1}$$

$$A_n = A_{parent} + C_{skip}(d) * R_m + n \tag{2}$$

After network address is assigned, any node can then route packets to its parent and direct children with the tree routing algorithm. Trivially, every other device in the network is a descendant of the ZigBee coordinator and no device in the network is the descendant of any ZigBee end device. Each node would check the destination address against its own to determine if the destination is a descendent on the tree or if it should be forwarded to its parent node. When a node changes its parent router due to mobility, a new 16-bit network address will be automatically assigned to preserve the tree addressing structure. In many cases, simple mobility of a router can cause cascading address changes across entire tree branches. As we will discuss further in the evaluation, the Device Discovery service in application layer would be quite limited in accommodating some of the network changes, resulting in various levels of performance penalty.

4.1 Mobile End Device

By design, if the end device is mobile while transmitting data, it should resume the transmission as soon as it acquires its new network address. If the data flow is two way, a route discovery and Device Discovery process will be triggered at the receiver. If the

end device is receiving data, the data flow would eventually recover if the application is successful in using the Device Discovery mechanism to rediscover the node's new network address. However, the Device Discovery mechanism would only work as intended under very limited mobility scenario (i.e., only one or two nodes moving within the network). As we will see in the subsequent section, when there are persistent or multiple occurrences of mobility, the longer routes and slower throughput (from multi-hopping) of tree routing tends to hinder the responsiveness of the described recovery scheme, causing a big degradation to performance.

4.2 Mobile Router

For ZigBee tree topologies, new network addresses are assigned in accordance to Eq. 2, to ensure the correct hierarchical tree structure. The stability of the addressing structure is important for the proper delivery of packets. Therefore, when ZigBee router acquires a new parent router and a new network address, it could potentially start a cascading network address change to all of its descendant nodes on impacted branches, which generally creates varying levels of inconsistency to the tree addressing scheme, thereby reducing the routing protocol's ability to function properly.

For the case that the mobile router moves out of the range of its original parent router and acquires a new network address, data reception will be halted completely, and cannot be recovered from any available ZigBee tree routing mechanisms. Sometimes, these simple movements would also jolt drastic structural change to the descendent nodes, influencing other existing flows. Even with the Device Discovery primitive in the application layer, the tree topology encounters great difficulties in recovering the path, since longer routes and slower throughput from tree routing tends to hinder the responsiveness of the described recovery scheme as mentioned earlier.

If the mobile router was sending data while it changes parent router, it would require its old descendants to change their network address. Depending on the number of impacted nodes, similar failure would be experienced as in the cases mentioned above. Except, the mobile router would probably be able to continue the data transmission once it acquires a new network address (in simple mobility scenarios). In any event, when there are persistent or multiple occurrences of mobility, the problem with tree routing would increase in complexity, and the tree routing will yield poor routing performances.

5 Evaluation

In this section, we present simulations results that illustrate the properties of ZigBee mesh and ZigBee tree routing schemes. We use the NS-2 simulator with Samsung's IEEE 802.15.4 extension [6], and contribute the implementation of the ZigBee tree routing and ZigBee mesh routing schemes according to the ZigBee standard version 1.0. The simulation is set to mimic the settings of a household/factory deployment. Nodes are initially aligned in an equally spaced grid before a selected percentage of nodes become mobile. Nodes move within the topology according to the random waypoint model [5], and all results are averaged across 10 independent trials of the same configuration. For all of our simulations, the network used in our simulation consists of

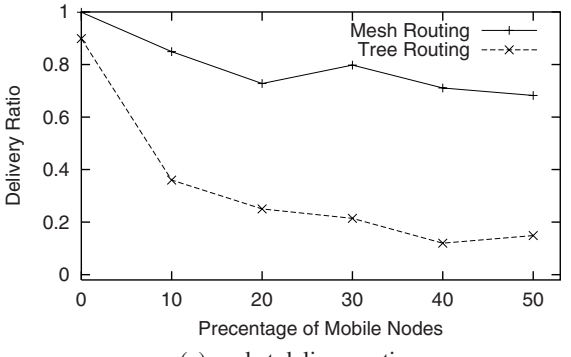
Table 1. General Simulation Parameters

Network Size	45m x 45m	Traffic Type, Packet Size	CBR, 127bytes
Number of Nodes	36 nodes	Mobility Model	Random Waypoint
Transmission Range	15 meters	$nwkMaxRouter(R_m)$	10
Network Setup Time	30 seconds	Number of Concurrent Data Flows	2
Simulation Duration	300 seconds	$nwkMaxDepth(L_m)$	5
Transmission Rate	10 packets/sec	$nwkMaxChildren(C_m)$	10

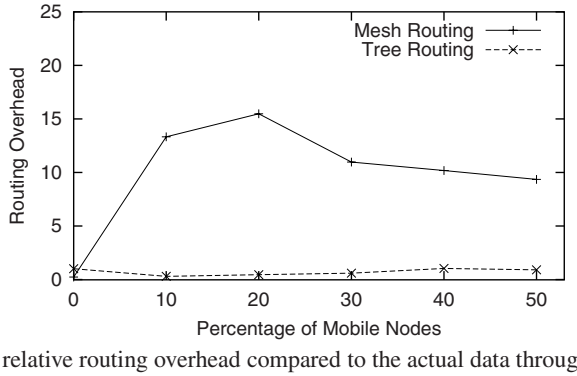
70% routers and 30% end devices, which are all randomly chosen. We use packet delivery ratio and relative routing overhead as our performance evaluation matrices. Packet delivery ratio is averaged over the number of flows in the network to reflect the mean per-flow delivery ratio. On the other hand, routing overhead is denoted by a normalized value of the total overhead of the network with respect to the traffic in the network. The parameters employed in the simulation are summarized in Table 1.

5.1 Scenarios with Varying Percentage of Mobile Nodes

This subsection studies the performance of the two ZigBee routing schemes when there are varying amount of mobile nodes in the network. Mobile nodes move at a speed of

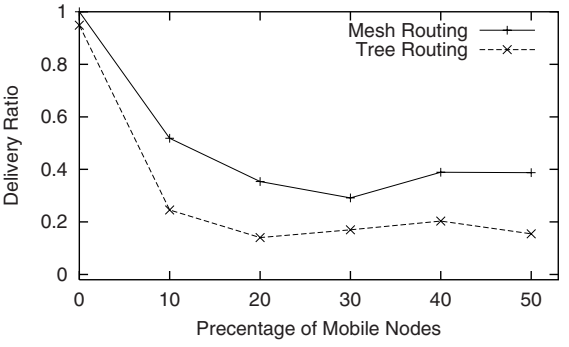


(a) packet delivery ratio

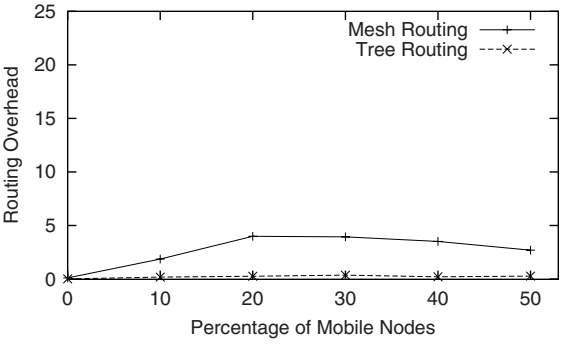


(b) relative routing overhead compared to the actual data throughput

Fig. 1. ZigBee router as mobile sender, data to stationary destination



(a) packet delivery ratio

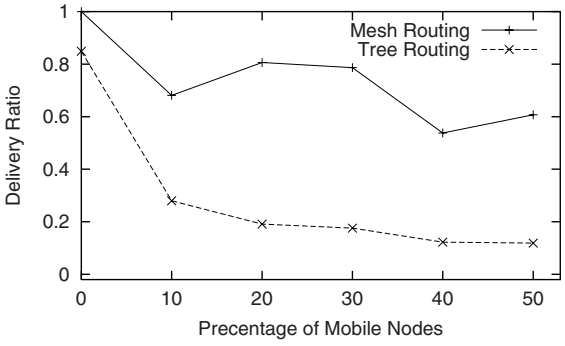


(b) relative routing overhead compared to the actual data throughput

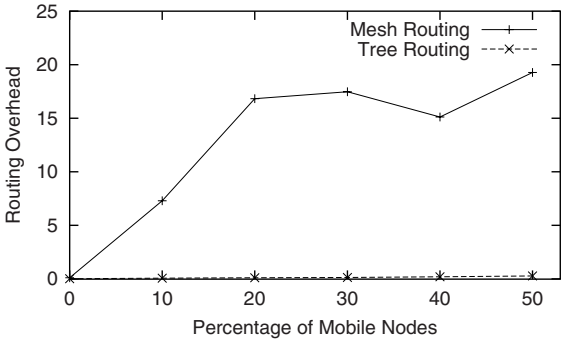
Fig. 2. ZigBee end device as mobile sender, data to stationary destination

1m/s randomly. Two general mobility cases were simulated. In the first scenario, the sender remains stationary while the receiver is mobile. In the second scenario, we keep the receiver stationary while setting senders to be mobile. We repeat the same simulations with two node types, i.e., ZigBee routers and end devices. Source and destination are randomly chosen, but all networking settings remain the same for all simulations. We vary the percentage of mobile nodes from 0 to 50% to observe the response from the two routing protocols to increasing percentages of mobile nodes in the network.

From the results depicted in Fig. 1-a and 2-a, it is clear that the device type plays a critical role in determining the delivery ratio for mobile senders. ZigBee routers can typically transmit out more data, while ZigBee devices can only send out half of the amount compare to the routers. Furthermore, the ZigBee end devices are more heavily influenced by the percentage of mobile nodes in the network compare to the ZigBee router. This is due to the fact the ZigBee end devices need to associate with a new parent when it moves, the extra association time actually degrades the packet delivery ratio. On the other hand, from Fig. 1-b and 2-b, we see that ZigBee routers actually incurs more routing overhead compare to the end devices. The additional routing overhead is from route repair messages that routers send/receive to repair the route.



(a) packet delivery ratio

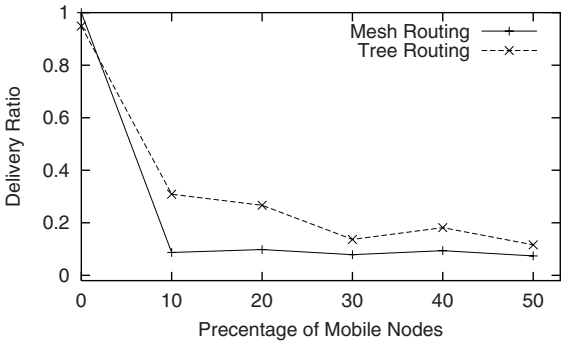


(b) relative routing overhead compared to the actual data throughput

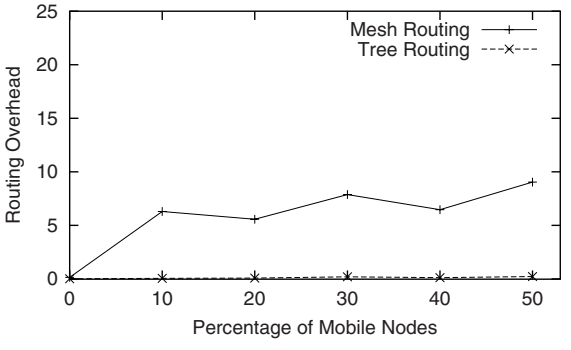
Fig. 3. ZigBee router as mobile receiver, data from stationary source

Additionally, in Fig. 1-a and 2-a, we see that mesh routing exhibits better packet delivery ratio compared to tree routing. The performance gap is especially evident when the mobile sender is a ZigBee Router. It is clear that the rigid routing scheme demanded by tree routing is less robust to increasing amounts of mobility in the network, as it lacks an effective route recovery method when a route fails. Thus, delivery ratio suffers. When the network is experiencing multiple instances of mobility, it is apparent that the application recovery mechanism has a minimum effect in repairing broken routes. As a results, tree routing performs quite poor when the network comprises of 20% or more mobile nodes. Yet, as seen in Fig. 1-b and 2-b, we see that tree routing incurs much smaller amount of routing overhead compare to mesh routing.

As the destination of data streams, all ZigBee receivers encounters some performance degradation (in terms of data delivery ratio) when it is mobile as illustrated in Fig. 3-a and 4-a. Device type actually differentiates the services received in the two routing schemes. Mobile receiver would clearly benefit if it is a ZigBee router in mesh routing. Nonetheless, device type would remain indifferent when tree routing is deployed, since movement in our tree topology would cause approximately the same amount of change regardless whether the node is a router or a end device.



(a) packet delivery ratio



(b) relative routing overhead compared to the actual data throughput

Fig. 4. ZigBee end device as mobile receiver, data from stationary source

In Fig. 3-a, mesh routing obviously performs much better than tree routing since the route repair mechanism in mesh routing can repair some of the mobility induced damages. Results also confirm the intuition that mesh routing consumes more overhead when there are more mobile nodes in the network. It is also clear that tree routing consistently consumes less overhead than mesh routing, regardless of the number of mobile nodes in the network.

The only scenario that tree routing outperforms mesh routing is when ZigBee end devices are receivers. In this scenario, ZigBee end devices suffer degraded performance in mesh routing because it constantly acquires new network addresses, and the Device Discovery service cannot recover the new network address in a timely manner. However, since the ZigBee end device would pick the lowest ID node as its initial parent, tree routing tends to pick a parent node that is further away. Thus, when there are more mobile nodes in the network, there is a good chance to get closer to its original parent node. This explains the superior performance of tree over mesh routing in Fig. 4-a.

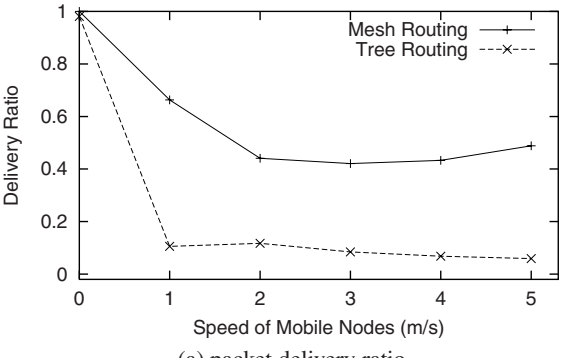
The results in this subsection suggests the suitability of mesh routing for ZigBee network anticipating many instances of mobile nodes. Additionally, it shows that ZigBee routers tend have better delivery ratio in most scenarios. Plus, it shows that tree routing

is the more effective scheme for static ZigBee networks with low data rate applications, due to its low overhead consumption.

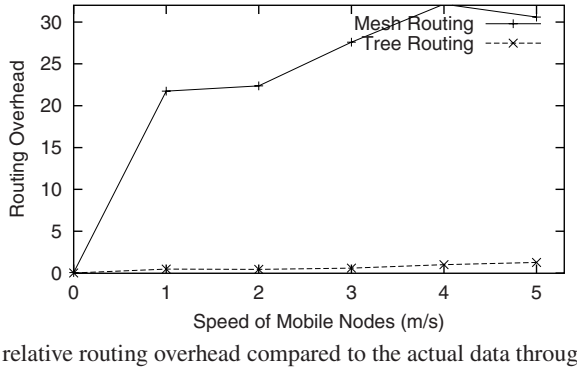
5.2 Scenarios with Mobile Nodes of Varying Speed

Following the same methodology in the previous subsection, this subsection studies the routing performance of the two ZigBee routing schemes when the mobile nodes in the network are moving at varying speeds. The ZigBee network in question consist of 70% routers and 30% end devices, and 20% of the nodes in the network are selected randomly as mobile nodes. Specifically, we evaluate for packet delivery ratio when the nodes are moving from 1m/s to 5m/s in 1m/s increments.

Fig. 5-a and 6-a clearly show that the device type plays a critical role in determining the delivery ratio in mesh routing. ZigBee routers can typically transmit out more data, while ZigBee devices can only send out half of the amount compare to the routers. On the other hand, from Fig. 5-b and 6-b, we see that ZigBee routers actually incurs more routing overhead compare to the end devices. The additional

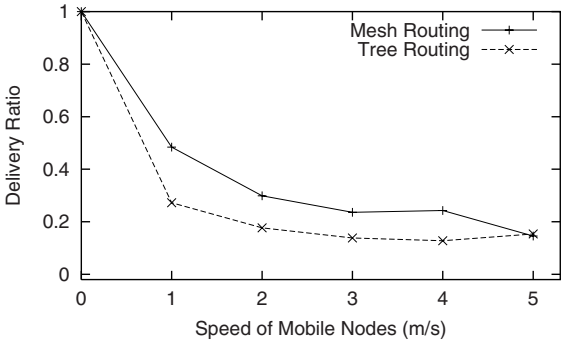


(a) packet delivery ratio

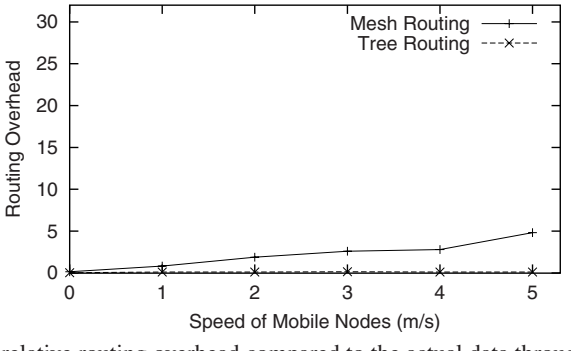


(b) relative routing overhead compared to the actual data throughput

Fig. 5. ZigBee router as mobile sender, data to stationary destination



(a) packet delivery ratio



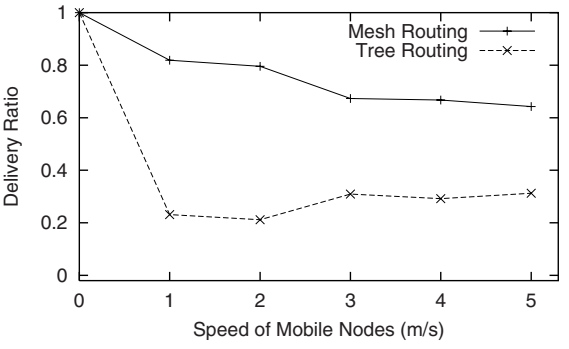
(b) relative routing overhead compared to the actual data throughput

Fig. 6. ZigBee end device as mobile sender, data to stationary destination

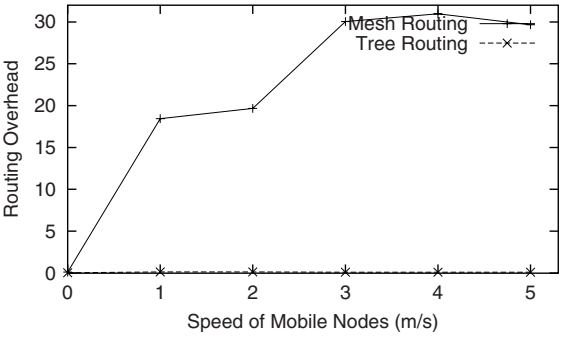
routing overhead is from the various route repair messages that routers send/receive to repair the route. We also see that as node speed increases, the delivery ratio decreases.

Additionally, in Fig. 5-a and 6-a, we see that mesh routing exhibits better packet delivery ratio than tree routing. Like in the previous subsection, the performance gap is especially evident when the mobile sender is a ZigBee Router. This is again due the ineffectiveness of tree routing's rigid routing scheme, and ZigBee router's ability to route for itself.The route repair mechanism in mesh routing makes them far more robust to mobility then their tree routing counterparts. When network speed increases, the application Device Discovery mechanism provided minimal help in re-establishing the route. As a results, we witness the same amount of performance degradation for the two tree routing scenarios. Yet, as seen in Fig. 5-b and 6-b, we see that tree routing again incurs much smaller amount of routing overhead compare to mesh routing.

In most scenarios, ZigBee receivers tend to encounters more severe performance degradation when it is traveling at higher speeds, as shown in Fig. 7-a and 8-a. As in the last subsection, device type actually differentiates the services received in the two



(a) packet delivery ratio



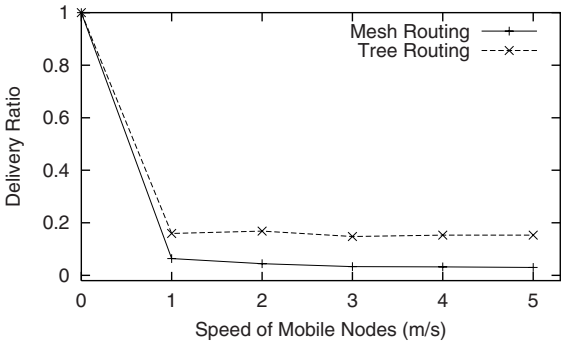
(b) relative routing overhead compared to the actual data throughput

Fig. 7. ZigBee router as mobile receiver, data from stationary source

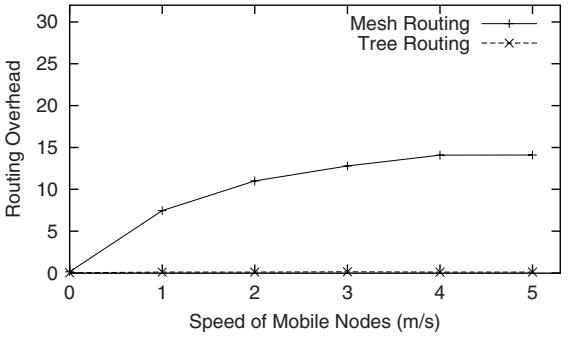
routing schemes. Mobile receiver would clearly benefit if it is a ZigBee router in mesh routing. Nonetheless, device type would remain relatively indifferent when tree routing is deployed, for the same reason pointed out earlier.

As depicted in Fig. 7-a, mesh routing with ZigBee routers exhibited more resiliency against high node speeds, even though it consumes more overhead than tree routing as illustrated in Fig. 7-b and 8-b. Like in the previous subsection, the only scenario that tree routing outperforms mesh routing is when ZigBee end devices are receivers. The reason is the same as in the previous subsection, and it also illustrates the inefficiency of the application Device Discovery mechanism in recovering the route.

The overall results in this subsection suggest the suitability of mesh routing for ZigBee networks anticipating high speed mobile nodes. It also shows that tree routing is more effective for static ZigBee network with low rate applications, due to its low overhead consumption (and saves more energy). However, mesh routing is clearly more robust to nodal mobility, which closes echoes the findings in previous sections.



(a) packet delivery ratio



(b) relative routing overhead compared to the actual data throughput

Fig. 8. ZigBee end device as mobile receiver, data from stationary source

6 Conclusion

In this study, we discussed ZigBee routing and its support for device mobility, and we analyzed the adequacy of current provisions in dealing with different mobility cases. Our evaluation results indicate that when network is static, both mesh and tree routing schemes work as intended; however, ZigBee end devices experiences detrimental packet losses in almost all mobility scenarios. This situation worsens under multiple instances of mobility, and when mobile nodes travel at higher speeds. Yet, ZigBee router typically suffers less packet losses under mobile scenarios. This behavior is closely related the fact that ZigBee router are routing capable, while the ZigBee end devices are not. We also realized that the current recovery mechanism is inadequate in accommodating multiple instance or rapid mobility. Additional design work is underway to resolve the various problems pointed out in this paper.

References

1. Ieee 802.15.4. http://www.ieee802.org/15/pub/TG4.html
2. Zigbee specification v1.0 (June 2005)

3. Jafari, R., Encarnacao, A., Zahoory, A., Dabiri, F., Noshadi, H., Sarrafzadeh, M.: Wireless sensor networks for health monitoring. In: ACM/IEEE ICMUS (2005)
4. Korhonen, I., Parkka, J., Gils, M.V.: Health monitoring in the home of the future. In: IEEE EMBM (2003)
5. PalChaudhuri, S., Boudec, J.-Y.L., Vojnovic, M.: Perfect simulations for random trip mobility models. In: 38th Annual Simulation Symposium (2005)
6. Zheng, J., Lee, M.J.: A Comprehensive Preformance Study of IEEE 802.15.4. In: Sensor Network Operations, ch. 4, pp. 218–237. IEEE Press, Los Alamitos (2006)

On Using Probabilistic Forwarding to Improve HEC-Based Data Forwarding in Opportunistic Networks*

Ling-Jyh Chen[1], Cheng-Long Tseng[2], and Cheng-Fu Chou[2]

[1] Institute of Information Science, Academia Sinica
[2] Department of Computer Science and Information Engineering, National Taiwan University

Abstract. In this paper, we propose the HEC-PF scheme, an enhancement of our previous H-EC scheme for effective data forwarding in opportunistic networks. The enhanced scheme modifies the aggressive forwarding phase of the H-EC scheme by implementing a new *Probabilistic Forwarding* feature, which decides whether to forward a message to a newly encountered node based on the *delivery probability*. Using simulations as well as realistic network traces, we evaluate the performance of the proposed scheme in terms of delivery latency and completion ratio. The results show that the HEC-PF scheme outperforms the EC and H-EC schemes in all test cases, and the performance gain is even more substantial when network connectivity is extremely poor. By varying the parameters of the HEC-PF scheme, we show that its completion ratio improves as the *maximum forwarding distance* or the *hop distance considered when calculating the delivery probability* increases. The effectiveness of the HEC-PF scheme makes it an ideal solution that goes a long way toward ensuring effective data delivery in opportunistic networks.

Keywords: Opportunistic Networks; Probabilistic Forwarding; Erasure Coding.

1 Introduction

An opportunistic network is a type of challenged network that has the following characteristics: (1) network contacts (i.e., communication opportunities) are intermittent; (2) an end-to-end path between the source and the destination has never existed; (3) disconnection and reconnection are common occurrences; and (4) the link performance is highly variable or extreme. Because of various disruptions and long delays, traditional MANET and Internet routing techniques can not be applied directly in opportunistic networks. Hence, with the development of numerous opportunistic networking applications, such as wireless sensor networks (WSN), underwater sensor networks (UWSN), pocket switched networks (PSN), people networks, and transportation networks, it is necessary to develop an effective data forwarding scheme that can better accommodate the various characteristics of opportunistic networks.

* This work was funded by the National Science Council under grant numbers NSC 95-2221-E-001-025.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 101–112, 2007.

Several data forwarding schemes have been proposed for opportunistic networks [5] [9] [12] [15] [16] [17] [19]. Such schemes can be divided into two main categories, *replication-based* and *coding-based*, according to their basic technical strategies. Replication-based routing schemes are the most popular design choice for existing opportunistic routing schemes. Basically, such schemes input multiple identical copies of data into the network, and rely on node mobility to forward the data to the destination [16]. A message is regarded as successfully delivered when at least one of the multiple copies is received by the destination. Intuitively, if the number of replicas in the network is sufficiently large, replication-based schemes should achieve the best delay performance (i.e., the shortest delivery latency) in opportunistic networks. However, the main drawback of this type of scheme is the tremendous traffic overhead associated with flooding a network with data replicas. As a result, when network resources are limited (e.g., the buffer space and network bandwidth), replication-based schemes will probably degrade performance reliability unless additional overhead reduction strategies are in place to alleviate the traffic overhead (e.g., [5] [9] [12] [15]).

For coding-based routing schemes, a message (or group of messages) is transformed into another format prior to transmission [17] [19]. The design principle of these schemes is to embed additional information (e.g., redundancy [17] or a decoding algorithm [19]) in the coded blocks such that the original message can be successfully reconstructed with only a certain number of the coded blocks. More precisely, unlike replication-based schemes, which rely on successful delivery of each individual data block, coding-based schemes consider a block successfully delivered when enough blocks are received to reconstruct the original data. As a result, coding-based schemes are usually more robust than replication-based schemes when a network's connectivity is extremely poor; however, they are less efficient when the network is fairly connected due to additional information embedded in the coded blocks.

In [7], Chen et al. proposed a hybrid scheme, called H-EC, which combines the strengths of replication-based schemes and coding-based schemes by integrating their respective *aggressive forwarding* technique and erasure coding technique. Hence, the H-EC scheme not only remains robust when the network connectivity is extremely poor, but also performs efficiently when the network is fairly connected. Even so, the performance of the scheme depends to a large extent on the message scheduling algorithm used in the aggressive forwarding phase. This aspect has not been discussed extensively and algorithms have only been implemented in a First-Come-First-Served (FCFS) fashion [3]. Thus, in this paper, we investigate effective message scheduling algorithms that consider both the frequency and volume of contacts in a network's history, and thereby substantially improve the data forwarding performance of the H-EC scheme in opportunistic networks.

The remainder of the paper is organized as follows. In Section 2, we review related works on opportunistic routing, and provide an overview of the H-EC scheme. In Section 3, we propose the HEC-PF scheme, which employs the *Probabilistic Forwarding* feature in the aggressive forwarding phase of the H-EC scheme. Section 4 presents a comprehensive set of simulation results for various opportunistic network scenarios; the results are also analyzed and explained in detail. We then present our conclusions in Section 5.

2 Related Work and Overview of the H-EC Scheme

2.1 Related Work

Routing in an opportunistic network is challenging and completely different to conventional network routing methods. In opportunistic networks, an ideal routing scheme has to provide reliable data delivery, even when the network's connectivity is intermittent or when an end-to-end path is temporally unavailable. Moreover, since "contacts" in an opportunistic network may appear arbitrarily without prior information, scheduled optimal routing methods (e.g., linear programming-based routing [11]) and mobile relay approaches (e.g., Message Ferrying [20]) cannot be applied.

Currently, replication is the most popular design choice for opportunistic routing schemes. For instance, the *Epidemic Routing* scheme [16] sends identical copies of a message simultaneously over multiple paths to mitigate the effects of a single path failure, thereby increasing the possibility of successful message delivery. However, flooding a network with duplicate data tends to be very costly in terms of traffic overhead and energy consumption. To address the excess traffic overhead incurred by flooding replicate data, the *Controlled Flooding* proposed in [9] reduces the flooding cost while maintaining reliable message delivery.

Node mobility also impacts on the effectiveness of opportunistic routing schemes. When network mobility differs from the well-known random way-point mobility model, the overhead of epidemic- and/or flooding-based routing schemes can be further reduced by considering node mobility. For instance, the *PRoPHET* scheme [14] calculates the *delivery predictability* from a node to a particular destination node based on the observed contact history, and forwards a message to its neighboring node if and only if that neighbor node has a higher delivery predictability value. Leguay *et al.* [12] revised this scheme by taking a node's *mobility pattern* into account, i.e., a message is forwarded to a neighbor node if and only if the neighbor node has a mobility pattern similar to that of the destination node. The approach described in [12] shows that the revised *mobility pattern*-based scheme is more effective than previous methods.

Another class of opportunistic network routing schemes is based on encoding techniques, which transform a message into another format prior to transmission. For example, the integration of *network coding* and epidemic routing techniques has been proposed to reduce the number of transmissions required in a network [19], while [17] proposes combining *erasure coding* and the simple replication-based routing method to improve the data delivery in the *worst delay performance* cases in opportunistic networks. Following [17], an Estimation-based Erasure-Coding (EBEC) routing scheme has been proposed to adapt the delivery of erasure coded blocks using the Average Contact Frequency (ACF) estimate [13]. Moreover, [7] proposes a hybrid scheme that combines the strength of erasure coding and the advantages of *Aggressive Forwarding*, so that it is robust in *worst delay performance cases*, and performs efficiently in *very small delay performance cases*.

2.2 H-EC: An Overview

We now present an overview of erasure coding schemes and the H-EC scheme [7], which is a hybrid data forwarding scheme based on erasure coding.

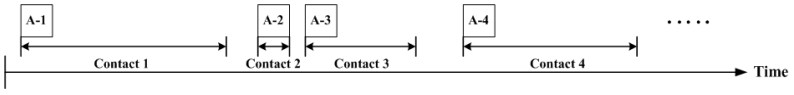


Fig. 1. The erasure coding-based data forwarding algorithm (EC). In this figure, one erasure coded block (A) is split equally among four relays ($n = 4$).

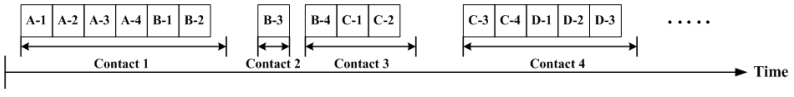


Fig. 2. The A-EC scheme, i.e., EC with aggressive forwarding. In this figure, four erasure coded blocks (A,B,C,D) are transmitted, with $n = 4$.

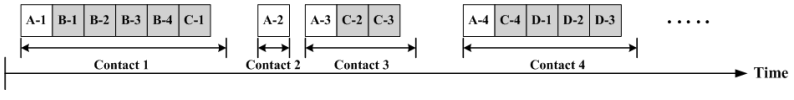


Fig. 3. The H-EC scheme. Under the scheme, two copies of four erasure coded blocks (A,B,C,D) are transmitted: the first copy of EC blocks (the white blocks) is sent using the EC algorithm, and the second copy (the gray blocks) is sent using the A-EC algorithm in the residual contact time. Each coded block is split into 4 equal-sized sub-blocks ($n = 4$).

By adding redundancy, erasure coding improves the fault-tolerance without the overhead of strict replication of the original data [18]. In a generic erasure coding scheme, given a message of size M bytes, a replication factor of erasure coding r, and a coded message fragmented into several blocks of equal-sized bytes b, the number of coded blocks can be derived by $N = \frac{M \times r}{b}$. Moreover, this message can be successfully reconstructed as long as $\frac{1}{r}$ of the coded blocks is received.

An erasure code-based (EC) forwarding algorithm is proposed in [17]. In this scheme, (see Fig. 1), the erasure coded blocks are split equally among n relays[1], which are only allowed to send messages to the destination directly (this is the well-known "two-hop" scenario used in [8]). Each relay forwards the same number of coded blocks, and there are no duplicates in a relay.

As reported in [17], the EC scheme achieves the most effective *worst-case delay performance* with a fixed amount of overhead. However, the drawback of the scheme is that it can not provide a good *very small delay performance* compared to other popular replication-based approaches. The reason for this inefficiency is because, if most network contacts are much longer than the required time, the EC scheme tends to *waste* the residual contact period; hence, it is ineffective, as illustrated in Fig. 1.

To resolve the above problem, [7] proposed an enhanced scheme called A-EC, i.e., EC with an *aggressive forwarding* feature, as shown in Fig. 2. Under this scheme, the source sends as many coded blocks as possible during each contact period. It has been shown that the A-EC scheme is better able to utilize network contacts; thus, it can

[1] For simplicity, following [17], we assume N=n for all cases.

be expected to outperform the EC scheme for *very small delay performance* cases. However, for *worst delay performance* cases, A-EC yields a poor delivery ratio and/or a very large delivery delay when *black-holes*[2] are present in the network [7].

Taking advantage of the strengths of the EC and A-EC schemes to achieve better message delivery performance in both *worst delay performance* and *very small delay performance* cases, Chen et al. [7] proposed a hybrid scheme, called H-EC, which is illustrated in Fig. 3. In the H-EC scheme, two copies of EC blocks (constructed by the erasure coding and replication techniques described earlier) are transmitted by the sender. The first copy of EC blocks is sent in a similar way to the method used to send blocks in the original EC scheme (the white blocks in Fig. 3), while the second copy is sent using aggressive forwarding during the residual contact time after sending the first EC block (the gray blocks in Fig. 3). For general opportunistic network scenarios (i.e., without black-hole nodes), the H-EC scheme utilizes each contact opportunity better because of the aggressive forwarding feature; however, if black-hole nodes are present in the network, the scheme's performance is similar to that of the EC scheme, which achieves better forwarding in *worst delay performance* cases.

The performance of the H-EC scheme depends to a large extent on the message scheduling algorithm used in the aggressive forwarding phase, which is not discussed in [7]. However, in this paper, we assess the impact of probabilistic message scheduling algorithms on the performance of H-EC routing. To this end, we propose an extension of the H-EC scheme, called HEC-PF, in the following section.

3 HEC-PF: H-EC with Probabilistic Forwarding

In this section, we propose a message scheduling algorithm, called *Probabilistic Forwarding*, for the aggressive forwarding phase of the H-EC scheme. The resulting scheme is called HEC-PF. Unlike the H-EC scheme [7], the HEC-PF scheme does *NOT* enter the aggressive forwarding phase unless a newly encountered node has a higher likelihood of successfully forwarding the message to the destination node than the current node. We describe the HEC-PF scheme in the following sub-sections.

3.1 Delivery Probability

The key issue for the HEC-PF scheme is how to estimate the likelihood of successfully transmitting a message from a given node to the destination node. Similar to the PRoPHET scheme [14], the HEC-PF scheme estimates the *delivery probability* based on the observed contact history. However, unlike PRoPHET, the HEC-PF scheme considers the *contact frequency* in the history as well as the *contact volume*, which represents the proportion of time that the two nodes are in contact in the last T time units. More specifically, if there are K nodes in the network, we denote the i-th node as X_i, the j-th node as X_j, the aggregate contact volume between the node pair X_i and X_j in the last T time units as t_{X_i,X_j}, and the *delivery probability* for the node pair X_i and X_j with a distance of at most k-hops as $P^k_{X_i,X_j}$. The one-hop delivery probability from the

[2] A node in a network is called a *black-hole* if it is either unreliable (e.g., it has very limited battery power and/or buffer size) or it hardly moves towards the destination [7].

source node (X_S) to the destination node (X_D) is given by the ratio of the aggregate contact volume over the overall contact volume[3], as shown in Eq. 1.

$$P^1_{X_S,X_D} = \frac{t_{X_S,X_D}}{\sum_{i=1}^{K} t_{X_S,X_i}} \tag{1}$$

In addition, the two-hop delivery probability, $P^2_{X_S,X_D}$, can be derived by Eq. 2. The equation is comprised of three components: the scaling constant, $\omega_2 \in [0...1]$, which decides the impact of two-hop message transfer on the overall delivery probability; the likelihood value, $1 - P^1_{X_S,X_D}$, which is the probability that a message can *not* be transmitted directly from node X_S to node X_D (i.e., it is impossible to complete the message delivery in one hop); and the sum of the two-hop *transitive* delivery probability based on the *transitive* property, i.e., if node X_S frequently encounters node X_i, and node X_i frequently encounters node X_D, then X_i is a good candidate relay node for forwarding messages from node X_S to node X_D.

$$P^2_{X_S,X_D} = \omega_2(1 - P^1_{X_S,X_D}) \sum_{\substack{1\le i\le K \\ i\neq S, i\neq D}} (P^1_{X_S,X_i} P^1_{X_i,X_D}) \tag{2}$$

Similarly, the three-hop delivery probability can be estimated by Eq. 3 and the k-hop delivery probability can be derived by Eq. 4. Finally, the delivery probability of transferring a message from node X_S to node X_D is given by summing the delivery probabilities of all k cases, as shown by Eq. 5.

$$P^3_{X_S,X_D} = \omega_3(1 - P^1_{X_S,X_D} - P^2_{X_S,X_D}) \times \sum_{\substack{1\le i,j\le K \\ i\neq S, i\neq D \\ j\neq S, j\neq D}} (P^1_{X_S,X_1} P^1_{X_1,X_2} P^1_{X_2,X_D}) \tag{3}$$

$$P^k_{X_S,X_D} = \omega_k \left(1 - \sum_{i=1}^{k-1} P^i_{X_S,X_D}\right) \times \sum_{\substack{1\le a_i\le K, \\ a_i\neq S, a_i\neq D, \\ a_i\neq a_j; \forall 1\le i,j\le k}} \left(P^1_{X_S,X_{a_1}} \left(\prod_{i=1}^{k-2} P^1_{X_{a_i},X_{a_{i+1}}}\right) P^1_{X_{a_{k-1}},X_D}\right) \tag{4}$$

$$P_{X_S,X_D} = \sum_{i=1}^{k} P^i_{X_S,X_D} \tag{5}$$

Note that deciding an adequate value for k involves a tradeoff. On the one hand, the larger the value of k, the more accurately we can approximate the delivery probability; on the other hand, using a large k is very likely to incur an enormous storage and computation overhead. An ideal solution would be able to adapt its k settings to the properties of the network. For simplicity, we set k to a constant value in the simulations, and defer a detailed discussion and evaluation of this issue to a future work.

[3] If $i == j$, $t_{X_i,X_j} = 0$.

3.2 Probabilistic Forwarding

We now describe the HEC-PF scheme in detail. As mentioned earlier, the scheme incorporates a new *Probabilistic Forwarding* feature that decides whether to forward a message to a newly encountered node based on the *delivery probability* estimate, rather than on a first-come-first-served basis. The HEC-PF scheme is better able to deliver a message successfully because it tends to utilize relay nodes that have higher delivery probabilities.

More precisely, suppose that node X_i holds a message from the source node X_S for transmission to the destination node X_D, and the message has not been relayed more than H times so far. There are two cases where X_i could accidentally encounter another node X_j (assuming $X_j \neq X_D$) if there is time remaining after X_i sends out one block of the first copy of the EC blocks. First, if X_i is the source node, X_i must make a decision about whether to enter the *aggressive forwarding* phase by comparing the two delivery probabilities, P_{X_i,X_D} and P_{X_j,X_D}. If $P_{X_i,X_D} > P_{X_j,X_D}$, X_i will enter the aggressive forwarding phase in the same way as the original H-EC scheme; otherwise, it will follow the EC scheme and not enter the aggressive forwarding phase.

In the second case (i.e., X_i is not the source node), X_i first checks whether the next-to-be-sent block was sent by the source node during the *aggressive forwarding* phase (i.e., whether it belongs to the second copy of the EC blocks). If it was, X_i forwards the block to X_j as long as X_j has a higher delivery probability than X_i of successfully forwarding the block to X_D (i.e., $P_{X_j,X_D} > P_{X_i,X_D}$); otherwise, the block belongs to the first-copy of the EC blocks, and X_i forwards it to X_j automatically, i.e., without checking the delivery probability.

4 Evaluation

We use a set of simulations to evaluate the delay and completion ratio performance of the HEC-PF scheme in opportunistic networks. We implement the HEC-PF scheme by extending the H-EC codes [3] and run simulations in DTNSIM [2], a Java-based DTN simulator. Similar to the scenarios described in [17], messages are generated at a Constant Bit Rate (CBR) of 12 messages per day for 160 days (i.e., $M = 12 \times 160 = 1,920$), and the size of each message is 1,000 bytes. For all EC based schemes, the code block size b is set to 125 bytes, and the other two parameters, r and n, are set to 2 and 16 respectively. In addition, under the HEC-PF scheme, the sliding time window T (used to estimate the delivery probability) is set to 1,000 seconds; and the scaling constant ω_i is set to 0.25 for $i = 2...5$. The simulation results represent the average performance of 200 simulation runs. In each run the source and destination pair was randomly selected from all the participating nodes.

We evaluate three network scenarios. The first is generated according to the power-law distribution by setting both the inter-contact time and the contact duration of the networks as power-law distributed random variables with coefficient equal to 0.6 (following [10]). There are 34 participating nodes. The other two scenarios are based on realistic campus wireless network traces, namely, the iMote [1] and UCSD [4] traces, which are publicly available for research purposes. Table 1 outlines the basic properties of the three network scenarios.

Table 1. The properties of the three network scenarios

Trace Name	Power-Law	iMote	UCSD
Device	N/A	iMote	PDA
Network Type	N/A	Bluetooth	WiFi
Duration (days)	16	3	77
Devices participating	34	274	273
Number of contacts	25,959	28,217	195,364
Avg # Contacts/pair/day	2.89205	0.12574	0.06834

The UCSD trace is client-based and records the availability of WiFi-based access points (APs) for each participating portable device (e.g., PDAs and laptops) on the UCSD campus. Similar to [6] [7] [10], we assume that two participating devices in ad hoc mode encounter a communication opportunity (i.e., a network contact) if and only if they are both associated with the same AP at the same time.

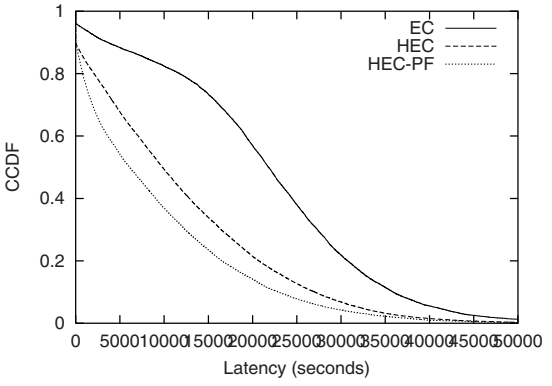
The iMote trace is a human mobility trace collected at the 2005 IEEE Infocom conference. It was aggregated from 41 Bluetooth-based iMote devices, which were distributed to the student attendees for the duration of the 3-day conference. Each iMote device was pre-configured to periodically broadcast query packets to find other Bluetooth devices within range, and record the devices that responded to the queries. In addition to the distributed iMote devices, another 233 devices were recorded in the trace. They may have been other Bluetooth-enabled devices used during the conference. For simplicity, we assume there is a network contact between two Bluetooth devices if there exists a query-and-response interaction between them.

4.1 Evaluation I: Two-Hop Scenario

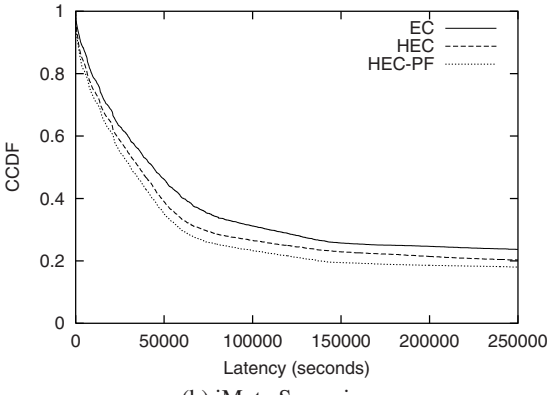
In the first set of simulations, we evaluate the delay performance of the HEC-PF scheme for message delivery in opportunistic networks. Three network scenarios are examined via simulations, with H set to 2 (i.e., the conventional two-hop scenario used in [7] [17]). The k parameter of the HEC-PF scheme is set to 2 (i.e., we consider the transitive property of message delivery with a distance of up to two hops). Fig. 4 depicts the average data latency distribution results in Complementary CDF (CCDF) curves.

From Fig. 4, we observe that the HEC-PF scheme consistently outperforms the H-EC and EC schemes in all test scenarios. The reason is that HEC-PF employs the *Probabilistic Forwarding* feature, which decides whether to forward a message to a newly encountered node based on the *delivery probability* estimates, rather than on the first-come-first-served (FCFS) basis used in the original H-EC scheme. As a result, HEC-PF is better able to make use of nodes that are more likely to encounter the destination node, and thereby achieve better message delivery performance.

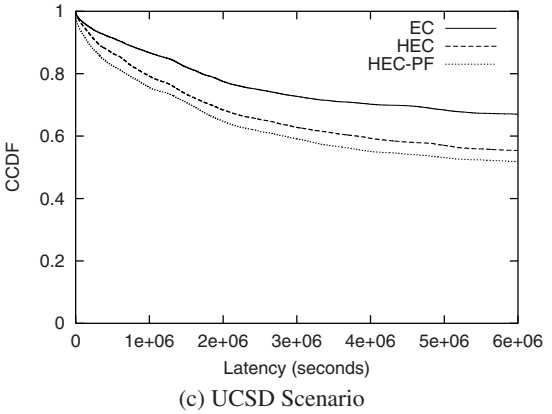
We also observe that the *completion ratio*, i.e., the percentage of messages successfully transmitted before the end of the simulation, degrades as the network connectivity (i.e., the average number of network contacts per node pair, per day) decreases. For example, although the HEC-PF scheme achieves approximately 99% completion ratio in the Power-Law scenario, it can only achieve about 80% in the iMote scenario and



(a) Power-Law Scenario



(b) iMote Scenario



(c) UCSD Scenario

Fig. 4. Distribution (CCDF) of average latency performance of the EC, H-EC, and HEC-PF schemes ($N = 16$, $r = 2$, $k = 2$, and $H = 2$)

50% in the UCSD scenario. The reason is that the two-hop forwarding strategy is not sufficient to deliver all messages in the limited simulation time because the network's connectivity is poor. Another observation is that the performance gain of the HEC-PF scheme (compared to the H-EC scheme) is not significant. This implies that the calculation of the delivery probability (with $k = 2$ in this case) cannot provide sufficient information to make a decision on whether to forward a message in the aggressive forwarding phase of the HEC-PF scheme. The findings motivate us to investigate the impact of the HEC-PF parameters on the performance of message delivery in opportunistic networks. We present the evaluation in the next subsection.

4.2 Evaluation II: Variable H Scenarios

In the second set of evaluations, we evaluate the performance of the HEC-PF scheme in the UCSD scenario with various maximum forwarding distance settings ($H = 2, 3, 4, 5$). Recall that the connectivity of this scheme is deemed to be poor. Similar to the previous evaluation, we set the k parameter of the HEC-PF scheme to 2. The average data latency distribution results are shown as CCDF curves in Fig. 5.

The results in Fig. 5 show that the CCDF curve falls as the value of H increases. More specifically, at the end of the simulation, the HEC-PF scheme improves the completion ratio from 48% to 62% as H is increased from 2 to 5. The reason is that the larger the setting of H, the greater the likelihood that a message will be delivered to the destination node eventually. Of course, when a large H is employed in the HEC-PF scheme, an extensive amount of one-hop data forwarding is required; thus, more energy will be consumed in the network (as shown in Table 2, the transmission overhead of the HEC-PF scheme increases substantially as the value of H increases). Again, an ideal solution for HEC-PF should adapt its H value to the properties of the network in order to compensate for the above tradeoff. We defer a detailed discussion and evaluation of this issue to a future work.

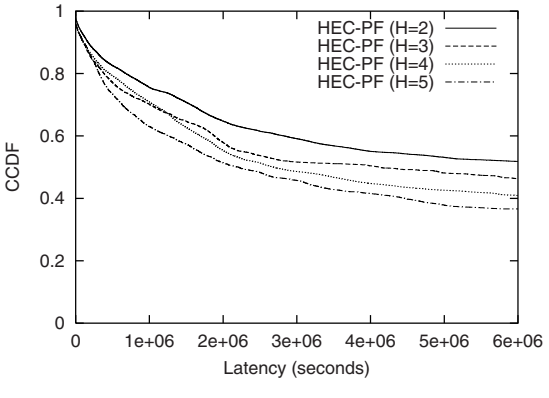


Fig. 5. Distribution (CCDF) of average latency performance of the HEC-PF scheme in the UCSD scenario with various H settings ($N = 16$, $r = 2$, and $k = 2$)

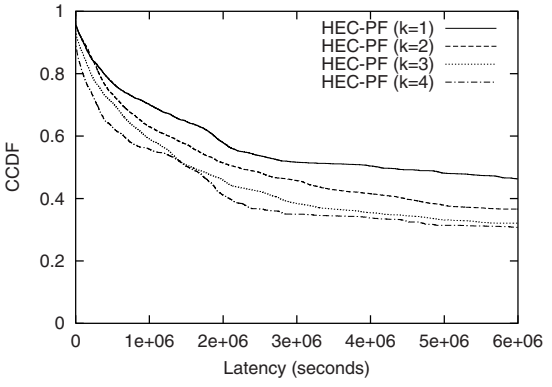


Fig. 6. Distribution (CCDF) of average latency performance of the HEC-PF scheme in the UCSD scenario with various k settings ($N = 16$, $r = 2$, and $H = 5$)

Table 2. Transmission overhead of HEC-PF with various H values. (Unit: MBytes)

Scenario	H=2	H=3	H=4	H=5
PowerLaw	7,527	9,747	13,088	15,455
iMote	2,324	3,200	4,020	4,818
UCSD	5,048	10,327	13,857	17,309

Table 3. Transmission overhead of HEC-PF with various k values. (Unit: MBytes)

Scenario	k=1	k=2	k=3	k=4
PowerLaw	15,062	15,455	15,899	16,234
iMote	4,523	4,818	4,897	4,950
UCSD	15,668	17,309	21,197	22,018

4.3 Evaluation III: Variable *k* Scenarios

Here, we evaluate the performance of the HEC-PF scheme in the poorly connected network scenario (i.e., the UCSD scenario) with various k values ($k = 1...5$). The configurations of the evaluation are the same as previously, except that the maximum forwarding distance H is set to 5. Fig. 6 depicts the average data latency distribution results in CCDF curves.

From Fig. 6, it is evident that the CCDF curve falls as k increases, which means that, given the same H setting, the completion ratio of message delivery increases as the hop distance (used to estimate the delivery probability) increases; however, the transmission overhead of the HEC-PF scheme only increases moderately as the value of k increases, as shown in Table 3. The figure also shows that the performance gain of the completion ratio decreases as k increases, which indicates that the completion ratio tends to converge. It is also worth noting that k must be configured less than H when estimating the delivery probability, since it is impossible to deliver a message successfully with a hop distance larger than the maximum forwarding distance.

5 Conclusion

We have proposed a scheme called HEC-PF that extends the basic H-EC scheme for data forwarding in opportunistic networks. The HEC-PF scheme incorporates a novel feature, called *Probabilistic Forwarding* feature, which decides whether to forward a message to a newly encountered node based on the *delivery probability* estimate in the aggressive forwarding phase. As a result, the scheme can find relays that are more likely

to transmit a message to the destination node based on the historical record of network contacts. Using simulations as well as realistic network traces, we evaluated the performance of the proposed scheme in terms of its delivery latency and completion ratio. The results show that it outperforms the EC and H-EC schemes in all test cases, and the performance gain is even more significant when the network connectivity is extremely poor. In addition, by varying the values of the parameters of the proposed scheme, we have shown that its completion ratio improves as the maximum forwarding distance or the considered hop distance of the delivery probability increases. Work on developing mechanisms to determine the HEC-PF parameters that can adapt to the properties of a network is still ongoing. We will report on the results in the near future.

References

1. Crawdad project. http://crawdad.cs.dartmouth.edu/
2. Delay tolerant network simulator. http://www.dtnrg.org/code/dtnsim.tgz
3. H-ec module for dtnsim simulator. http://nrl.iis.sinica.edu.tw/DTN/download/
4. Ucsd wireless topology discovery project. http://sysnet.ucsd.edu/wtd/
5. Burgess, J., Gallagher, B., Jensen, D., Levine, B.N.: Maxprop: Routing for vehicle-based disruption-tolerant networking. In: IEEE Infocom (2006)
6. Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., Scott, J.: Impact of human mobility on the design of opportunistic forwarding algorithms. In: IEEE Infocom (2006)
7. Chen, L.-J., Yu, C.-H., Sun, T., Chen, Y.-C., Chu, H.h.: A hybrid routing approach for opportunistic networks. In: ACM CHANTS (2006)
8. Grossglauser, M., Tse, D.: Mobility increases the capacity of ad-hoc wireless networks. In: IEEE Infocom (2001)
9. Harras, K.A., Almeroth, K.C., Belding-Royer, E.M.: Delay tolerant mobile networks (dtmns): Controlled flooding in sparse mobile networks. In: IFIP Networking (2005)
10. Hui, P., Chaintreau, A., Scott, J., Gass, R., Crowcroft, J., Diot, C.: Pocket switched networks and human mobility in conference environments. In: ACM WDTN (2005)
11. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: ACM SIGCOMM (2004)
12. Leguay, J., Friedman, T., Conan, V.: Dtn routing in a mobility pattern space. In: ACM WDTN (2005)
13. Liao, Y., Tan, K., Zhang, Z., Gao, L.: Estimation based erasure-coding routing in delay tolerant networks. In: IWCMC (2006)
14. Lindgren, A., Doria, A.: Probabilistic routing protocol for intermittently connected networks. Technical report, draft-lindgren-dtnrg-prophet-01.txt, IETF Internet draft (July 2005)
15. Lindgren, A., Doria, A., Schelen, O.: Probabilistic routing in intermittently connected networks. ACM Mobile Computing and Communications Review 7(3), 19–20 (2003)
16. Vahdat, A., Becker, D.: Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University (2000)
17. Wang, Y., Jain, S., Martonosi, M., Fall, K.: Erasure coding based routing for opportunistic networks. In: ACM WDTN (2005)
18. Weatherspoon, H., Kubiatowicz, J.D.: Erasure coding vs. replication: A quantitative comparison. In: IEEE IPTPS (March 2002)
19. Widmer, J., Boudec, J.-Y.L.: Network coding for efficient communication in extreme networks. In: ACM WDTN (2005)
20. Zhao, W., Ammar, M., Zegura, E.: A message ferrying approach for data delivery in sparse mobile ad hoc networks. In: ACM MobiHoc (2004)

Employment of Wireless Sensor Networks for Full-Scale Ship Application

Bu-Geun Paik[1], Seong-Rak Cho[1], Beom-Jin Park[1], Dongkon Lee[1], Jong-Hwui Yun[2], and Byung-Dueg Bae[2]

[1] Maritime & Ocean Engineering Research Institute, KORDI, 104 Shinseong St., Yuseong-gu, Daejeon, 305-343, Korea
{ppaik,scho,baracude,dklee}@ moeri.re.kr
[2] Korea Maritime University, 1 Dongsam-dong, Youngdo-ku, Busan, 606-791, Korea
{jhyun,captcos}@ hhu.ac.kr

Abstract. In this study, basic experiments regarding the wireless sensor network were conducted on a 3,000-ton-class training ship as the first step in applying the ubiquitous technology to a real ship. Various application fields of the technology in terms of the provision of safety and convenience on a ship would be extracted through these experiments. To efficiently adopt the ubiquitous technology for ship application, it is necessary to identify the state-of-the-art ubiquitous technology and to prepare countermeasures against the harsh environment of a ship. Especially, the characteristics of the wireless sensor network were investigated at the test bed ashore as well as on a real ship to figure out the complementary items before full-scale ship application.

Keywords: ubiquitous, ship, WSN (wireless sensor network), Zigbee, monitoring.

1 Introduction

During ship operation on the seas, it has different requirements according to its type and purpose. A commercial ship transports various kinds of cargo, such as containers, logs, ore, crude oil, and LNG (liquefied natural gas), to the destination port within a given time. Comfort and convenience are expected of a passenger ship. Military ships should have mobility and the capabilities that are necessary for it to be able to carry out its mission. Since a ship stays on the seas most of the time during its operation, greater convenience must be provided to its crew and passengers so that the efficiency and safety of its operation could be ensured.

A number of ship management technologies aimed at increasing the efficiency and safety of a ship have been developed and proposed by many researchers. The EU (European Community) has developed intelligent hull-monitoring systems to reduce the risk of structural failure, spills into the sea and damage to cargo, and to improve passenger safety and comfort [1]. Nguyen and Nelson [2] discussed an approach to integrating data collection and analysis of ship machinery for the assessment of the conditions and operational performance of the ship's equipment. Nielsen et al. [3]

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 113–122, 2007.

introduced the concept of the onboard management of wave-induced structural loads and ship motions through the measurement of relative wave motions, bending moment, and so on. Recently, Cho et al. [4] reported the ubiquitous sensor network technology for ship application and related basic experiment results.

The services for the crew and passengers of ships lag behind those for people ashore. On land, there are many opportunities to enjoy the ubiquitous technology and to have a convenient and safe life on account of it, such as through home automation or home network systems [5], tracking systems [6], security or safety systems, and health care systems [7]. For the crews and passengers of ships on the seas, however, it is not easy to acquire the benefits of ubiquitous technology because of several restrictions.

Most ship owners or shipping companies are not willing as of yet to provide comfort and convenience to their respective crews at the expense of higher operational costs, although the rate of automation and the level of safety have increased in the latest ships. Moreover, the aging of the currently operated ships could also make efficient operations difficult. For example, significant man hours are used in checking scattered spare parts on a ship and in manually recording a maintenance log. Much time and hard work is required in preserving and maintaining the ship's cargo because a few crew members are assigned to accomplish tasks related to the ship's cargo. The ubiquitous technology can be applied in various areas of ship operations so as to promote the latter's efficiency.

In this study, the wireless sensor network was employed as a ubiquitous technology on a real ship, and the ship's environment for such technology was investigated. The Zigbee platform was chosen for the sensor network field of the WSN system. In addition, several problems regarding real-ship application were identified, and basic experiments about the characteristics of WSN were conducted at the test bed ashore to provide reference data for the appropriate design of the WSN for a real ship. Especially, practical items concerning the communication depth of WSN and power consumption were discussed for the WSN design in a real ship. Through the WSN experiments in the main-engine room of the real ship, the increase of the communication depth was found to be necessary. The power consumption tests were carried out in the test bed and expected to provide a reference data for node deployments in a real ship.

2 Characteristics of Ships and of the WSN Technologies

A ship operates in a very harsh environment. Many steel plates are cut or welded together to make a block, and then a ship is constructed by putting many of these blocks together. This means that a ship is a big structure made mainly of steel. Besides this, the main engine of the ship transfers significant power to the propeller through a long shaft, for the propulsion of the ship. As the deckhouse, a superstructure on the upper deck of a ship, is located above the main-engine room, the crew members inside the deckhouse suffer from the vibration and noises induced by the propeller or main engine. Most of all, the ship motions, such as pitching, rolling, and heaving, can cause considerable inconvenience to the crew members and passengers according to the sea conditions. If the ubiquitous technology would be applied in ships, this kind of operational environment should be taken account of.

The most well-known ubiquitous technology is the RFID tag. The RFID tag, which has a specific identification code, is attached to an object, and various services (e.g., location finding [8], remote control, management, and communications between objects) are made possible by it with the use of radio waves, which produce an awareness of the surrounding environment. The price of RFID tags using the ultra-high frequency of 900 MHz or the microwave of 2.4 GHz has recently been lowered, and the price of RFID readers is going down as well. Although the RFID tag may ensure the safety and security of a ship's crew members and passengers (those carrying an RFID tag), it is very difficult to reliably recognize an RFID tag carried by a person who is moving about on the ship because of the essential characteristics of high frequency wave.

Another ubiquitous technology is WSN (wireless sensor network), which has been receiving considerable attention along with RFID. The WSN has a structure where several sensor networks are connected to an external network through a gateway. The sensor nodes send data to a sink node nearby, where the data from each node are accumulated before being transmitted to a gateway. The data transferred from the gateway can also be transmitted using satellite. The WSN technology has achieved significant growth because of the ad-hoc network technology and the routing protocol standard. Especially, wireless communication technologies such as WLAN (wireless LAN), Zigbee, and Bluetooth came into the spotlight in terms of the construction of efficient wireless networks. In other words, wireless communication has some problems to be solved for the successful ship application, such as the fact that it makes use of multipass and a radio wave screen due to the steel structure in a ship. As these problems can reduce the reliability of data transmission or recognition, it is necessary to prepare backup plans such as variations in network topologies and the addition of repeaters or relay antennas.

3 Full-Scale Ship Tests Using the WSN

3.1 Data Delivery Ratio for Zigbee

The particulars of the ship that was chosen for this study are shown in Table 1. Hannara, a 3,000-ton-class training ship, has relatively many cabins because it was designed to be sailed to train many students. Fig. 1 shows photo of the ship Hannara. The deckhouse of Hannara consists of the following, arranged vertically (from the bottom to the top): the bottom deck, which houses the main-engine room; the second deck, for the accommodations of students; the shelter deck, for the accommodations of the crew, the boat deck, for the accommodations of faculty members; the training deck, for navigation training; and the bridge deck, which is used as a wheel house.

The preliminary WSN tests were conducted when the ship was moored to the port and when all the equipments in the main-engine room, except for the electric generator, were not being operated. The wireless communication tests were done in one cabin at the shelter deck, which is the middle deck of the ship. The door of the cabin is shown in Fig. 2. It was made of steel but its surface was covered with smooth materials. Data transmission tests were carried out using two ZPAs (Zigbee Protocol Analyzers), one sending 100 10-byte packets and the other receiving those packets.

The RF (radio frequency) strength, indicating the RSSI(received signal strength indication) as well as the number of packets, was measured to evaluate the transmission quality. We performed the measurements 10 times at each case. When the sending ZPA (located within the cabin) and the receiving ZPA (located outside the cabin) were 1 m away from the door, a 98~100% delivery ratio (= received packet number/sent packet number) and an RSSI of -27~-30 dBm were found, which are not bad for wireless communication. Here, dBm is an abbreviation for the power ratio in decibel (dB) of the measured power referenced to one milliwatt (mW). Zero dBm equals one milliwatt. The delivery ratio and RSSI were 76% and -37 dBm, respectively, when the receiving ZPA was at an 8 m distance from the cabin door. Although the cabin door was closed, wireless communication was found to be possible when a sensor node is arranged near the door because the cabin door has a ventilation window at its lower part as well as a small gap between the wall and the door itself, as shown in Fig. 2.

Fig. 1. Hannara for full-scale ship tests

The tests regarding delivery ratio were also conducted in the corridor fronting the cabin, with a width of 1.5 m and a length of about 40 m. The sending ZPA was at the fixed location near one end of the corridor, and the receiving ZPA was moved to the other end of the corridor, 5 m away from the sending ZPA. The test results regarding the averaged delivery ratio and RF strength are shown in Table 2, and a photo of the corridor is shown in Fig. 3.

Table 2 shows that the wireless communication network will be on if two sensor nodes are within the line of sight, even at a distance of 40 m. In this experiment, as each ZPA was positioned 1.5 m from the bottom, interference from crew members or passengers passing through the corridor is unavoidable. If the basic data about the possible interference from a person passing through the corridor would be prepared in the future, the optimized location of the sensor node or router without any interference would be found, and this could be used in designing the WSN for a real ship.

Fig. 2. Door of the cabin

Fig. 3. Corridor in the shelter deck

The next stage was to measure the data delivery ratio at the stairs and between decks. The stairs from the shelter deck to the boat deck had a height of about 3 m and had steel walls with a thickness of over 15 mm, providing a sort of shielding space from the microwave. The entrance door to the stairs at the shelter deck was opened, and the sending ZPA was positioned at the upper corner of the entrance. The receiving ZPA, on the other hand, was positioned at the top of the stairs. The transmission ratio was 100%, and the RSSI value was -10 dBm, indicating very good communication. When the receiving ZPA at the boat deck was moved to different locations to remove it from the line of sight, the delivery ratio and RSSI value were 100% and -14 dBm, respectively, at a distance of 2 m from the boat deck entrance. Even when the receiving ZPA at the boat deck was located just above the sending ZPA at the shelter deck, the delivery ratio and RSSI value were 98% and -21 dBm, respectively

Table 1. Particulars of the ship Hannara

Length (m)	102.7
Width (m)	14.5
Height (m)	7
Tonnage (ton)	3640
Speed (knot)	15 (max. 17)
Main Engine	4,000HP diesel engine

The entrance door to the boat deck was always open, but the entrance door to the second deck, below the shelter deck, was made of steel and did not have any ventilation window. A delivery ratio of 63% and an RSSI of -46 dBm were obtained

Table 2. Wireless Communication Tests for the Long Corridor in the Shelter Deck

Distance (m)	Averaged delivery Ratio (%)	Averaged RF Strength (dBm)
5	98	-13
10	100	-12
15	94	-20
20	100	-17
25	100	-15
30	98	-27
35	100	-23
40	100	-25

when the receiving ZPA was beyond the entrance door to the second deck, and when the steel door was closed. From these experiments, the wireless communication network was found to be on when the steel door of the entrance to each deck was open. In addition, the closing of the entrance door to each deck would seriously reduce the quality of the wireless communication through the stairs even if there was a small gap between the entrance door and the side walls. Therefore, the plans should be established in such a way as to ensure that the wireless communication would not be interrupted, by using some bypass antennas or replacing the wireless network with the wired network at the troubled region.

3.2 WSN Tests in Main Engine-Room

The basic WSN tests were conducted in the main-engine room, which had several equipments and was considered very important in the monitoring of the ship's environmental conditions. The main engine was stopped, but the electric generator was operated. Fig. 4 shows the main-engine room viewed from the main-engine control room, and the laptop computer and sink node were placed on the control console. Since various equipments such as pumps or filters, as well as the main engine, were arranged in a complex way inside the main-engine room, it was very important to properly employ the routers and sensor nodes. In the basic experiments, since there were no advance reference data about the delivery ratio or RSSI according to the microwave diffraction, the routers and sensor nodes were employed at places on the line of sight. A schematic diagram of the sensor network field is shown in Fig. 5. The sink node, indicating the gateway, was placed at the top of the console, whose height was 1.7 m, inside the control room. Router 1 was placed on the top of an equipment that was 6 m away from the sink node and that had a height of 1.7 m. Router 2 was placed on the top of a cylindrical steel structure that was 7 m away from router 1 and that was near the stairs leading to the second deck. Router 3 was placed at the entrance door to the second deck, and was about 4 m away from router 2. Finally, the sensor nodes were positioned at the appropriate places based on the communication depth, and they communicated with each other from the main-engine

control room to the utility room of the 2nd deck by hopping. Sensor nodes 1 and 3 measured the temperature and humidity simultaneously. The intensity of illumination was measured by sensor nodes 2 and 4. Router 2 had sensor nodes 1 and 2 as its children, while router 3 had sensor nodes 3 and 4 as its children. The communication depths of routers 1 and 2 were 1 and 2, respectively. The child of each router had a depth one step smaller than the router's depth. The empty space in the second deck pertains to the path of the large exhaust pipes for the main engine. After the operation of the WSN, it was confirmed that the wireless communication worked well and that it could be useful for the monitoring of equipment or for fire prevention.

Fig. 4. Control room for main-engine

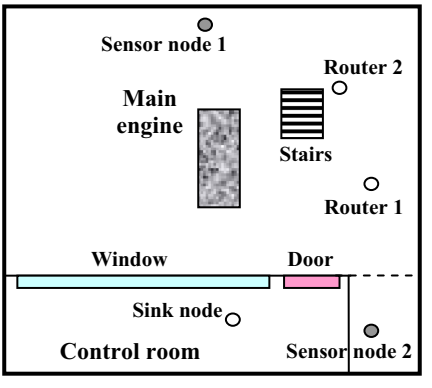


Fig. 5. Employment of WSN at the bottom deck (top view)

Several problems were found, although the basic experiments on the WSN that were conducted on a real ship were generally successful. The first problem was the communication depth of the WSN. Router 3 could allow its children to have only a fourth depth because the total depth of this WSN was four. Although the ship Hannara has a considerably small main-engine room, the monitoring of both the bottom and second decks was done partially due to the restricted communication depth of the WSN. In other words, the equipment and environment at the right side of the main-engine room can be monitored, but the monitoring of the equipment and environment at the left side is nearly impossible. In the second deck, the worst situation occurred from the viewpoint of the monitoring of equipment and environment. Therefore, it would be better to monitor only one deck in the case of the WSN with a small depth. To monitor two decks simultaneously, it is necessary to increase the depth of the WSN.

The second problem was related to the battery consumption of the sensor nodes. As most of the companies that are producing WSN modules design and manufacture these through approximate estimation, without accurate information regarding battery consumption, useful reference data are certainly necessary. There are some parts of the monitoring process that require a short sensing period, but the other parts may require a rather long sensing period. Power consumption tests were found to be necessary in terms of the sensing period. Especially, as the use of gas sensors for fire prevention will require significant power consumption, it would be desirable to adopt a regular power service.

The third problem that was found concerns microwave diffraction. In the present study, the routers and sensor nodes were placed within the line of sight because there were no reference data about the diffraction. This, however, is not efficient for the WSN design because the communication depth can be consumed in vain to avoid obstacles within the line of sight. The basic data related to microwave diffraction are necessary for the WSN design together with the consideration of multipath effect in a ship.

The last problem that was found concerns placement. Although the main engine was not actually operated, there were parts of the main engine that had a high temperature at the bottom deck. As the positioning of the routers and sensor nodes near the parts of the main engine with a high temperature would result in damages on the WSN modules, the placement of the modules should be carefully considered. In addition, high vibration and noises as well as high-temperature parts should be treated with caution when the main engine is being operated.

4 Investigation of the WSN Characteristics at the Test Bed

4.1 Battery Consumption Tests

It is necessary to investigate the characteristics of the sensor node for the appropriate application of it to a real ship. Although the sensor node of the present WSN needs low power instead of regular power, the lowest battery limit for its normal operation, and the life of the battery, should be checked in advance. Especially, if the crew must frequently replace each battery with a new one during a long-term voyage, which is expected to last for over 10 days, the crew would be annoyed and might no longer want to use the WSN system. Thus, basic data concerning the time of a battery replacement, as well as the battery consumption, will be helpful for the application of the WSN to a ship.

In the case of the use of a sensor node encouraged by Zigbee alliance, the request from the sink or coordinator can change the sleep mode into the wake mode, and vice versa. However, as this method requires that each sensor node be ready to continuously recognize requests from the sink, the sensor node does not completely realize low power consumption. The sensor nodes that were used in this study have a micro clock outside their RF and MCU chips, which would allow them to synchronize the operation modes and to realize low power consumption because only the micro clock would be awakened, and the other chips would continue being in the sleep mode. In spite of the sensor nodes' realization of low power consumption, the instability of the WSN system due to its fast power consumption should be considered at the operation mode with a short sleeping period. In this study, battery tests related to the sensor nodes were conducted at the test bed on land.

To investigate the power consumption rate of each battery, the remaining voltage in the battery was measured for the temperature and gas sensor nodes with an 8 sec sleeping period. The voltage of the battery was initially 3.4 V, and the voltage value was measured continuously for about 100 hr from 2 p.m. The lowest voltage limits of the temperature and gas sensor nodes were 2.73 V and 2.23 V, respectively. The communication was interrupted because the RF module stopped operating when the

remaining voltage in the battery became lower than the minimum voltage required. Fig. 6 shows the remaining voltage and temperature measured according to the time variation. The power of the battery was consumed independently of the temperature value, and the remaining voltage in the battery had a smooth and stable time history. In addition, since the temperature sensor node sensitively detected the ambient temperature, it can be said that the temperature environment can be controlled in the future by monitoring the temperature as part of context awareness. In the case of the gas sensor node, it excessively consumed the power of the battery from the beginning of the gas monitoring. The remaining voltage started from about 2.9 V and reached the minimum voltage required within 40 hr, as shown in Fig. 7. The sleeping time of 8 sec. was too short for the gas sensor node to be appropriately operated, and a time period of over a minute would be required for a much longer operation. Consequently, it is inappropriate for the gas sensor node to have the ability of low power consumption for the monitoring of natural gas, methane, butane, and other gases to prevent an explosion or fire in a real ship. The gas sensor node with regular power, indicated by a power adaptor, would instead be desirable for the real application of the WSN system.

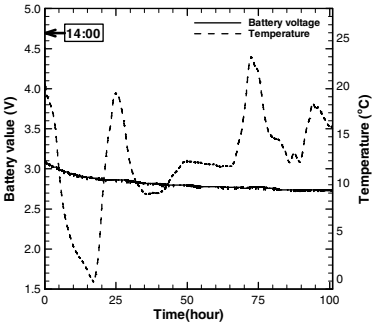


Fig. 6. Temperature sensor node

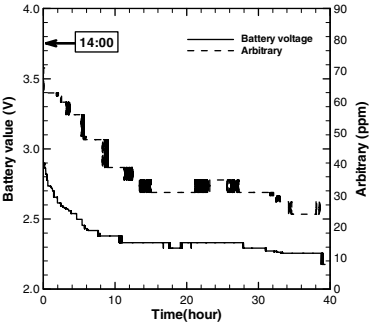


Fig. 7. Gas sensor node

When the sleeping time of the temperature sensor node became 64 sec, the remaining voltage was 2.88 V after about 205 hr. Based on its relation with the voltage consumption data of the 8-sec period, the life of the battery was predicted to be about 15 days. The power consumption characteristics of the battery should be understood beforehand because the ship owner or crew would be afraid to use the WSN system with sensor nodes whose battery has a short life.

5 Conclusion

In the present study, a monitoring system was introduced for the abnormal operation of equipment and for fire prevention in dangerous regions, which are important application fields of the WSN technology. Several realistic problems were found from the basic WSN tests that were conducted on a real ship. The regular wireless communication is generally possible within a ship; however, the reference data

regarding the communication depth, the power consumption rate of the battery in a sensor node, and the placement of the RF modules should be obtained more for the WSN design of a full-scale ship. In addition, the enduring time of each sensor node should be investigated carefully considering the multipath effects to provide a real ship with a useful WSN system besides the battery consumption tests at the test bed on land.

Acknowledgments. This research was accomplished with support from the inherent research project entitled "Development of smart operation technologies for exploration fleets based on the ubiquitous concept (PE0116A)."

References

1. MARINTEK., Intelligent hull-monitoring systems for the reduced risk of structural failure, spill into the sea, and damage to cargo, and for improved passenger safety and comfort (HULLMON+). G3RD-CT-2000-00329, EU Framework Programme (2000)
2. Nguyen, T.V., Nelson, H.W.: A systems approach to machinery condition monitoring and diagnosis. In: NDIA: Proceedings of the 4th Annual Systems Engineering Conference (2001)
3. Nielsen, J.K., Pedersen, N.H., Michelsen, J., Nielsen, U.D., Baatrup, J., Jensen, J.J., Petersen, E.S.: Sea sense: real-time onboard decision support. Annual Report, Force Technology (2006)
4. Cho, S.R., Lee, D.K., Paik, B.G., Yoo, J.H., Park, Y.H., Park, B.J.: A study on USN technologies for ships. In: Proceedings of Ubiquitous Intelligence and Computing, July 11-13, 2007, Hong Kong, China (2007)
5. Hakem, N., Misson, M.: Study of the throughput of the wireless home automation network using the encapsulation of two medium-access methods. In: Proceedings of Communication Systems and Networks, September 9-12, 2002, Spain (2002)
6. Oppermann, L., Broll, G., Capra, M., Benford, S.: Extending authorizing tools for location-aware applications with an infrastructure visualization layer. In: Proceedings of Ubiquitous Computing, September 17-21, 2006, Orange County, CA, USA (2006)
7. Hodges, S., Williams, L., Berry, E., Izadi, S., Srinivasan, J., Butler, A., Smyth, G., Kapur, N., Wood, K.: SenseCam: a retrospective memory aid. In: Proceedings of Ubiquitous Computing, September 17-21, 2006, Orange County, CA, USA (2006)
8. Yun, K.H., Choi, S.W., Kim, D.J.: A robust location tracking system using the ubiquitous RFID wireless network. In: Proceedings of Ubiquitous Intelligence and Computing, Wuhan, China, September 3-6, 2006 (2006)

Improving the Performance of the Wireless Data Broadcast by the Cyclic Indexing Schemes

Long-Sheng Li, Ming-Feng Chang, and Gwo-Chuan Lee

Department of Computer Science and information Engineering,
National Chiayi University,
No.300 Syuefu Rd., Chiayi 60004, Taiwan, R.O.C.
sheng@mail.ncyu.edu.tw, mfchang@csie.nctu.edu.tw,
gclee@nuu.edu.tw

Abstract. Wireless data broadcast is an effective approach to disseminate information to a massive number of users. Indexing techniques for broadcasting data can reduce the battery power consumptions of mobile terminals by decreasing the tuning time. The organization of the indexes affects the efficiency of data searching. We investigate how the degree of the index node affects the tuning time, and thus minimize the power consumption of user's terminals. We proposed a performance measurement for the tuning time and a cyclic indexing algorithm. The numerical results suggest the degree of an index node be 3 when the access probabilities of the data tend to be uniformly distributed so that the expected tuning time is minimal. When the access distribution of the data nodes is skewer, the tuning time can be minimized by setting the degree in the index node 2.

Keywords: Broadcast, wireless, tuning time, tuning cost, access time, the Hu-Tucker algorithm.

1 Introduction

Wireless data broadcast is an efficient technology to overcome the limited bandwidth in the ubiquitous computing. Wireless data broadcast over radio channels allows users to access data simultaneously at a cost independent of the number of users. It is a powerful way to disseminate data to a massive number of users in the ubiquitous computing. A centralized server periodically broadcasts the data to a large number of mobile terminals through a wireless medium. The mobile terminals receive the broadcasts and filter out the data that is not desired [2]. This service is especially useful for disseminating data that are commonly accessed, such as traffic information for navigation system and real-time stock information. The location-dependent web service can also utilize wireless data broadcast. One disadvantage of the wireless data broadcast is that the mobile terminals can only wait for the requested data.

Power-efficient wireless communication is another important issue for the wireless data broadcast. A simple way to reduce the power consumption is to add auxiliary

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 123–133, 2007.

information to enable the mobile terminals to receive only the data the user needs. A mobile terminal can be three power modes: transmission mode, receiving mode, and doze mode.

There are two basic approaches for users to access data through radio channels [8]. One is push-based scheme, where the clients retrieve data by only listening to a certain channel in the receiving mode. The clients cannot inform the broadcast server about what they need due to the lack of uplink communication channels from the users to the server. The other approach is pull-based scheme where the clients send requests to retrieve data. There is an uplink channel through which a client can send requests for specific data items to the broadcast server. The broadcast server may arrange the broadcast sequence according to the request received. In the power management view, the client saves power by avoiding transmissions and waking up from the doze mode only when necessary. The push-based scheme is a better strategy to overcome the limited battery power.

To evaluate the efficiency of the wireless broadcasting, two criteria are often used: access time and tuning time [1]. The access time is the average time from the moment a client requests data identified by a primary key to the point when the requested data are received by the client. The access time determines the response time of data access. The tuning time is the time spent by a client listening to the channel. The tuning time determines the power consumed by the client to retrieve the requested data. Indexing techniques insert auxiliary information indicating when each data item will be broadcasted to reduce the tuning time [1][2][4][5]. After receiving the index, a client waits for the requested data most of time in the doze mode in which low power is consumed and only wakes up to receive data when the requested data is coming. Therefore, the tuning time can be reduced and the battery power is conserved.

2 The System Architecture

A broadcast server broadcasts the data to the clients by radio channels. The clients receive the broadcast data and the requested data are filtered. To consume less power of the clients, the broadcast server inserts indexes before the broadcast data to indicate the offsets to the requested data. The clients receive the indexes and go to doze mode. When the requested data are broadcasted, the client wakes up to the receiving mode and receives the requested data. Moreover, to provide efficient search of the indexes, an alphabetic Huffman tree is used for the indexing tree. The clients using this scheme should tune to the beginning of the indexes to get the offset to the requested data. The waiting time between the start of tuning and the beginning of the indexes is half of a broadcast cycle in average. This is referred to the distributed indexing scheme [1].

To reduce the access time of the distributed indexing scheme, the broadcast bandwidth is spilt into several physical channels or logical channels [9]. All data are assigned into a data channel. The indexes of the same level are proposed to occupy on the same channel. Fig. 1 shows the indexing tree and the channel assignment of the broadcast data.

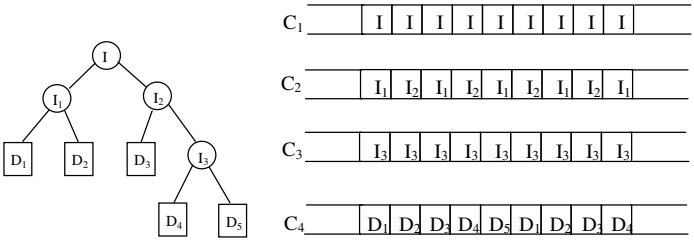


Fig. 1. The indexing tree and the channel assignment of the broadcast data

The distributed indexing scheme assumed the access probabilities of the data items are the same. Shivakumar and Venkatasubramanian released the assumption [9]. Let n be the number of data items. Every data item has the popularity probability to indicate the expected number of access to the data items. We assume the popularity probabilities of the data items be $f_1, f_2, \ldots, f_n$. If the tuning time for the data item i is T_i, the average tuning time is

$$\left(\sum_{i=1}^{n} f_i \times T_i\right) / \left(\sum_{i=1}^{n} f_i\right).$$

The tuning time T_i is dependent on the depth of the data item in the indexing tree. The problem to construct an indexing tree to minimize the average tuning time is similar to the Huffman coding, but the alphabetic ordering of the data items is preserved. Hu and Tucker proposed an algorithm to optimize the alphabetic-ordered Huffman code [11][12]. Shivakumar and Venkatasubramanian suggested a k-ary Hu-Tucker algorithm to minimize the average tuning time, but didn't describe the algorithm clearly [9].

An open problem is remained unsolved in the k-ary Hu-Tucker algorithm. For some n, it is impossible to construct a tree that the branches of all internal nodes are filled with k nodes. The k-ary Hu-Tucker algorithm constructs the internal nodes with 2 to k branches, but doesn't specify exact rules to construct the internal nodes. A strategy to determine the branches of the internal nodes of an indexing tree to obtain the minimal average tuning time is needed for the k-ary Hu-Tucker algorithm.

The tuning time is determined only by the depth of the indexing tree. If we increase the branches of the index, the tuning time is reduced. However, increasing the branches should increase the capacity of the index. For the wireless broadcasting, the indexes can be broadcasted on the index channels. The size of the index represents the bandwidth requirement of the radio channel. In wireless communications, a radio channel is partitioned into slots of constant size. The 3rd generation personal communication service provides the function to assign the fixed bandwidth of the channel to a dedicated service [8]. Therefore, the size of the indexes should be the same to fit in a time slot. The tuning time should count the number of indexes received and the size of the index. Assume the depth of the data node i is d_i, and the degree of the index is k. β represents the length of the key and the offset. The average tuning time can be expressed as:

$$\overline{T} = \frac{k\beta\sum_{i=1}^{n}(d_i - 1)f_i}{\sum_{i=1}^{n} f_i} = k\beta\sum_{i=1}^{n}(d_i - 1)f_i \,. \tag{1}$$

3 The Proposed Alphabetic Huffman Algorithms

Huffman tree and minimize the average tuning time. For n data nodes, it may not be possible to construct an index tree where all indexes have exact k downward branches. That is, empty branches are remained in some indexes. We call this category of index as the incomplete index. In our proposed algorithm, we gather those empty links in one index of the index tree, i.e., there is only an incomplete index in the tree. Under this assumption, the number of the non-empty links of the incomplete index can be determined from the number of data nodes, n, and the degree, k. Let a % b represent the remainder of a/b, and k_1 be the number of the non-empty links of the incomplete index. k_1 can be expressed as

$$k_1 = \begin{cases} b + (k-1), & \text{for } n \,\%\, (k-1) = b \text{ and } b = 0 \text{ or } 1 \\ b, & \text{for } n \,\%\, (k-1) = b \text{ and } b \neq 0,1 \end{cases}. \tag{2}$$

Using the techniques of the binary Hu-Tucker tree, we construct the k-ary index tree by merging k nodes with the least access probability into an index node of the index tree. The access probability of the index is the sum of the access probabilities of its k children. The number of the non-empty links of the incomplete index is calculated first. It is due to that we reduce the average tuning time by merging nodes with less access probability into an index in the lower level of the tree. This algorithm will be referred to as the k-ary Incomplete-index First Alphabetic Huffman Algorithm (IFAH). The algorithm is shown in the following.

Step 1. Let S=(N_1, N_2, …, N_n), the ordered list consists of all data nodes sorted by search key in increasing order. N_i=D_i.

Step 2. Calculate the number of the non-empty links k_1 of the incomplete index using Equation 2.

Step 3. Construct the incomplete index node.

- Find k_1 consecutive nodes in S whose sum of access probabilities is minimum.
- Replace the k_1 consecutive nodes with an index node in S. The access probability of the index node equals to the sum of the access probabilities of the replaced nodes.

Step 4. If |S|=1, then go to Step 7.

Step 5. Construct the k-degree index nodes.

- Find the k "consecutive" nodes, but index nodes can be bypassed, in S that have the minimum sum of access probabilities.
- Replace the k "consecutive" nodes with a new index node in S.

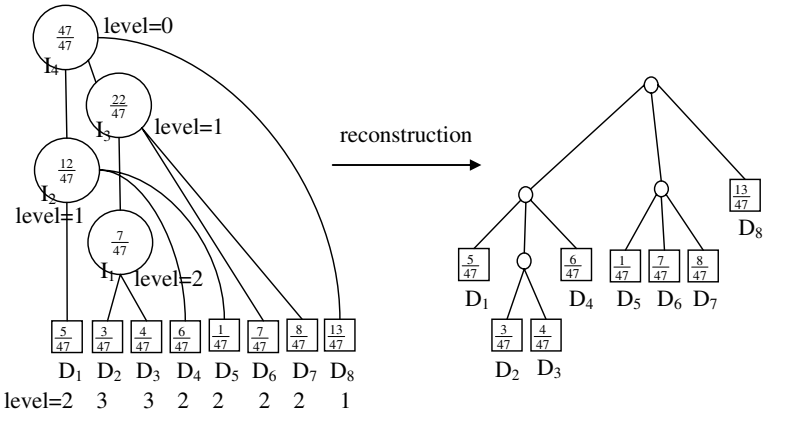
Step 6. If |S|=1, then go to Step 7.
Else go to Step 5.

Step 7. Determine the level of each data node in the index tree.

Step 8. Reconstruct the index tree according to the levels of the data nodes.

- Initialize the ordered list S as in Step 1.
- Find k_1 consecutive data nodes whose levels are the same. The levels of k_1 consecutive data nodes must be the maximum among the remaining nodes.
- Combine the k_1 consecutive nodes to an index node. Replace the k_1 consecutive nodes with the index node in S.
- Find k consecutive nodes whose levels are the same and the maximum among the remaining nodes, and combine the k consecutive nodes at the highest level first. Then, the nodes on the next-to-highest level are combined.
- The process continues until there is only one node left and its level must be 0.

Fig. 2 shows an example index tree obtained by the IFAH. The boxes represent the data nodes and the numbers in the boxes represent the access probabilities of the data nodes. The circles represent the index nodes and the numbers in the circles represent the combined access probabilities of the linked nodes. i is the key for the data node D_i. The IFAH constructs index node I_1 first, because D_2 and D_3 have the minimum sum of access probabilities. In constructing index node I_2, index node I_1 is bypassed, because D_1, D_4, and D_5 are the 3 "consecutive" nodes that have the minimum sum of access probabilities. The index nodes construction process continues until only one node is left in S. We assign level 0 to the root. According to the links of the indexes, we assign the level values to all index nodes and data nodes. Then, the index tree is reconstructed from the highest level of the data nodes.



(a) The index tree before reconstruction (b) The index tree after reconstruction

Fig. 2. An example of the IFAH

The format of an index is as follows,

Key 1	Offset 1	Key 2	Offset 2	Key 3	Offset 3

Key i is the boundary key value for searching requested data. If the key of the requested data is larger or equal to Key i and less than Key (i+1), Offset i is the offset for the index or data slot of the lower level in the index tree.

Note that the index tree in Fig. 3 places D_1 in its left-most leaf, i.e., the index tree starts from D_1, the first data node. However, a *k*-ary alphabetic tree does not necessarily start from D_1; it can starts from any data node. Fig. 3 shows the index trees starting from different data nodes. The numbers on the links under the index nodes are the boundary key values of the index nodes. Fig. 3 (a) is an example of *k*-ary alphabetic tree starting from D_1. Fig. 3 (b) shows an example of *k*-ary alphabetic tree starting from D_2; the data node before the D_2 is rotated to the end of the ordered list. In this example, we show how to retrieve data node D_1. The boundary key values of the root index are 2, 5, and 6. The key of D_1 is less than 2. Therefore, we chose the offset of boundary key value 6 to obtain the index of the next level. The index of the next level shows the offset of the requested data node D_1. This shows that a *k*-ary alphabetic tree can start from any data nodes. That is, the alphabetic order of the data nodes in the index tree can be treated as a cycle. The average tuning times are 5.62 in Fig. 3 (a) and 4.49 in Fig. 3 (b), respectively. We apply the rotatability to improve the IFAH. The new algorithm will be referred to as the *k*-ary Cyclic Incomplete-index First Alphabetic Huffman Algorithm (CIFAH). The CIFAH modifies Step 3 and 5 of the IFAH. In CIFAH, we treat the ordered list as a cycle and find the minimum sum of access probabilities.

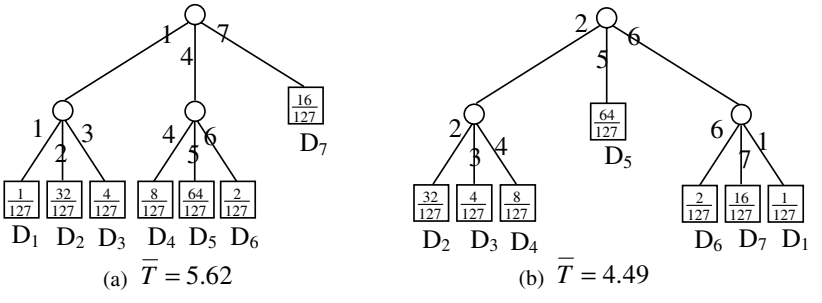


Fig. 3. The index trees with different alphabetic orders

4 The Numerical Analysis

To simplify the analysis of the tuning cost of the proposed alphabetic Huffman algorithms, we made the following assumptions:

- There is no fault in the broadcasting or reception.
- The initial probe is uniformly distributed in the broadcast cycle.

First, consider a special case where the access probabilities of the data nodes are identical, and the optimal index tree is a full *k*-ary tree that has no incomplete index. Let *k* be the degree in each index, and *d* be the depth of the index tree. The number of data nodes is $n = k^{d-1}$. The average tuning time is $\overline{T} = k(d-1)$. If *k* could be any real number, the average tuning time can be minimized when $\frac{d\overline{T}}{dk} = 0$.

$$\frac{d\overline{T}}{dk} = \frac{d(\frac{k \ln n}{\ln k})}{dk} = 0 \Rightarrow k = e = 2.71828...\cdot$$

Since k is a natural number, the result suggests that the average tuning time may be minimized when the degree of the index is 3.

Release the limitation of the full k-ary tree, we assume the probability distributions of all data nodes are uniformly distributed. That is, $f_1 = f_2 = f_3 = \cdots = f_n = 1/n$ and $d = \lceil \log_k n \rceil$. The index tree is a full k-ary tree when $n = k^d$. For $k^d - k + 2 \le n < k^d$, all leaves are at the same level (level d). The average tuning time is $\overline{T} = k(d-1)$. For $k^d - 2k + 3 \le n \le k^d - k + 1$, there are one leaf at level (d-1) and (n-1) leaves at level d. The average tuning time is $\overline{T} = k(n(d-1)-1)/n$. For $k^d - 3k + 4 \le n \le k^d - 2k + 2$, there are two leaves at level (d-1) and (n-2) leaves at level d. The average tuning time is $\overline{T} = k(n(d-1)-2)/n$. For $k^d - k^{d-1}k + (k^{d-1}+1) \le n \le k^d - (k^{d-1}-1)k + (k^{d-1}-1)$, there are k-1 leaves at level (d-1) and (n-k+1) leaves at level d. The average tuning time is $\overline{T} = k(n(d-1)-(k^{d-1}-1))/n$. Therefore, if $k^d - (i+1)k + (i+2) \le n \le k^d - ik + i$,we have

$$\overline{T} = k(n(d-1)-i)/n \text{ , for } i=0, 1, \ldots, k^{d-1}-1.$$

The average tuning time can be expressed as

$$\overline{T} = \frac{k(n(d-1) - \left\lfloor \frac{k^d - n}{k-1} \right\rfloor)}{n}, \text{ where } d = \lceil \log_k n \rceil \tag{3}$$

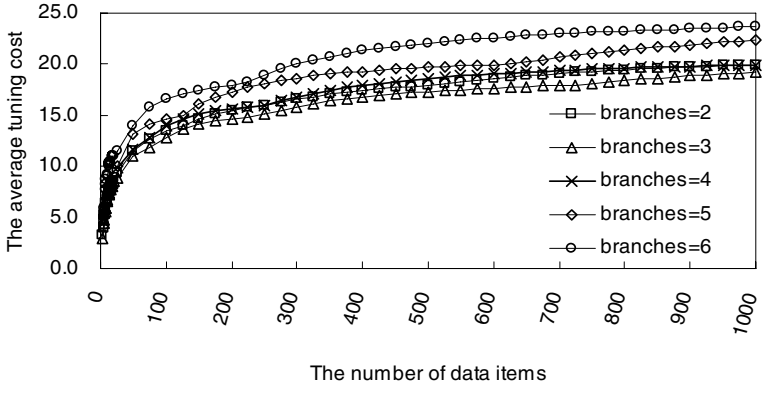


Fig. 1. The average tuning time for data with the uniform distribution

Fig. 4 shows the tuning time as functions of the number of data nodes and the degree of the index node. The access probabilities of the data nodes are all equal. The number of the data nodes varies from 2 to 1000. The five curves, in the figure,

represent the average tuning time for the cases where the degrees are 2, 3, 4, 5, and 6, respectively. The average tuning time increases as the number of data nodes increases due to the increasing height of the index tree. The tuning time increases as the degree of the index is larger than 3. Therefore, when the access probabilities are uniformly distributed, the index nodes of degree 3 tend to minimize the average tuning time.

Consider the case where the access probabilities are non-uniformly distributed. We assume the distribution of the access probabilities is Zipfian [9][14][13]. For n data nodes, the access probability of a data node D_i is as follows,

$$f_i = \frac{1}{i^r \times \sum_{i=1}^{n} 1/i^r},$$

where r is the rank of the distribution.

Note that, the larger the rank r is, the skewer the probability distribution is. In addition, f_i decreases as i increases. In this sector, we use the rank r to set the access probabilities of data nodes. Then, reorder the sequence of the data nodes using a random number generator. The number of possible sequence orders is $n!$. Therefore, it is impossible to evaluate all possible sequence orders for a large number of data nodes. To simplify the computation, the sequence order is randomly generated. In our experiments, we generate 10000 random sequences for each Zipfian distribution, and then generate the index tree for each random sequence order, and calculate the average tuning time.

Fig. 5 shows the results of the average tuning time for different ranks of Zipfian distribution. For a small rank (e.g., r=0.2) and a large number of data nodes, the minimum average tuning time can be obtained when the degree is 3. The results are consistent with that of the uniform distribution. It is because that a smaller rank for Zipfian distribution results in the less skew probabilities distribution. For a large rank (e.g., r=2) in Zipfian distribution, the minimal average tuning time is found when the degree is 2. This is because the large number of branches increases the tuning time of every data node in the index tree. Consider the index trees of a given degree. The skewer the access probability distribution is, the less the tuning time is. This is

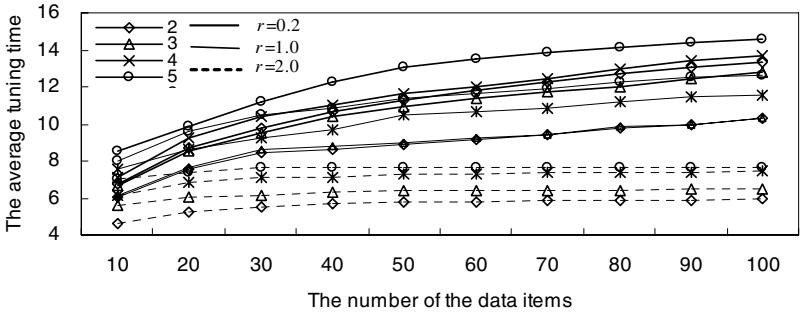


Fig. 2. The average tuning time of different number of the branches with r=0.2, 1.0, and 2.0 with the IFAH

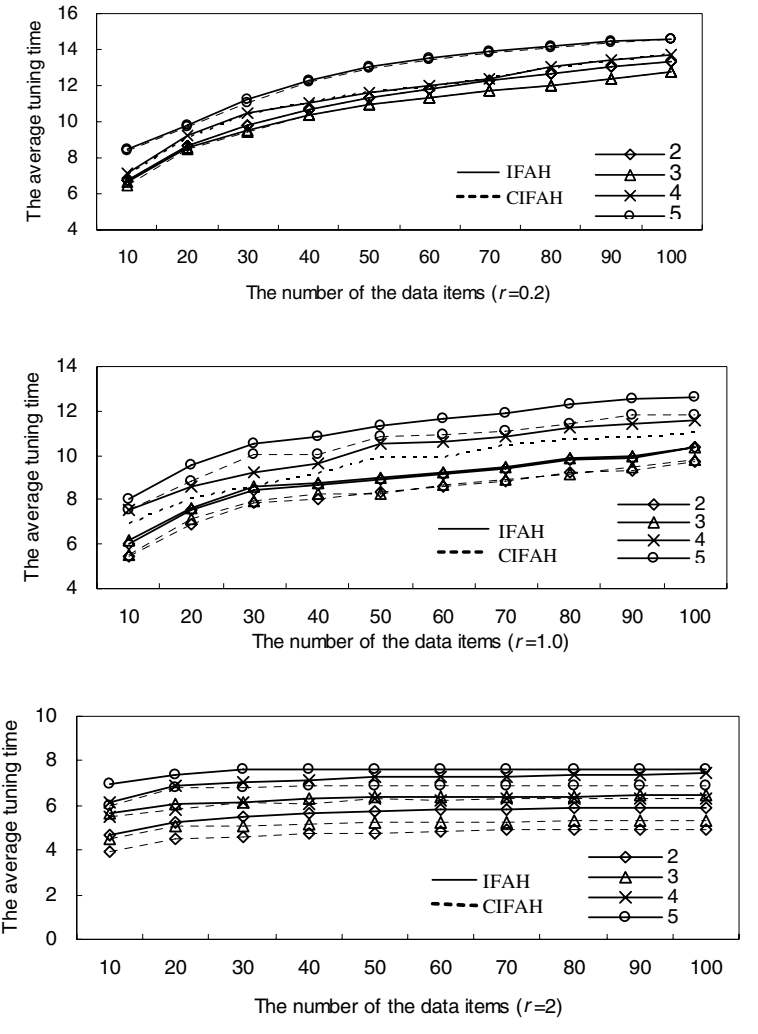


Fig. 3. The average tuning time of the IFAH and CIFAH

because as the access distribution gets skewer, fewer data nodes commands more access probability. The data nodes of large access probabilities trends to be placed at the lower levels of the index tree. As a result, the tuning time decreases.

Fig. 6 shows that the average tuning time of the CIFAH is less than that of the IFAH. The CIFAH is an efficient algorithm in reducing the average tuning time. It is because we can find the minimum tuning time from the selected nodes in the cycle sequence to build low cost indexes in the index tree. The improvement ratio of the CIFAH with large rank r is larger than that with small rank. This is because there are data nodes of larger access probabilities for the skewer access probabilities distributions. The CIFAH has the capability to find an ordered list for the data nodes

to construct an index tree that places those frequently accessed data nodes in the lower level. Therefore, the improvement ratio of the tuning time increases.

5 Conclusions

In this paper, we proposed indexing schemes to obtain minimal tuning time in the wireless broadcast system. The IFAH is an algorithm similar to the Hu-Tucker algorithm in organizing the indexes. To reduce the tuning time, the CIFAH can improve the IFAH by rotating the sequence of the data nodes.

From the experiments, we have the following results for the indexing schemes.

- If the access probabilities of the data are uniformly distributed, the tuning time is minimal when the degree of the index node is 3.
- For the data nodes whose access probabilities are Zipfian distributed, the tuning time increases as the number of the data nodes increases. It is because that the depth of the index tree increases as the number of the data nodes increases.
- The CIFAH can effectively reduce the tuning time when the access probabilities are of Zipfian distribution, since it is more likely to find consecutive nodes with less access probability to be merged into an index node in the rotatable data cycle.
- For the Zifian distribution, the improvement ratio of the CIFAH increases as rank r increases, i.e., the distribution gets more distorted. It is because skewer access probabilities let the CIFAH have more chances to find k consecutive nodes of less tuning access probability in the rotatable broadcast cycle to construct an index node in the index tree.
- The tuning time increases as the degree of the index increases, since index of large degree increases the tuning time of every data node in the index tree.

We provide the cyclic indexing construction schemes to reduce the average tuning time. To reduce the tuning time, the degree of the index in the index tree is suggested to be 2 or 3. The frequencies of the broadcasted data may not be uniform in a broadcast cycle. In the future, we can schedule the broadcast sequence according the access probabilities and a new indexing scheme is required to reduce the tuning time.

References

1. Imielinski, T., Viswanathan, S., Badrinath, B.R.: Energy Efficiency Indexing on Air. In: Proceedings of the International Conference on SIGMOD, pp. 25–36 (1994)
2. Imielinski, T., Viswanathan, S., Badrinath, B.R.: Data on Air: Organization and Access. IEEE Transactions on Lnowledge and Data Engineering 9(3), 353–372 (1997)
3. Su, C.-J., Tassiulas, L.: Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery. Wireless Networks 6, 279–288 (2000)
4. Lee, W.C., Lee, D.L.: Using Signature Techniques for Information Filtering in Wireless and Mobile Environments. Distributed and Parallel Databases 4(3), 205–227 (1996)

5. Lee, C.K., Leong, H.V., Si, A.: A Semantic Broadcast Scheme for a Mobile Environment Based on Dynamic Chunking. In: 20th International Conference on Distributed Computing Systems, pp. 522–529 (2000)
6. Saran, H., Shorey, R., Kumar, A.: Policies for Increasing Throughput and Decreasing Power Consumption in Bluetooth MAC. In: 2000 IEEE International Conference on Personal Wireless Communications, pp. 90–94 (2000)
7. Lo, S.-C., Chen, L.P.: An Adaptive Access Method for Broadcast Data under an Error-Prone Mobile Environment. IEEE Transactions on Knowledge and Data Engineering 12(4), 609–620 (2000)
8. Hu, J.-H., Yeung, K.-L., Feng, G., Leung, K.-F.: A Novel Push-and-Pull Hybrid Data Broadcast Scheme for Wireless Information Networks. In: 2000 IEEE International Conference on Communications, vol. 3, pp. 1778–1782 (2000)
9. Shivakumar, N., Venkatasubramanian, S.: Energy-Efficient Indexing For Information Dissemination In Wireless Systems. ACM-Baltzer Journal of Mobile Networks and Nomadic Applications 1, 433–446 (1996)
10. Peng, W.-C., Chen, M.-S.: Dynamic Generation of Data Broadcasting Programs for a Broadcast Disk Array in a Mobile Computing Environment. In: CIKM 2000. Proc. Of the ACM 9th International Conf. on Information and Knowledge Management, pp. 6–11 (November 2000)
11. Hu, T.C., Tucker, A.C.: Optimal computer search trees and variable-length alphabetic codes. SIAM Journal Applied Math. 21(4), 514–532 (1971)
12. Knuth, D.E.: Dynamic Huffman Encoding. Journal Algorithms 6(2), 163–180 (1985)
13. Li, W.: Random texts exhibit Zipf's law-like word frequency distribution. IEEE Trans. Information Theory 36(6), 1842 (1992)
14. Zipf, G.K.: Human Behavior and the Principle of Least Effort. Addison-Wesley Press, Cambridge, Massachusetts (1949)
15. 3GPP TS 25.324: Radio Interface for Broadcast/Multicast Services.

Revisiting Fixed Priority Techniques

Nasro Min-Allah[1,2,3], Wang Yong-Ji[3], Xing Jian-Sheng[1,3], and Junxiang Liu[3]

[1] Graduate University, Chinese Academy of Sciences, Beijing 100039, P.R. China
[2] Department of Computer Sciences, COMSATS University, 44000, Pakistan
[3] Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China
{nasar,ywang,jiansheng,liu}@itechs.iscas.ac.cn

Abstract. Discrete scheduling is preferred over continuous scheduling for preemptive scheduling problems, however, classical continuous schedulability tests can not work effectively with discrete scheduling. This area of integrating discrete scheduling with continuous schedulability tests remains unexplored. Two contributions are made in this paper; firstly, an empty-slot method is introduced and extended to discrete schedulability analysis and secondly, an efficient exact feasibility test is proposed that has lower complexity in contrast to current feasibility tests in terms of reducing the number of scheduling points, where task feasibility is analyzed.

Keywords: Real-Time Systems, Fixed priority Scheduling, Discrete Scheduling, Rate Monotonic Analysis, Time Demand Approach.

1 Introduction

Preemptive scheduling- a task may be preempted and resumed at some later stage- has become a mature field and a lot of literature is available [1], [2], [5]. Real-time systems implement preemptive scheduling through the priority scheme: higher priority job preempts a lower priority job. The system at each time instant assigns a priorities to active jobs and allocates the processor to the one acquiring the highest priority.

Currently, two approaches are available to implement preemptive scheduling algorithms in hard real-time systems: event-driven and timer-driven [6]. As long as preemption is concerned it can only occur at specified discrete time intervals for timer-driven approach, while for event-driven approach, preemptions occur when external interrupts arrive. In both case, interrupts are handled only, when current instruction is executed and system status is stored for later reference. As each instruction consumes a fixed number of CPU's clock time, a preemption only occurs at specified discrete time interval for event-driven approach. So it is a reasonable abstraction that preemptions occur only at discrete time intervals for hard real-time systems [7]. It is the preemption that makes scheduling either continuous or discrete. As integer values are easier to be implemented and operated than real numbers, discrete scheduling is preferred over continuous scheduling in hard real-time computer systems.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 134–145, 2007.

Schedulability analysis is crucial for hard real-time systems to maintain system timing constraints. A lot of work has been done on Rate Monotonic (RM) and Deadline-Monotonic (DM) analysis [4], [9], [13], [16], [17]. However, most of the work done is focused on i) continuous schedulability analysis and ii)their complexity is pseudo-polynomial (in exact form). In this paper, we categorically address the above issues in Section 2 and 3, respectively.

In [10], Santos et al. proposed an empty-slot method and applied it to analyze the schedulability of communication tasks in real-time LANs. However, their task system is restricted to implicit-deadline and the execution time of each task must be equal to 1. Subsequently, Santos and Orozco [11] extended this method to MTSP (Multiple Tasks-Single Processor) systems and removed the restriction on the execution time of tasks but the task system is still restricted to implicit-deadline and the method is applicable to the RM schedulability analysis only.

Up to now, there has been no general method for analyzing discrete schedulability. We address this issue in Section 2. where we first analyze the empty-slot method and extend it for general discrete schedulability analysis. Then we analyze the discrete schedulability with a more general, constrained-deadline, synchronous task system and propose a new discrete schedulability test, which can be applied to all static-priority scheduling algorithms. This paper also presents a novel technique to handle the complexity of exact tests by reducing the number of inequalities which are needed to be tested otherwise for determining system feasibility, in Section 3. Finally, conclusions are drawn in Section 4.

2 Discrete Scheduling by Empty-Slot Method

2.1 Empty-Slot Method

Empty-slot method [10] is an appropriate approach for general discrete schedulability analysis. The main idea is: time is slotted and the duration of one slot is taken as unit of time. For any task τ_i, its parameters c_i, p_i and d_i are mutually commensurable and are assumed to be positive integers. Slot 1 is the first slot. Initially, all slots are empty. Each task acquires its slots according to the scheduling algorithm. Authors in [10] represented a node by a task, which must transmit its messages in one slot. The network is specified as a task set $\tau = (1, p_1), ..., (1, p_n)$. The expression $W_n(t) = \sum_{h=1}^{i-1} \lceil \frac{t}{p_1} \rceil$ gives this worst case of load in the interval $[0, t]$. Set τ is said to be non-saturated iff $W_n(M) < M$, where M denotes the least common multiple of all tasks. It has been proved that τ is RM schedulable iff for $i = 2, ..., n, \sum_{h=1}^{i-1} \lceil \frac{t}{p_h} \rceil < 1$ and $p_i \geq e_{1(i-1)} =$ least $t : t = 1 + \sum_{h=1}^{i-1} \lceil \frac{t}{p_n} \rceil$, where $e_{1(i-1)}$denotes the first empty slot of the higher priority $i-1$ tasks. There exist some constraints on the above task model i.e. tasks must have transmission time (execution time) equal to 1 and deadline equal to period.

Santos et al. extended this method to MTSP systems and the constraints on execution times were weakened by assuming any integer values [11]. However, the task model is still implicit-deadline synchronous. τ is RM schedulable iff $\forall i \in$

$2, ..., n, \sum_{h=1}^{i-1} \frac{c_h}{p_h} < 1$ and $p_i \geq e_{c(i-1)}$, where $e_{c(i-1)}$ denotes c_i-th empty slot of the higher priority $i-1$ tasks. From the above results, it is a reasonable conjecture that a similar result can be obtained for constraint-deadline synchronous task model when the timing constraint $\forall i, d_i = p_i$ is weakened with $\forall i, d_i \leq p_i$.

2.2 Discrete Static-Priority Schedulability Analysis

Preliminary Results. Let N denotes a natural number and $N^+ = N - \{0\}$. Let us define the function:

$$f(t) = \sum_{i=1}^{n} c_i \lceil \frac{t}{p_i} \rceil \tag{1}$$

Since $f(t)$ gives the maximum load in the interval $[1, t]$. Some conclusions can be drawn:
A. $f(t_2) - f(t_1)$ is the workload in the interval $[t_1, t_2]$
B. $f(t+1) - f(t)$ is the workload generated at the beginning of slot $t+1$. If the system is schedulable and $f(t+1) > f(t)$, then the interval $[t+1, t+(f(t+1)-f(t))]$ is full. If $f(t+1) = f(t)$ then slot $t+1$ can be empty.
C. if $f(t) \geq t$, slot t is full, otherwise slot t could be either full or empty.
It is obvious that $f(t)$ is non-decreasing monotonically. We can prove the following lamma :

Lamma 1

$$f(t_1 + t_2) = f(t_1) + f(t_2) - j, j \in \{0, 1, ..., \sum_{i=1}^{n} c_i\} \tag{2}$$

Proof. As

$$\lceil (a+b)/c \rceil = \begin{cases} \lceil a/c \rceil + \lceil b/c \rceil \\ \lceil a/c \rceil + \lceil b/c \rceil - 1 \end{cases} \quad \forall a, b, c \in N^+$$

$$\lceil \frac{(a+b)}{c} \rceil = \lceil \frac{a}{c} \rceil + \lceil \frac{b}{c} \rceil - j, j \in 0, 1$$

$$\text{then} f(t_1 + t_2) = \sum_{i=1}^{n} \lceil \frac{t_1 + t_2}{p_i} \rceil c_i$$

$$= \sum_{i=1}^{n} \{ \lceil \frac{t_1}{p_i} \rceil c_i + \lceil \frac{t_2}{p_i} \rceil c_i - j_i c_i \}$$

$$= \sum_{i=1}^{n} \lceil \frac{t_1}{p_i} \rceil c_i + \sum_{i=1}^{n} \lceil \frac{t_2}{p_i} \rceil c_i - \sum_{i=1}^{n} j_i c_i$$

$$= f(t_1) + f(t_2) - j$$

where $j = \sum_{i=1}^{n} j_i c_i$, as $j_i \in \{0, 1\}, j \in \{0, 1, ... \sum_{i=1}^{n} c_i\}$. We can see that τ repeats its behavior every M slots and to study its behavior it suffices to analyze it in the interval $[1, M] \leq M$. If τ is schedulable, the workload generated by τ

in the interval $[1, M]$ must execute to completion, therefore $f(M) \leq M$ is a necessary condition for τ to be schedulable. If $M - f(M) > 0$, then it gives the number of empty slots in the interval $[1, M]$ and the system is said to be non-saturated. If $M - f(M) = 0$, the system has no empty slots in the interval $[1, M]$ and is called saturated.

Theorem 1. *For a non-saturated system under any static-priority scheduling algorithm, the i-th empty slot e_i is the minimum value such that $t = i + f(t)$, $t \in [1, +\infty], \forall i \in N^+$.*

Proof. As e_i is the i-th slot, there are $i-1$ empty slots in the interval $[1, e_i - 1]$. Since e_i is empty, workload generated by all tasks in the interval $[1, e_i - 1]$ can execute to completion. Therefore, $f(e_i-1) = (e_i-1)-(i-1) = e_i-i$. No workload is generated at the instant e_i, $f(e_i) = (e_i - 1)$, namely $e_i = i + f(e_i)$. Here we prove that e_i is the minimum $t|t = i + f(t)$. Since $\forall t < e_i$, there are at most $i-1$ empty slots in the interval $[1, t]$, then $f(t) \geq t - (i - 1) > t - 1 \Longrightarrow t < i + f(t)$. Therefore, e_i is the minimum $t|t = i + f(t)$.

Theorem 2. *For a non-saturated system under any static-priority scheduling algorithm, there is*

$$e_{i+1} - e_i \leq e_a, \forall i \in N, a \in N^+ \tag{3}$$

Proof. To prove Theorem 2 it suffices to prove $\forall i, e_{i+a} \in [e_i + 1, e_i + e_a]$. Since e_i is empty, the workload generated in the interval $[1, e_i - 1]$ has been dealt with. According to Lemma 1 and conclusion A, the workload is:
$W = f(e_i + e_a) - f(e)i) = f(e_i) + f(e_a) - j - f(e_i) = f(e_a) - j$
From Theorem 1, there is $f(e_a) = e_a - a$, so $W = e_a - a - j$. The difference between the workload and the length of the interval is no less than a, so there are at least a empty slots in the interval. Therefore, $e_{i+a} \in [e_i + 1, e_i + e_a]$.

2.3 Discrete Static-Priority Schedulability Test

Theorem 3. *If task set τ_{n-1} is static-priority schedulable, when added a new low priority task τ_n, then τ is static-priority schedulable if and only if τ_{n-1} is non-saturated and $D_n \geq e_{c_n(n-a)}$.*

Proof. First we prove the sufficient condition: Task τ_n generates workload at instant $1, p_n + 1, 2p_n + 1$, and so on. To check the schedulability of τ it suffices to check if there are enough empty slots left to execute task τ_n in the interval $[kp_n + 1, kp_n + d_n]$. From Theorem 2, we know that $e_{c_n(n-1)} \geq e_{i+c_n(n-1)} - e_{i(n-1)}$. If there are empty slots in the first interval $[1, d_n]$ to execute task τ_n to completion, there must be enough empty slots in the following intervals $[p_n + 1, p_n + d_n], [2p_n + 1, 2p_n + d_n]$, and so on. As $d_n \geq e_{c_n(n-1)}$, there are enough empty slots in the interval $[1, d_n]$, to execute task τ_n to completion, so τ is schedulable. Next we prove the necessary condition. If τ is static-priority schedulable, then τ_{n-1} obviously is non-saturated and schedulable. As task τ_n's priority is the lowest, there must be $d_n \geq e_{c_n(n-1)}$. From the theorem above, we can easily get the following theorem.

Theorem 4. *A task set τ is static-priority schedulable if and only if for $i = 2, ..., n$,*

$$\sum_{j=1}^{i-1} \lceil \frac{c_i}{p_i} \rceil \leq 1 \text{ and } d_i \geq e_{c_i(i-1)} \tag{4}$$

Proof. This schedulability test includes two parts. The first one guarantees that τ_{i-1} is non-saturated and consequently it can incorporate another task, τ_i. The second part guarantees that there are sufficient slots to execute task τ_i to completion before its deadline. We refer to this test as empty-slot static-priority (ESSP) schedulability test in rest of the paper. For each task, we calculate and check if it's the c_i-th empty slot of the higher priority $i-1$ tasks. The maximum number of times to calculate free slot for each task is d_n . So the time complexity of ESSP is $O(n \times d_n)$. As d_n increases with n in the worst case, the time complexity depends not only on the dimension n of the problem but also the magnitude d_n of the data involved, so the time complexity of ESSP is pseudo-polynomial.

2.4 Comparisons with Classical Continuous Schedulability Tests

Audsley et al. [9] proposed a necessary and sufficient schedulability test for RM scheduling:

$$\forall i, 1 \leq i \leq n, WR_i \leq d_i \tag{5}$$

WR_i is the worst-case response time of τ_i and is given by the smallest $x \in N^+$ that satisfies the following recursive equation.

$$WR_i^0 = \sum_{j=1}^{i} c_j \tag{6}$$

$$WR_i^{l+1} = c_i + \sum_{j=1}^{i-1} \lceil \frac{WR_i^l}{p_j} \rceil c_j \tag{7}$$

The above procedure stops when the same value is found for two successive iterations of l, or when the deadline d_i is exceeded. Its time complexity is pseudo-polynomial[12]. For simplicity, we only compare ESSP with the technique presented in [9]. ESSP exhibit similar behavior as classical test, though they are obtained with different methods. Compared with the classical test, ESSP has many advantages. First, integer number is easy to be operated with than real number. Secondly, ESSP can easily solve many practical problems: Given τ, which is non-saturated and static priority schedulable, determine

- Whether $S(n+1)$ is also schedulable? (by adding a new low priority task)
- Up to what extent the execution time of a task in τ can be expanded while keeping the system feasible?
- What is the value of the execution time of the task that saturates the system?

Although the time complexity of ESSP is pseudo-polynomial i.e. $O(n \times d_n)$, normally d_n may not necessarily increase with n and the time performance of ESSP may not necessarily be so. Further more, we improve it to handle on-line admission control by the following methods:

(1) If τ_{n-1} is non-saturated, then $S(i)|i = 1, ..., n-2$ is also non-saturated.
(2) For $d_i \geq e_{c_i(i-1)}$ to hold, $e_{c_i(i-1)} = mint|t = c + i + \sum_{j=1}^{i-1} c_j \lceil \frac{t}{p_j} \rceil \geq \sum_{j=1}^{i} c_j$, we can check this inequality from the slot $\sum_{j=1}^{i} c_j$ instead of 1.
(3) If $c_i + \sum_{j=1}^{i-1} c_j \lceil \frac{t}{p_j} \rceil = t^* > t$, then the next slot to be checked is t^*.
(4) For each task, the number of ceiling calculations is not d_n, but increases from $\sum_{j=1}^{i} c_j$ to d_n.

Thus, the number of effective operations needed for ESSP is much less than $n \times d_n$. From the above analysis, we can see that ESSP is more convenient to be used and more efficient to tackle practical problems than classical continuous schedulability tests.

3 Enhanced Deasibility Analysis

3.1 Previous Results on Exact Test

Time Demand Approach. For validating feasibility problem, both necessary and sufficient condition (NSC) based tests are proposed in literature [4], [8], [9], [13], [14], [17]. Recently, authors in [3], [4] extended the work illustrated in [8] by proposing an exact feasibility test that reduces the number of scheduling points. Our results show that reducing the number of scheduling point does not necessarily means lowering the number of inequalities in actual.

To determine whether a task can meet all its deadlines, we compute the total demand for processor time by a task τ_i at time t, as

$$W_i(t) = c_i + \sum_{j=1}^{i-1} \lceil \frac{t}{p_j} \rceil c_j \tag{8}$$

A periodic task τ_i is feasible if we find some $t \in [0, d_i]$ satisfying

$$L_i = \min_{0<t\leq d_i} (W_i(t) \leq t) \tag{9}$$

As t is a continuous variable, there are infinite numbers of points to be tested. The entire task set τ is feasible iff

$$L = \max_{1\leq i\leq n} \{ \min_{0<t\leq d_i} \frac{W_i(t)}{t} \} \leq 1 \tag{10}$$

The first attempt to limit the infinite number of points in interval $t \in [0, t]$ is made by authors in [8]. The authors in [8] show that $W_i(t)$ is constant, except at

finite number of points when tasks are released, called rate monotonic scheduling points. Consequently, to determine whether τ_i is schedulable, we need to compute $W_i(t)$, only at multiples of $\tau_i \leq \tau_j, 1 \leq j \leq i$. specifically, let

$$S_i = \{ap_b | b = 1, ..., i; a = 1, ..., \lfloor d_i/p_b \rfloor\} \tag{11}$$

We conclude that an dividual task is feasible iff the following equation is true.

$$L_i = \min_{t \in S_i} \frac{W_i(t)}{t} \leq 1 \tag{12}$$

L_i is needed to be analyzed only at a finite number of points i.e. S_i. In rest of the paper, we represent this technique by TDA (Time Demand Approach). For any task τ_i , the number of elements in set S_i is of particular interest. Every element means testing an inequality constraint for finding schedulability of τ_i at run time. The number of elements in S_i becomes huge especially when ratio p_n/p_1 is large [3]. Clearly, an efficient technique would be the one which is capable of reducing the number of inequalities testing during online feasibility analysis.

Hyper-Planes Exact Test. To reduce scheduling points, E. Bini and G. C. Buttaazo provided a formulation, called Hyper-planes Exact Test (HET) recently in [3], [4] that reduces scheduling point for τ_i from set S_i to a reduced set $H_i(t)$. For any task τ_i, their test begins with p_i and expands its search space by

$$H_i(t) = H_{i-1}(\lfloor \frac{t}{p_i} \rfloor d_i) \cup H_{i-1}(t) \tag{13}$$

where H_0(t)={t}.

3.2 Deadline Monotonic Analysis Improved

The necessary and sufficient condition (NSC) for the schedulability of τ_1 is that $c_1 \leq p_1$. Using Equation 8, it is equally important for a low priority task to be tested in interval $[0, p_1]$ despite the fact that there is a fair chance of not fulfilling the cumulative workload constituted by current task because interference from the higher priority tasks (τ_{i+1} to τ_n) is also added to computation demands of current task τ_i. We avoid this redundancy of points by proposing that when the time demand for task τ_i is not fulfilled at some point t, then

$$\sum_{j=1}^{i} \lceil \frac{t}{d_j} \rceil c_j + c_{lower} > t \tag{14}$$

is always true, where c_{lower} is the execution demand of tasks whose priority is lower than τ_i, since $c_{lower} > 0 \ \forall c_{lower} \in [c_{i+1}, ...c_n]$.

To reduce the intended search space, we present some definitions and theorems, which eventually lead us to the main contribution of this section (Theorem 8).

Theorem 5. *-For any given periodic tasks set τ, scheduled by DM, task τ_i has a set of DM scheduling points $S_i : S_i \subseteq S_{i+1}$*

Proof. To prove this theorem, we must prove that if there exists a scheduling point $t \in S_i$, then $t \in S_{i+1}$ also holds good. For an arbitrary element of S_i, it follows that $t = ap_b$, where $a \in \{1, ..., \lfloor d_i/p_j \rfloor\}$, $j \in 1, ..., i$ and $b \in 1, ..., i$. According to DM, tasks priorities are inversely proportional to their deadlines, tasks τ_i has higher priority than τ_{i+1} as $d_{i+1} \geq d_i$, so it can be seen that $a \in \{1, ..., \lfloor d_{i+1}/p_j \rfloor\}$, $j \in 1, ..., i+1$ and $b \in 1, ..., i+1$, hence $t \in S_i$ and therefore $S_i \in S_{i+1}$ holds true.

Definition 1 (False Point). *DM scheduling point t, is said to be false point for any task τ_i if it satisfies the inequality constraint:$W_i(t) > t$.*

Theorem 6. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$ scheduled by DM, every false point for τ_i must also be a false point for τ_{i+1}.*

Proof. According to the definition of false point for task τ_i ;$W_i(t) > t$, thus for task τ_{i+1}

$$W_{i+1}(t) = \sum_{j=1}^{i+1} \lceil \frac{t}{d_i} \rceil c_j = \sum_{j=1}^{i} \lceil \frac{t}{d_i} \rceil c_j + \lceil \frac{t}{d_{i+1}} \rceil c_{i+1}$$
$$= W_i(t) + c_{i+1} > W_i(t) \tag{15}$$

As $c_{i+1} > 0$, therefore $W_{i+1}(t) > t$ and hence t is also a false point for τ_{i+1}.

Theorem 7. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$ scheduled by DM, every false point for τ_i is also a false point for task having priority lower than τ_i.*

Proof. This theorem can be directly deduced from Theorem 6.

Theorem 8. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$, τ_i can be feasibly scheduled for all tasks phasings using DM iff*

$$L_i = \min_{t \in Z_i} \frac{W_i(t)}{t} \leq 1 \tag{16}$$

where $Z_i = S_i - X_{i-1}$. By extension $X_0 = \emptyset$.

Proof. From Theorem 7, we known that, if t is a false point for τ_{i-1} then it must also be a false point for τ_i. Consequently to calculate L_i, we confine our search to a reduced set $Z_i \subseteq S_i$, by excluding all chained points.

The whole set τ is feasible iff

$$L = \max_{1 \leq i \leq n} L_i \leq 1 \tag{17}$$

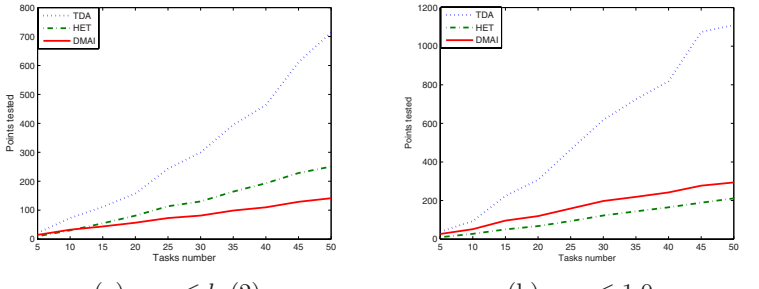
We replace S_i by Z_i, which immediately reflects the reduced complexity of our technique $Z_i \subseteq S_i$; allows us to search reduced number of inequalities for determining task feasibility. We use the term Deadline Monotonic Analysis Improved (DMAI) to refer our scheme hereafter in this paper. The effectiveness of DMAI becomes more prominent when it is applied to a large task set.

3.3 Experimental Results

In this section, we evaluate the performance of DMAI. In order to make a comparison with DMAI, both TDA and HET are also implemented. For our experiments, we generate random task periods in the range of $[10, 1000]$ with uniform distribution. Similarly, for corresponding task execution demands, random values are taken in the range of $[1, p_i]$, also with uniform distribution. Together, a series of periodic tasks is generated by varying the range from of 5 to 30 with increase of 5 tasks and average is taken after 200 iterations. Results are obtained under varied system utilization, however, due to space limitations, only two are shown for each experiment in the following. The work is compared, in light of two performance evaluation criteria i.e. the number of DM scheduling points used and, the number of inequalities tested , which is the intended contribution of this paper.

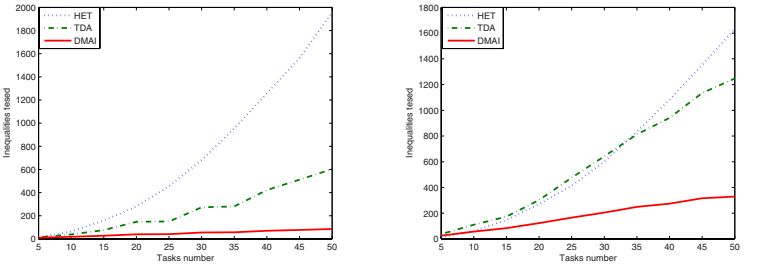
Number of Scheduling Points. In this section, we demonstrate the effectiveness of DMAI in terms of reducing the number of scheduling points. Fig. 1 provides a comparison among the tests by counting the number of points that are actually utilized before feasibility is concluded. We analyzed all tests by varying system utilization from $ln(2)$ to 1.0. When tasks set size increases i.e. $n \rightarrow \infty$ more inequalities are needed to be tested as p_n/p_1 is becoming larger. It is found that HET uses union operator and has implicit tendency for testing repeated values from intended search space obtained with Equation 13, we plot only unique values here. It can be seen that both HET and DMAI have effectively reduced the number of scheduling points as compared to TDA. This improvement is due to generally reduced sets $(H_{i-1}(p_i) \subseteq S_i)$ and $(Z_i \subseteq S_i)$ required for any task τ_i by HET and TDA respectively.

Number of Inequalities. In this important experiment, we extract the number of inequalities, which are being tested by all necessary and sufficient tests discussed earlier in Section 3.1 and 3.2 respectively. We observe that the number of inequalities tested is directly influenced by variation in system utilization. As



(a) $u_{tot} \leq ln(2)$ (b) $u_{tot} \leq 1.0$

Fig. 1. Effect of utilization on average number of scheduling points



(a) $u_{tot} \leq ln(2)$ (b) $u_{tot} \leq 1.0$

Fig. 2. Advantage of straight forward approaches over recursive technique

shown in Fig. 2, the little variation in behavior of HET under varied system utilization is mainly due of its dependency on task periods. In contrast, both DMAI and TDA equally exploit the execution demands of individual tasks and, are more inclined towards total system utilization. On one hand, for feasible task set having $u_{tot} \leq \ln(2)$ both DMAI and TDA converges early, while on the other, more points are needed to be analyzed for higher utilization when $u_{tot} \leq 1$. Clearly, the improved performance of DMAI is due to $(Z_i \subseteq S_i)$ which is a direct conclusion of Theorem 8 and, suppresses others in terms of testing reduced number of inequality constraint as shown in Fig. 2. It can be seen that, both TDA and HET use repeated points. This fact is witnessed by the difference in Fig. 1 and Fig. 2; there is a mismatch between number of scheduling points and inequalities tested for TDA and HET. As mentioned earlier, though HET reduces the number of scheduling points in theory, it looses its effectiveness at run time by scanning redundant points from the search space due to its recursive implementation. In our approach, there is a direct mapping between Fig. 1 and Fig. 2.

4 Conclusions and Future Work

We first introduced a schedulability analysis method using empty-slot for discrete scheduling. The method is then applied to static-priority scheduling and a new efficient discrete schedulability test is proposed. The analysis confirms that, empty-slot method is an efficient discrete schedulability analysis method and, can be generalized to allow more general task model, in particular when tasks may have deadlines larger than periods. The analysis can also be applied to the case in which task synchronization must be considered or when the priority ceiling protocol is used.

In addition to above, we have proposed DMAI, a state of the art solution for testing online feasibility of real time systems employing DM scheduling, which has much lower complexity than current feasibility tests. We evaluated the correctness and goodness of DMAI by means of mathematical proofs and

simulation results. The experiment aimed at testing the algorithm under different performance conditions and comparing its results with previous solutions. The experimental results obtained show the effectiveness of DMAI in providing an efficient online analysis for periodic tasks, and validated our theoretical results.

Acknowledgments

This work is jointly supported by COMSATS Institute of Information Technology under faculty development program, the National Natural Science Foundation of China (Grant Number: 60373053) and the research collaboration between the Chinese Academy of Sciences and the Royal Society of the United Kingdom (Grant Number: 20030389, 20032006).

References

1. Liu, J.W.S.: Real Time Systems. Prentice Hall, Englewood Cliffs (2000)
2. Krishna, C.M., Shin, K.G.: Real-Time Systems. McGrawHill, New York (1997)
3. Bini, E., Buttazzo, G.C.: The Space of Rate Monotonic Schedulability. In: Proceedings of the 23th IEEE Real-Time Systems Symposium, pp. 169–177 (2002)
4. Bini, E., Buttazzo, G.C.: Schedulability Analysis of Periodic Fixed Priority Systems. IEEE Transactions on Computers 53(11), 1462–1473 (2004)
5. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. of the ACM 20(1), 40–61 (1973)
6. Katcher, D.I., Arakawa, H., Strosnider, J.K.: Engineering and analysis of fixed priority schedulers. IEEE Trans. On Software Engineering 19(9), 920–934 (1993)
7. Baruah, S., Mok, A., Rosier, L.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems 2, 301–324 (1990)
8. Lehoczky, J.P., Sha, L., Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In: Proceedings of the IEEE Real-Time System Symposium, pp. 166–171 (1989)
9. Audsley, N.C.: Deadline monotonic scheduling, Report YCS.146, Depart. of Comput.Sci., University of York (1990)
10. Santos, J., Gastaminza, M.L., Orozco, J., Picardi, D., Alimenti, O.: Priorities and protocols in hard real-time LANs. Computer and Commun. 14(9), 507–514 (1991)
11. Santos, J., Orozco, J.: Rate monotonic scheduling in hard real-time systems. Information Processing Letters 48, 39–45 (1993)
12. Audsley, N.C., Burns, A., Richardson, M.F., Wellings, A.J.: Hard real-time scheduling: the deadline monotonic approach. In: Proceedings of 8th IEEE Workshop on Real-Time Operating Systems and Software, pp. 133–137 (1991)
13. Sjodin, M., Hansson, H.: Improved response-time analysis calculations. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 399–409 (1998)
14. Joseph, M., Pandya, P.: Finding response times in a real-time system. The Computer Journal 29(5), 390–395 (1986)

15. Leung, J.Y.T., Whitehead, J.: On the Complexity of Fixed-Priority Scheduling of Periodic. Real-Time Tasks Performance Evaluation 2, 237–250 (1982)
16. Kuo, T.-W., Mok, A.K.: Load Adjustment in Adaptive Real-Time Systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 160–171 (1991)
17. Manabe, Y., Aoyagi, S.: A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. Real-Time Systems 14(2), 171–181 (1998)
18. Tindell, K.W., Bums, A., Wellings, A.J.: An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Systems Journal 6, 133–151 (1994)

A Server-Side Pre-linking Mechanism for Updating Embedded Clients Dynamically

Bor-Yeh Shen[1] and Mei-Ling Chiang[2]

[1] Department of Computer Science,
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
byshen@cs.nctu.edu.tw
[2] Department of Information Management,
National Chi-Nan University, Puli, Taiwan, R.O.C.
joanna@ncnu.edu.tw

Abstract. To allow embedded operating systems to update their components on-the-fly, dynamic update mechanism is required for operating systems to be patched or added extra functionalities in without the need of rebooting the machines. However, embedded environments are usually resource-limited in terms of memory size, processing power, power consumption, and network bandwidth. Thus, dynamic update for embedded operating systems should be designed to make the best use of limited resources. In this paper, we have proposed a server-side pre-linking mechanism to make dynamic updates of embedded operating system efficiently. Applying this mechanism can reduce not only memory usage and CPU processing time for dynamic update, but also data transmission size for update components. Power consumption can be reduced as well. Performance evaluation shows that compared with the approach of Linux loadable kernel modules, the size of update components can be reduced about 14-35% and the overheads in embedded clients are minimal.

Keywords: Embedded System, Operating System, Dynamic Update, Modules, LyraOS.

1 Introduction

Dynamic update allows operating systems to update their components on-the-fly without rebooting the whole systems or stopping any system services. This opens up a wide range of opportunities: fixing bugs, upgrading services, improving algorithms, adding extra functionalities, runtime optimization, etc. Although many operating systems have already supported different kinds of mechanisms to extend their kernels, they usually do not aim at resource-limited environments. For instance, Linux uses a technique called loadable kernel modules (LKMs) [1]. By using this technique, Linux can load modules, such as device drivers, file systems, or system call to extend the kernel at run time. However, LKMs may take lots of overheads in embedded environments. Since embedded systems are usually resource limited, in order to keep the added overheads minimal while providing dynamic update in an embedded operating system, we propose the server-side pre-linking mechanism which is a

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 146–157, 2007.
IFIP International Federation for Information Processing 2007

client-server model similar to the server-side linking mechanism proposed in the operating system portal (OSP) framework [2]. Unlike the OSP framework, our server-side pre-linking mechanism does not have to negotiate between client and server to know the starting address of components on client hosts. Besides, we can perform component linking on the server-side before components are requested by clients. Thus, we can also save the components processing time on server hosts.

To demonstrate the feasibility of our proposed dynamic component update and component protection mechanisms, we have designed and implemented this mechanism in LyraOS [3] operating system. LyraOS is a research operating system designed for embedded systems, which uses component-oriented design in the system development. However, just like many embedded operating systems such as eCos [4] and MicroC/OS-II [5], LyraOS can be only statically configured at source-code level, so system cannot be updated or extended on-the-fly. Performance evaluation shows that the loader size under LyraOS is only about 1% and 7% as compared with the Linux loadable kernel module of the Linux 2.4 and the Linux 2.6. The component sizes under LyraOS are only about 14-35% of the Linux loadable kernel module. The component loading time also takes a few milliseconds. The component invocation time also adds only a few overheads caused by providing dynamic component exported interface and memory protection for un-trusted components.

Although our proposed dynamic component update and component protection mechanisms are implemented in LyraOS operating system, we believe that these experiences can serve as the reference for other component-based embedded operating systems that require an efficient and safe mechanism to dynamically update their components.

The rest of this paper is organized as follows. Section 2 introduces the LyraOS operating system. Section 3 introduces the related work. Section 4 details the design and implementation of our dynamic update mechanism. Section 5 shows our performance evaluation results and Section 6 concludes this paper.

2 LyraOS

LyraOS [3] is a component-based operating system which aims at serving as a research vehicle for operating systems and providing a set of well-designed and clear-interface system software components that are ready for Internet PC, hand-held PC, embedded systems, etc. It was implemented mostly in C++ and few assembly codes. It is designed to abstract the hardware resources of computer systems such that low-level machine dependent layer is clear cut from higher-level system semantics. Thus, it can be easily ported to different hardware architectures [6].

Figure 1 shows system architecture of LyraOS. Each system component is complete separate, self-contained, and highly modular. Components in LyraOS can be statically configured at source-code level. In addition to being light-weight system software, it is a time-sharing multi-threaded microkernel. Threads can be dynamically created and deleted, and thread priorities can be dynamically adjusted.

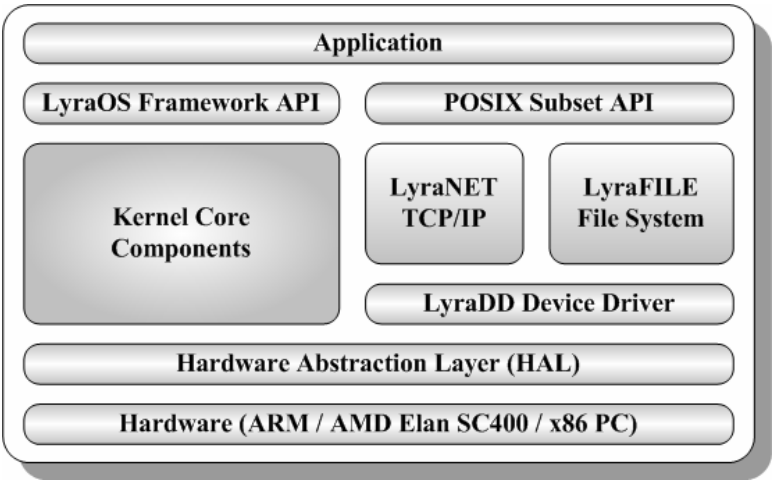


Fig. 1. LyraOS system architecture

3 Related Work

Linux Loadable Kernel Modules (LKMs) [1] are object files that contain codes to extend the running kernel. They are typically used to add support for new hardware, file systems, or for adding system calls. When the functionality provided by an LKM is no longer required, it can be unloaded. Linux uses this technology to extend its kernel at run time. However, Linux modules can be removed only when they are inactive. Another problem of LKMs is its space overheads. It needs additional kernel symbol table in client site and additional symbol table in loadable modules due to dynamic symbol linking. Dynamic symbol linking also takes lots of time during module loading. Our approach can eliminate these overheads. Besides, the LKMs require privilege permission to perform kernel modules loading. All of these modules are located in the kernel level and have the same permission as kernels. Thus, operating systems may crash because a vicious module is loaded in the kernel.

In operating system portal (OSP) [2], all the dynamically loadable modules are located on the server host. A user-level process is responsible for loading, linking and transmitting these modules to the clients. A kernel-level module manager is installed on the client to make the client kernel extensible. The server-side linking mechanism proposed in OSP is similar to our server-side pre-linking mechanism. Unlike the OSP framework, our server-side pre-linking mechanism can perform component linking on the server-side previously before components are requested by clients. We do not have to know the starting address of components on each client host because components will be relocated by client's relocation hardware. Thus, the components processing time on server hosts can also be saved since we do not need to link components for each request of clients.

SOS [7] is a dynamic operating system for mote-class sensor nodes. It uses dynamically loadable software modules to create a system supporting dynamic addition, modification, and removal of network services. The SOS kernel provides a

set of system services that are accessible to the modules through a jump table in the program memory. Furthermore, modules can also invoke functions in another module. The SOS kernel provides a dynamic function registration service for modules. Modules can register functions that they provide with the SOS kernel. The kernel stores information regarding the dynamic functions in a function control block (FCB) data structure. Processes can use a system call to subscribe a function.

4 Design and Implementation

According to the implementation of our component-based LyraOS operating system, an updatable unit may be a set of functions and global variables or an encapsulation of data members and methods. In both cases, software developers usually need to define a clear interface to the unit or make the unit inherit the interface from a virtual base class. Originally, other components should invoke the unit only through the static interface.

In this research, we implement our proposed dynamic component update mechanism in LyraOS. In our system, components are executable and linkable format (ELF) [8] files and components can be a set of functions, global variables, or C++ classes. Components do not have to use static interface. The only one thing that updatable components need to do is to register their exported methods to the component manager. Then, the external components will invoke these methods through the component manager.

Additionally, to make our system more flexible and safe, we separate all of the updatable components into two groups, trusted components and un-trusted components. In order to avoid un-trusted components causing our system crash, we divide the original LyraOS from single mode into user and kernel modes. Trusted components are located in kernel mode and can invoke system services directly. Un-trusted components are located in user mode and run in different protection domains enforced by hardware memory protection. Components permit system services invocation and communicate with other components only through the system call invocation when they are un-trusted.

In our system, all the dynamically updatable components are located on the server host and are pre-linked. A component server running on the server-side is responsible for loading and transmitting these pre-linked components to the embedded clients. A dynamic loader called LyraLD within the operating system kernel on the embedded client is responsible for downloading and installing pre-linked components. A component manager manages all of the components on the client-side and provides an interface for client-side applications to add, remove, or invoke components. For example, if an embedded client wants to add a new functionality, the embedded client will send a request through the component manager interface to the LyraLD. LyraLD will send a request to a remote component server to download a new component. The component server will respond with a pre-linked component which provides the functionality requested. Finally, the LyraLD will download and install this component directly without the need of linking or relocation.

4.1 Server-Side Pre-linking

Since embedded environments are usually resource-limited, we implement the server-side component pre-linking mechanism to keep the imposed overheads minimal while providing dynamic component update in an embedded operating system.

As mentioned above, components in our design and implementation have been linked on the server-side before components are requested by embedded clients. These components are linked according to their types (i.e., trusted or un-trusted) and symbol tables of embedded clients. The trusted component will be linked with the kernel symbol of the embedded client while the un-trusted one will be linked with the user library symbol table of the client. Especially, we do not need to know where the component will reside in the client-side memory (i.e., the starting address of the component). All of the updatable components will be linked at the same starting virtual address through the linker script we defined. Then the components will be relocated by the client-side relocation hardware that we will describe later. Because the updatable components can be linked in a prior time, we can save the component processing time on the server-side when components are requested.

Figure 2 shows our server-side pre-linking architecture. In our system, there is a component server on the server-side responsible for handling client requests. The component server on the server host receives request from the embedded client kernels and performs tasks as follows. If a pre-linked component is found in the pre-linked component storage, the component server will send the pre-linked component to the embedded clients immediately. Otherwise, the component server will link the components on demand.

The merits of our approach can be summarized as follows. The server-side component pre-linking can save not only the memory and the disk storage on embedded clients but also the component transmitting time because we downgrade the sizes of updatable components. Besides, it eliminates the need for clients to perform dynamic linking. Furthermore, the power consumption of embedded devices can be also decreased.

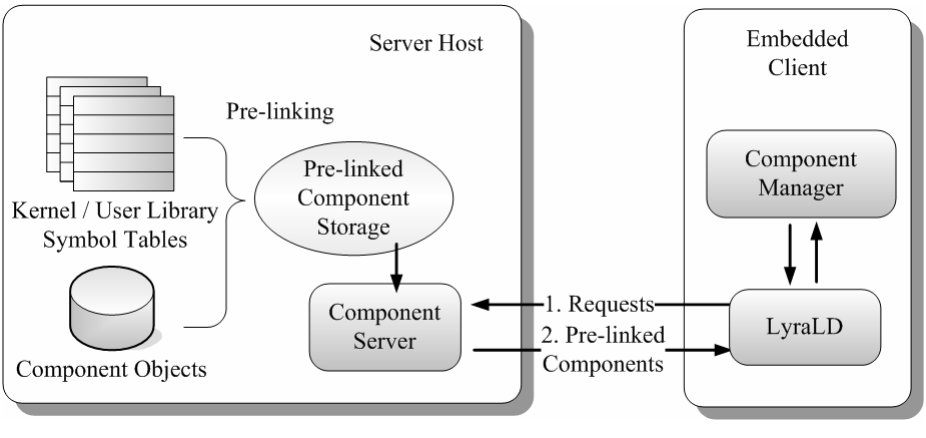


Fig. 2. Server-side pre-linking architecture

4.2 Client-Side Loading

We develop a dynamic component loader called LyraLD and a component manager in LyraOS to perform dynamic component loading and component management. Both the LyraLD and the component manager reside in the kernel level. Currently, the LyraLD use the trivial file transfer protocol (TFTP) [9] to download pre-linked components from the component server.

Figure 3 shows the steps of client-side component loading and installing. First, the component manager receives an invocation request to load a new component. Second, the component manager checks whether the component exists or not. If the component is not found in the client-side, the component manager will call LyraLD to send a request to a remote component server to download this component. Third, the LyraLD downloads a pre-linked component image returned from the remote component server to the client-side memory. Fourth, after the LyraLD reads the pre-linked component image's header from the memory address where the image is located, the LyraLD will verify the pre-linked component image, initialize component environments, and move each section of the image to the virtual address that the ELF header specified. Finally, the LyraLD will jump to the entry address of the component image to execute the component's initialization function that registers the component exported methods to the component manager.

Table 1 shows our component manager API. Components can be added, removed, updated, and invoked through these APIs. In order to provide dynamic component exported interface, the register method can register the component exported methods to the component method vector table when a component is loaded. As a component is downloaded and loaded into memory, the LyraLD will get the entry point address from the header of the component and then jump to this address to perform the registration of component's methods.

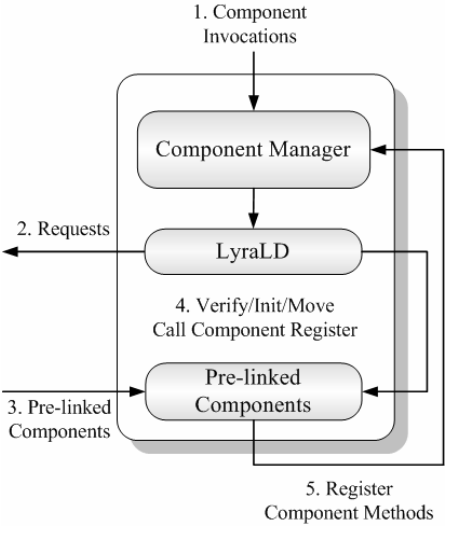


Fig. 3. Client-side loading

```
function entry(Opt, Addr)

switch(Opt)
begin

  case REGISTER:
    CM::Register(1, functionA);
    CM::Register(2, functionB);
    // ......
    break;

  case IMPORT:
     // convert and import
     // component states from Addr
     break;

  case EXPORT:
    // export states of this component
    // return address of export states
    break;

  end

end function
```

Fig. 4. Component interface

Table 1. Component manager API

Methods	Descriptions
CM::Add(name, ver)	The CM::Add() method adds a new component name with version ver from a remote component server and returns component's ID.
CM::GetCID(name, ver)	The CM::GetID() method returns component ID of component name (version ver).
CM::Invoke(cid, mid, arg)	The CM::Invoke() method invokes a method mid of a component cid and passes arguments arg through the component manager.
CM::Register (mid, fptr)	The CM::Register() method registers method's ID mid and its address fptr to the component manager.
CM::Remove(cid)	The CM::Remove() method removes component whose ID is cid.
CM::Update(old, new)	The CM::Update() method updates a component from component ID old to component ID new.

Figure 4 shows our component interface. This function would be implemented by developers and will be linked as the entry point of updatable components during server-side pre-linking. Every updatable component has to implement this interface to register its methods and transfer its states. As the component jumps to the entry point, the component will invoke the register method to register its exported methods to the component manager. Therefore, other components can invoke these methods through the component manager without using static component interface. When we want to remove a component, all of the component information including current states of the component and function pointers of the component exported methods should be removed. Methods in Table 1 provide a component communication interface. In our system, components must communicate with each other through the component manager. This is because we provide dynamic component exported interface in our system and these interfaces of components are managed by the component manager.

4.3 Component Relocation

The component relocation in our system implementation takes advantage of the ARM fast context switch extension (FCSE) mechanism [10]. The FCSE is an extension in the ARM MMU. It modifies the behavior of an ARM memory translation. This modification allows our components to have their own first 32MB address space. Thus, we make each component have its own address space and relocate in the first 32MB of memory. As shown in Figure 5, there is only one page table in our system. The 4GB virtual address space is divided into 128 blocks, each of size 32MB. Each block can contain a component which has been compiled to use the address ranging from 0x00000000 to 0x01FFFFFF. Each block is identified with a 7-bit PID (Process ID) register. Through the FCSE mechanism, we can switch between components' address spaces by changing the PID register and do not have to flush caches and TLBs. The same functionality can be achieved by other architectures which provide paging and an address space identifier (ASID) found on many RISC processors such as Alpha, MIPS, PA-RISC, and SPARC.

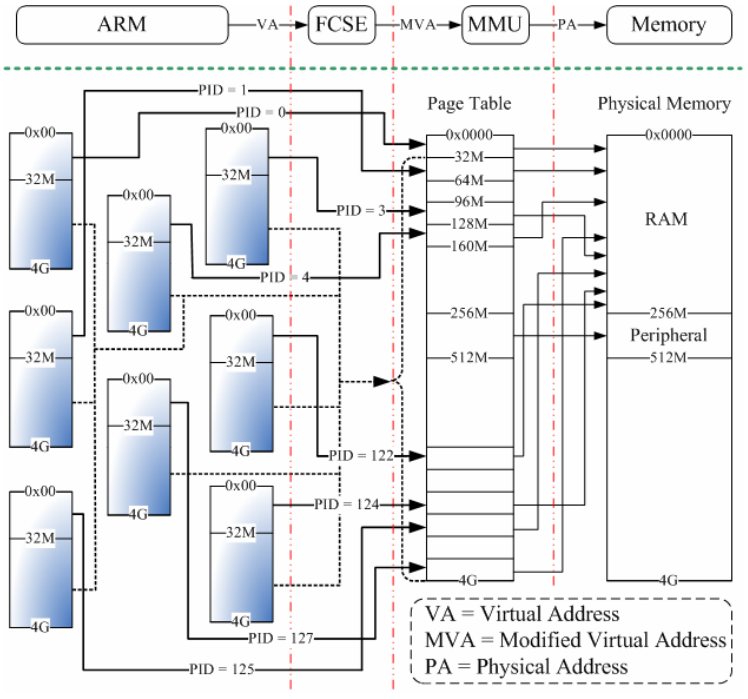


Fig. 5. Relocation by FCSE mechanism

However, there is a critical problem about communication among components. Since every component has an address space itself, we cannot pass component a pointer type argument that is pointed to another address space. Due to this reason, we use a shared memory mechanism to resolve this problem. A memory region which is greater than 32MB is reserved to store data that the argument points to. This is due to the fact that if an address is greater than 32MB, it will not be modified by FCSE. This means that the address space of components from 32MB to 4GB is shared. This also allows components to directly access our kernel core or user libraries which are out of the first 32MB without changing PID or page tables.

4.4 Component Protection

The ARM architecture provides a domain mechanism [10] to make different protection domains running with the same page table. We use this mechanism to make each un-trusted component have its own protection domain. A domain access control register (DACR) can be used to control the access permissions of components. Currently, each un-trusted component's first descriptors of the page table in our system are associated with one of the sixteen domains and its own DACR status. The DACR describes the status of the current component with respect to each domain. Since trusted components are the components that have been verified, they can use the same protection domain as kernel core and run in the kernel mode. However, although un-trusted components run in the user mode, they may also have vicious

codes to affect other un-trusted components. Therefore, they should locate in different protection domains and use the client access types. Thus, we can avoid the situation that the current un-trusted components will be affected as we load a new un-trusted component into our system. Although ARM only supports 16 domains which may be less than the number of un-trusted components concurrently in our system, we can apply other approaches such as domain recycling [11,12] to resolve this problem.

5 Performance

This section presents the performance evaluation of the proposed dynamic component update mechanism implemented in LyraOS. We compare the space overheads of our architecture with the Linux loadable kernel modules. The experimental environment consists of a client and a server host that are connected via a 100 Mbits/sec Ethernet. The server host is a Pentium 4 3.2GHz PC with 1GB RAM, running Linux 2.4.26. The client host is an ARM Integrator/CP920T development board with 128 MB RAM, running LyraOS 2.1.12.

5.1 Comparison of Space Overheads

Table 2 shows the loader sizes of the client kernel. We compare the size of LyraLD to the sizes of Linux LKMs linker/loader under kernel version both 2.4 and 2.6. The fundamental difference between Linux 2.4 and Linux 2.6 is the relocation and linking of kernel modules are done in the user level or kernel level. Loadable kernel modules in Linux are ELF object files which can be loaded by a user program called **insmod**. In Linux 2.4, **insmod** does all the work of linking Linux kernel module to the running kernel. While the linking is done, it generates a binary image and then passes it to the kernel. In Linux 2.6, the **insmod** is a trivial program that only passes ELF objects directly to the kernel, and then the kernel does the linking and relocation. In Table 2, the Linux 2.4 module linker/loader shows the static and dynamic size of the **insmod** program on Linux 2.4.26. The Linux 2.6 module linker and module loader were measured from the object files of **kernel/module.c** and **kernel/kmod.c** in the Linux 2.6.19 source tree. All symbols in these programs and object files have already been stripped. From the table we can see that, the size of LyraLD is less than 1% of the module linker/loader under Linux 2.4 and is about 7% of the module linker/loader under Linux 2.6.

Table 2. Sizes of loaders

Loader	Object Code Size	
Linux 2.4 module linker/loader	618,712 bytes 133,140 bytes	(static linked) (dynamic linked)
Linux 2.6 module linker Linux 2.6 module loader	14,088 bytes 2,060 bytes	(kernel/module.o) (kernel/kmod.o)
LyraLD (LyraOS loader)	1,140 bytes	

Table 3. Kernel and symbol sizes

Items	Size
LyraOS kernel image	35,752 bytes
LyraOS kernel symbol table	24,850 bytes
Linux 2.6.19 kernel image (vmlinux)	1,219,296 bytes
Linux 2.6.19 kernel image (zImage)	1,181,932 bytes
Linux 2.6.19 symbol table	505,487 bytes

Table 4. Component overheads

Components	Linux	LyraOS	Ratio
Task scheduler	4280 bytes	604 bytes	(14%)
Interrupt handler	7544 bytes	1612 bytes	(21%)
Timer driver	4424 bytes	992 bytes	(22%)
Serial driver	5640 bytes	1324 bytes	(23%)
Signal	7768 bytes	2736 bytes	(35%)
Semaphore	4116 bytes	632 bytes	(15%)

Table 5. Component loading and pre-linking time

Components	Client-side Loading	Server-side Pre-linking
Task scheduler	20.31ms	26ms
Interrupt handler	31.17ms	25ms
Timer driver	39.86ms	35ms
Serial driver	30.44ms	32ms
Signal	22.04ms	29ms
Semaphore	20.32ms	28ms

In addition, to perform the dynamic linking, Linux also requires the kernel symbol table to be stored on the client host. The size of the symbol table is dependent on the client-side kernel. From Table 3 we can see that, the kernel symbol table of LyraOS is about 24 Kbytes in our system. It occupies almost 70% of the LyraOS kernel size. The kernel symbol table of Linux 2.6.19 in our system is about 494 Kbytes. It occupies about 40% of the Linux kernel size.

Table 4 shows the component space overheads of the task scheduler, the interrupt handler, the timer driver, the serial driver, the signal, and the semaphore component in LyraOS and Linux. In this table, the column of Linux shows the sizes of ELF object files of these components under the Linux LKMs approach. The column of LyraOS shows the size of pre-linked images of these components under the LyraOS server-side pre-linking approach. The numbers in parentheses are the ratios of component overheads under LyraOS to those under the Linux LKMs. From the table we can see that, the sizes of components under the LyraOS approach are only about 14-35% of the sizes under the Linux LKMs approach. This is because the LKMs mechanism

contains more overheads for dynamic linking, such as symbol tables, string tables, relocation data, and other data structures.

5.2 Component Loading/Pre-linking Time

Table 5 shows the component client-side loading and server-side pre-linking time of those components we described above. The component loading only takes a few milliseconds. From the Table 4 and Table 5, we can see that the component loading time is not related to the sizes of the components. This is because the loader has to initialize some of the ELF sections. For example, BSS is a memory section where un-initialized C/C++ variables are stored. If there is a BSS section in a component, it needs to clear to zero while the component is loaded into memory. Besides, from the server-side pre-linking time we can see that embedded clients save lots of linking time when new components are loaded since the linking has been done previously on the server. We should know that the server-side pre-linking runs on a Pentium4 3.2GHz machine, and the frequency of ARM920T processors is only about 200MHz. It could cause large overheads if the component linking is performed on the embedded clients.

5.3 Component Invocation Time

In Figure 6, we invoke a method of each component we described above. “Direct Invocation” measures the invocation time of the direct component invocation. That is, direct component invocation invokes methods directly without calling the component manager and system calls. “Trusted Component” measures the invocation time of the trusted component invocation through the component manager. “Un-trusted Component” measures the invocation time of the un-trusted component invocation through the system call and the component manager. From the figure we can see that, it only adds a few overheads by providing dynamic component exported interface and memory protection for un-trusted components. Besides, relocation by hardware also keeps the overhead of switching between components’ address space minimal.

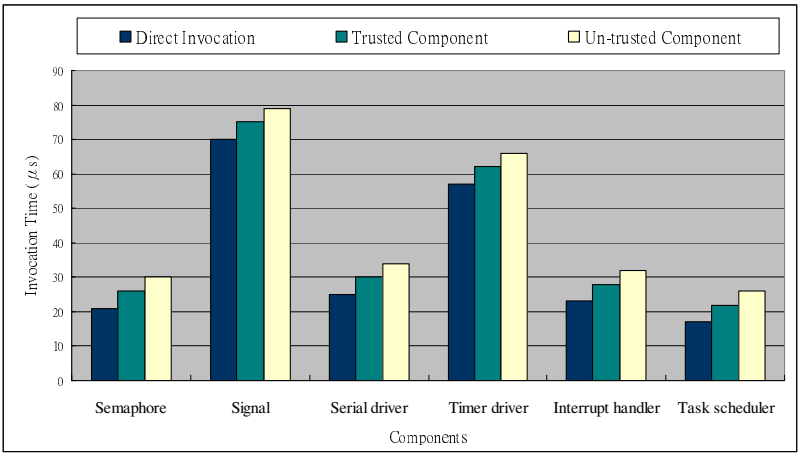


Fig. 6. Component invocation time

6 Conclusion

In this paper, we have proposed a server-side pre-linking mechanism to make an embedded operating system more extensible. The embedded operating system can be updated dynamically without the need of dynamic linker and symbol table. Besides, the dynamic component exported interface can make component developers change component exported interfaces easily. Furthermore, to make the system more flexible, components are separated into trusted components and un-trusted components, which run in different protection domains enforced by hardware memory protection.

After applying the proposed mechanisms in our target embedded operating system, LyraOS, the performance evaluation shows that the loader size under LyraOS is only about 1% and 7% as compared with the Linux loadable kernel module of the Linux 2.4 and the Linux 2.6. The component overhead under LyraOS is only about 14-35% of the Linux loadable kernel module. The component loading time also takes only a few milliseconds. The component invocation time also adds a few overhead caused by providing dynamic component exported interface and memory protection for un-trusted components.

References

1. Linux Loadable Kernel Module HOWTO, http://www.tldp.org/HOWTO/Module-HOWTO/
2. Chang, D.-W., Chang, R.-C.: OS Protal: an economic approach for making an embedded kernel extensible. Journal of Systems and Software 67(1), 19–30 (2003)
3. LyraOS, http://163.22.34.199/joannaResearch/LyraOS/index.htm
4. eCos, http://sources.redhat.com/ecos/
5. MicroC/OS-II, http://www.ucos-ii.com/
6. Cheng, Z.Y., Chiang, M.L., Chang, R.C.: A Component Based Operating System for Resource Limited Embedded Devices. In: IEEE International Symposium on Consumer Electronics, Hong Kong (2000)
7. Han, C.-C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A Dynamic Operating System for Sensor Nodes. In: Proceedings of the 3rd International Conference on Mobile Systems, Applications and, Services, Seattle, WA, USA (2005)
8. Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, Version 1.2, http://www.x86.org/ftp/manuals/tools/elf.pdf
9. The TFTP Protocol (Revision 2), http://www.ietf.org/rfc/rfc1350.txt
10. Seal, D.: ARM Architecture Reference Manual, 2nd edn. Addison-Wesley, Reading (2001)
11. Wiggins, A., Heiser, G.: Fast Address-Space Switching on the StrongARM SA-1100 Processor. In: Proceedings of the 5th Australasian Computer Architecture Conference, Canberra, Australia (2000)
12. Wiggins, A., Tuch, H., Uhlig, V., Heiser, G.: Implementation of Fast Address-Space Switching and TLB Sharing on the StrongARM Processor. In: Proceedings of the 8th Asia-Pacific Computer Systems Architecture Conference, Aizu-Wakamatsu City, Japan (2003)

Real-Time Scheduling Under Time-Interval Constraints*

Fábio Rodrigues de la Rocha and Rômulo Silva de Oliveira

Graduate Program in Electrical Engineering
Federal University of Santa Catarina, Florianópolis, Brazil
{frr,romulo}@das.ufsc.br

Abstract. This paper presents a new task model where jobs are divided into segments `A`, `B` and `C`. Segment `B` has a specific time-interval where it should execute to fulfill some application constraints. We consider the execution of `B` as valid if performed inside that time-interval, otherwise, its contribution may be valueless to its task. We adapt some scheduling approaches from the literature and present a feasibility test in terms of expected QoS for our scheduling problem.

Keywords: Real-Time, Scheduling, Task Model, Time-Interval, QoS.

1 Introduction

In most scheduling problems the main interest is to ensure that the task deadline will not be missed. In these cases, the deadlines as well as the periods are embedded constraints in the problem definition with a direct correspondence in physical world. As the years pass by, new task models, scheduling algorithms and feasibility tests were created to expand the algorithmic knowledge available to both the researcher and the system developer. The DM [1], RM [2] and EDF [2] are some well-known algorithms to assign priorities frequently using the periodic task model. In this model, every task τ_i has a fixed period T_i, a worst-case execution time W_i and a relative deadline D_i [2] (in many cases $D_i = T_i$).

The deadline for a task τ_i is a time-limit to τ_i finish its computation. As long as the computation ended before the deadline, the result is timely correct and its finishing time is unimportant. Although many applications can be represented by that model, there are some situations in which tasks have special constraints unachieved by periodic task models and mainly by the concept of deadline [3]. In some application, tasks demand part of their code to run inside a specific time-interval. The time-interval is a time window inside which the execution must take place and its start time is usually computed online. We present some real-world examples where we can have a time-interval.

① In embedded systems, tasks can send messages using an embedded protocol controller such as i^2c, RS232, USB, CAN. In low cost microcontrollers, during the data transmission the CPU is kept busy moving data from memory to controller

* This research was supported by CNPq and CAPES.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 158–169, 2007.

port and waiting for the response/end of transmission. Therefore, both the tasks and the data transmission must be scheduled. Moreover, the data transmission is non-preemptive and sometimes has to be delivered inside a time-interval.

② In ad hoc mobile systems the transmission of packets can be subject to route constraints. Assume that at time t_1 a source device S has a packet to transmit to a destination X. There is a chance that the radio signal from device S cannot reach the destination (there is no feasible route between the source and the destination X at time t_1). In these cases, the packet could be dropped due to a limited buffer space in S. A better solution would schedule the packet transmission to a future time t_2 when there will be a feasible route. However, as the routes dynamically change, time t_2 is only known during run-time.

None of these use cases show a time-limit as the main concern. In fact, they present examples where computations must take place inside a time-interval and maybe inside an inner ideal time-interval where the execution results in the highest benefit. In such cases, the concept of deadline as well as a periodic model are inappropriate to model applications. Unfortunately, by the lack of theoretical study and suitable task models, applications are implemented with conventional schedulers leading to a lack of predictability.

We present a new task model to fulfill a gap in the real-time literature. In our task model, tasks may request that part of their computations execute inside a time-interval to fulfill applications constraints. The start of this time-interval is adjusted on-line and the computations performed before or after the time-interval may be useless for applications purposes. Inside the time-interval, there is an ideal time-interval where the execution results the highest benefit. The benefit decreases before and after the ideal time-interval according to time-utility functions. We integrate and modify some scheduling approaches from the real-time literature in order to obtain a solution for our scheduling problem. As a result, we created an offline feasibility test which provides an accept/reject answer and a minimum and maximum expected benefit for tasks.

Related Work

A classic approach to obtain a precise-time execution is achieved through a time-driven scheduler [4]. However, that approach does not work in face of dynamic changes in task properties [5] such as the start time of our time-interval. The subject of value-based scheduling is studied in many papers. In [6] the authors give an overview of value based-scheduling, their effectiveness to represent adaptive systems and present a framework for value-based scheduling. A study about scheduling in overload situations is presented in [7]. In that paper, tasks have a deadline and also a quality metric. The scheduler performance is evaluated by the cumulative values of all tasks completed by their deadlines and the paper shows that in overload situations scheduling tasks by its value results in better performance. In [8] a task model is composed by tasks with a mandatory and an optional part that increases the benefit as the task executes. However, it is acceptable to execute only the mandatory parts. Also, the optional part is unrelated to a specific time to execute. The Time Utility Function model in which there is a function to assign a benefit obtained according to the task's completion

time is presented in [9] and an extension is presented in [10] with the concept of Joint Utility Function in which an activity utility is specified in terms of other activity's completion time. Differently from these work about task rewarding, our reward criteria is connected to the specific moment the job executes instead of its finish time or the amount of computation performed. An on-line interval scheduling problem in which a set of time-intervals are presented to a scheduling algorithm is presented in [11]. The time-intervals are non-preemptive, have start and end times and cannot be scheduled either early or late. As a future work, the authors discuss a slightly different problem in which the release times are more general and a task could request a given time-interval to be delivered in a determined time as in our problem. The time-interval would be allowed to slide slightly to accommodate other tasks. In the Just in Time Scheduling an earlier execution of a task is as bad as a later execution. The scheduling algorithm tries to minimize the earliness and tardiness (E/T). An overview of E/T problems and a polynomial algorithm for problems with non-execution penalties is presented in [12]. Similarly as in our model, in [13] tasks can adjust during run-time, when the next activation should execute to fulfill some applications constraints. A previous version of our work was presented in [14].

Organization
This paper is organized as follows. Section 2 presents the time-interval model. Section 3 presents a scheduling approach and section 4 presents some experimental evaluations. Section 5 presents the conclusions and future work.

2 Time-Interval Model

We propose a task model in which a task set τ is composed by tasks τ_i, $i \in \{1 \ldots n\}$. Tasks τ_i are described by a worst-case execution time W_i, period T_i, a deadline D_i and $T_i = D_i$. Each τ_i consists of an infinite series of jobs $\{\tau_{i1}, \ldots, \tau_{ij}, \ldots\}$, the j^{th} such job τ_{ij} is ready at time $(j-1) \cdot T_i, j \geq 1$ and must be completed by time $(j-1) \cdot T_i + D_i$ or a timing fault will occur. We define by **segment** a sequential group of instructions inside τ_i (as shown in Fig. 1). Task τ_i is composed by three segments named A_i, B_i and C_i. We denote the first index of a segment as the task and the second index as the job, thus the first job of segment A_i is named A_{i1}, the second job is A_{i2} and so on for all segments. The worst-case execution time of A_i is W_{A_i}, of B_i is W_{B_i} and of C_i is W_{C_i}. The sum of the worst-case execution time of all segments is equal to the worst-case execution time of task τ_i ($W_{A_i}+W_{B_i}+W_{C_i}=W_i$). We assume that there is a precedence relation among segments $A_i \prec B_i \prec C_i$.

The execution of segments A_i, B_i and C_i is subject to the deadline of task τ_i, D_i which in this sense is an end-to-end deadline. Segment A_i is responsible for performing its computations and it may require or not the execution of segment B_i. Hence, the arrival time of segment B_i is determined on-line by segment A_i. In case segment B_i is required to execute, segment C_i (which is a **housekeeping code**) will also execute. Therefore, even though the execution of segment A_i is periodic with period T_i, segments B_i and C_i are sporadic. In case neither B_i

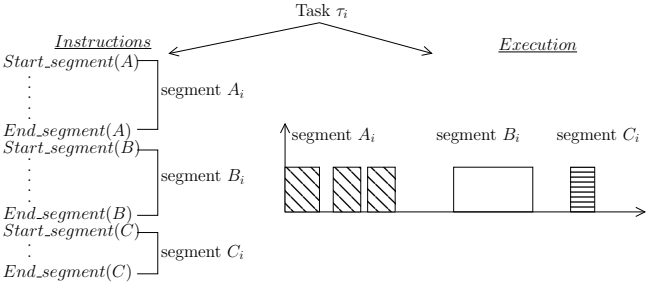


Fig. 1. Task τ_i with Segments

nor C_i are required to execute, segment A_i can execute up to the deadline D_i. Otherwise, as soon as segment B_i concludes, segment C_i is released to run. As we consider an uniprocessor system, segments cannot overlap in time.

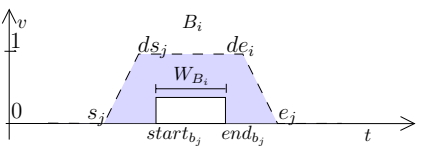
2.1 QoS Metric

The execution of segment B_{ij} is also subject to a **time-interval** $[s_{i,j}, e_{i,j}]$ which is defined by segment A_{ij} during run-time and can change for each job $\tau_{i,j}$, i.e: segment B_{ij} must execute inside this time-interval to generate a positive benefit. The length of $[s_{i,j}, e_{i,j}]$ is constant and named ρ_i. Inside the time-interval $[s_{i,j}, e_{i,j}]$, there is an **ideal time-interval** $[ds_{i,j}, de_{i,j}]$ with constant length named ψ_i where the execution of segment B_{ij} results in the highest benefit to τ_i ($W_{B_i} \leq \psi_i \leq \rho_i$).

The functions in Fig. 2 and 3 were made to represent ordinary applications requirements and so they also represent different real-time constraints. Figure 3 represents a **strict** benefit where the segment B_i must execute inside the ideal time-interval $[ds_j, de_j]$, otherwise the benefit is $-\infty$, meaning a catastrophic consequence. Figure 2 represents a **cumulative** benefit where the benefit decreases from maximum (inside the ideal time-interval) to zero at the time-interval limits. The choice of a particular function for a task is an application constraint which also determines the values of $s_{i,j}$,$e_{i,j}$,$ds_{i,j}$ and $de_{i,j}$.

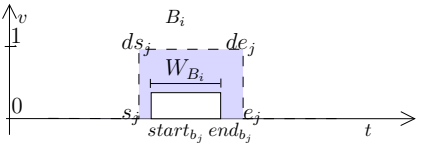
In those figures, the y axis represents the benefit v and the x axis is the activate time t. The segment B_{ij} is presented as running with its worst-case execution time (W_{B_i}), starting at $start_{bj}$ and ending at end_{bj}. The benefit $v(t)$ as a function of time is given by the equations in each figure. Equation 1 shows the QoS value as the cumulative benefit by the execution of segment B_{ij} inside the time-interval. The range in $[0\%, 100\%]$ represents the percentage of the maximum benefit. The maximum benefit is only achieved when B_i runs all its code inside the **ideal time-interval** $[ds_{i,j}, de_{i,j}]$. The goal is to maximize the QoS for each job B_i. In case B_i is not required there is no QoS value to account.

$$\mathrm{QoS}(B_{i,j}, start_{B_{i,j}}, end_{B_{i,j}}) = \frac{\int_{start_{B_{i,j}}}^{end_{B_{i,j}}} v\,(t)\,dt}{end_{B_{i,j}} - start_{B_{i,j}}} \cdot 100 \qquad (1)$$



$$v(t) = \begin{cases} 0, & t < s_j \text{ or } t > e_j \\ 1, & ds_j \leq t \leq de_j \\ 1 - \left(\frac{ds_j - t}{ds_j - s_j}\right), & s_j \leq t < ds_j \\ 1 - \left(\frac{t - de_j}{e_j - de_j}\right), & de_j < t \leq e_j \end{cases}$$

Fig. 2. Cumulative Benefit



$$v(t) = \begin{cases} -\infty, & t < ds_j \text{ or } t > de_j \\ 1, & ds_j \leq t \leq de_j \\ & e_j = de_j \end{cases}$$

$$s_j = ds_j$$

Fig. 3. Strict Benefit

Besides its time constraints, the time-interval problem may also present constraints regarding exclusive access during segment B execution inside the time-interval. The nature of the resource under segment B control imposes access constraints to ensure the consistence during resource's operation. We assume that during the execution inside the ideal time-interval the CPU is busy controlling the resources. In this paper we consider segment B as non-preemptive, which fulfills the access constraints. Therefore, the execution of B_i from task τ_i cannot be preempted by other task τ_j. The start of B_i may be postponed, however, once started it cannot be disturbed.

3 Scheduling Approach

For implementation purposes, it is natural to represent the segments in our model as a set of subtasks. Therefore, we map all the segments of task τ_i into subtasks keeping the same names A_i, B_i and C_i. Subtasks A_i and C_i are scheduled using a preemptive EDF scheduler by its capacity to exploit full processor bandwidth. A distinction is made to subtask B_i, which is non-preemptive and scheduled in a fixed priority fashion. In a broad sense, when a task τ_i is divided into subtasks each subtask possesses its own deadline and the last subtask has to respect the task's deadline, in this case an end-to-end deadline D_i. Even though the task τ_i has a deadline equal to period ($D_i = T_i$), the subtasks require inner deadlines, which must be assigned using a deadline partition rule.

The first challenge in our scheduling approach is the **deadline partition** among subtasks. The time-interval definition states that segment B_i must start adjusted by A_i. The minimum time to release B_i is a problem constraint and this value may be used as a deadline for the previous segment. Therefore, in the time-interval problem the deadline partition rule is determined using the problem constraints. We assume a lower bound and an upper bound for the release time of segment B_i $[Bmin_i, Bmax_i]$ and set the deadline $D_{A_i} = Bmin_i$ and $D_{B_i} = Bmax_i + \rho_{B_i}$ as in Fig. 4. The time interval in which the segment B_i can be active is $[Bmin_i, D_{B_i}]$ and named **time-window**. The second challenge is the **release time** of subtasks. It is necessary to enforce in the schedulability test

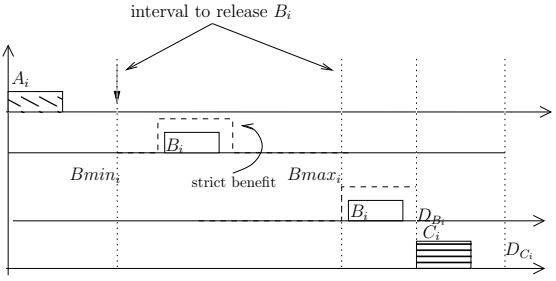


Fig. 4. Limits to Release B_i

that a subtask must be release only after a predetermined time. In this situation, we apply offsets [15] to control the release of subtasks and an offset oriented test to fulfill the requirement. The third challenge is the **non-preemptive** aspect of subtask B_i. The feasibility test must ensure that the execution of the non-preemptive subtask cannot be preempted by any other subtask. In this aspect, in [16] it is proposed a schedulability test to verify the schedulability of tasks in the presence of non-preemptive tasks with the higher priorities.

3.1 Offline Feasibility Test

We verify the schedulability of a task set τ splitting the problem in two parts. In the first part we test the schedulability of subtasks A_i and C_i in face of non-preemptive interferences by subtasks B_i. A negative answer (reject) means that all task set is unfeasible. In contrast, a positive answer (accept) means that all subtasks A_i and C_i will finish up to their deadlines even though suffering interference by non-preemptive subtasks. The second part applies a test based on response-time to verify if the strict subtasks B_i are schedulable. A negative answer means all task set is unfeasible. Otherwise, all strict subtasks B_i will execute inside their ideal time-intervals and receive the maximum QoS value. Using the same response-time test, we determine the minimum and maximum QoS value which can be achieved by all cumulative subtasks B_i.

Feasibility Test for Subtasks *A* and *C*. The feasibility of test of subtasks A and C is performed using the processor demand approach [17]. The processor demand of a task in a time-interval $[t_1, t_2]$ is the cumulative time necessary to process all k task instances which were released and must be finished inside this time-interval. We assume $g_i(t_1, t_2)$ as the processing time of τ_i.

The processor demand approach works by observing that the amount of processing time requested in $[t_1, t_2]$ must be less than or equal to the length of the time-interval. Therefore, $\forall t_1, t_2 \; g(t_1, t_2) \leq (t_2 - t_1)$.

Lets assume a function $\eta_i(t_1, t_2)$ as the number of jobs of task τ_i with release and deadline inside $[t_1, t_2]$. $\eta_i(t_1, t_2) = max\{0, \lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \rfloor - \lceil \frac{t_1 - \Phi_i}{T_i} \rceil\}$. Where T_i is the period of task i, Φ_i is the offset (phase) of task i and D_i is the deadline of task i. In Fig. 5 the only jobs accounted by η_i are jobs $\tau_{i,2}$ and $\tau_{i,3}$. Job $\tau_{i,1}$ has a release time before t_1 and $\tau_{i,4}$ has a deadline after t_2.

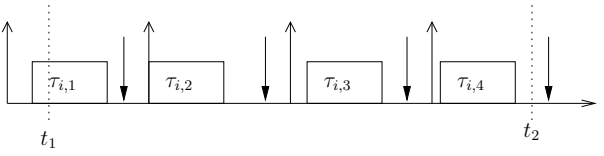


Fig. 5. Jobs of τ_i

The processor demand inside the time-interval is equal to the number of jobs which were activated and must be completed inside the time-interval multiplied by the computation time W_i. Therefore, $g_i(t_1, t_2) = max\{0, \lfloor \frac{t_2+T_i-D_i-\Phi_i}{T_i} \rfloor - \lceil \frac{t_1-\Phi_i}{T_i} \rceil\} W_i$ and the processing demand for all task set is :

$$g(t_1, t_2) = \sum_{i=1}^{n} max \left\{ 0, \left\lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \Phi_i}{T_i} \right\rceil \right\} W_i \qquad (2)$$

The schedulability of an asynchronous task set with deadline less than or equal to period can be verified by $\forall t_1, t_2 \qquad g(t_1, t_2) \leq (t_2 - t_1)$. In asynchronous task sets the schedule must be verified up to $2H + \Phi$ [18] where H is the hyper-period ($H = lcm\{T_1, T_2, \ldots, T_n\}$) and Φ is the largest offset among tasks ($\Phi = max\{\Phi_1, \Phi_2, \ldots, \Phi_n\}$). Hence, the schedulability test must check all busy periods in $[0, 2H + \Phi]$, which has an exponential time complexity $O(H^2)$ [19].

Accounting the Interference of Subtasks B

In [16] Jeffay and Stone have shown a schedulability condition to ensure the schedulability of EDF in the presence of interrupts. Basically, they assume interrupts as higher priority tasks which preempt every application task. Therefore the interrupt handler interference is considered as a time that is stolen from the application tasks. So, if tasks can finish before their deadlines even suffering the interference from the interrupt handler, the task set is schedulable. The task set is composed by n application tasks and m interrupt handlers. Interrupts are described by a computation time CH and a minimum time between jobs TH. The processing time for execute interrupts is $f(L)$.

Theorem 1. *A set τ of n periodic or sporadic tasks and a set ι of m interrupt handlers is schedulable by EDF if and only if*

$\forall L \geq 0 \qquad g(0, L) \leq L - f(L)$

where the upper bound $f(L)$ is computed by:

$f(0) = 0$

$$f(L) = \begin{cases} f(L-1) + 1, & if \sum_{i=1}^{m} \left\lceil \frac{L}{TH_i} \right\rceil CH_i > f(L-1) \\ f(L-1), & otherwise \end{cases} \qquad (3)$$

The proof is similar to the proof in [17]. The difference is that in any interval of length L, the amount of time that the processor can dedicate to the application tasks is equal to $L - f(L)$.

Using this method, subtask B_i is modeled as an interrupt handler, subtasks A_i and C_i are implemented as EDF subtasks and the feasibility checked using Theorem 1. The Theorem to account for interrupt handlers as expressed by Jeffay and Stone assumes a synchronous task set with deadlines equal to periods.

We extend this Theorem using the processor demand approach to represent asynchronous systems and deadlines less than periods. In this case, subtasks A_i arrives at time zero ($\Phi_{A_i} = 0$) and C_i arrives at time Φ_{C_i}. We assume that at a specific time an interrupt starts B_i (using the designation in [16]). To ensure subtask C_i runs only after subtask B_i, subtask C_i has the offset set to $\Phi_{C_i} = D_{B_i}$. We assume $F(t_1, t_2)$ as the processor demand due to interrupts in $[t_1, t_2]$. The new feasibility test in which all subtasks C_i have offsets and subtasks B_i are modeled as interrupts is: $\forall L \geq 0 \qquad g(t_1, t_2) \leq (t_2 - t_1) - F(t_1, t_2)$

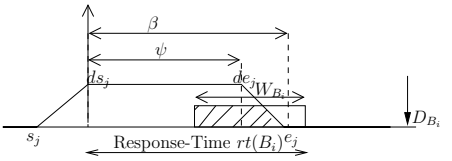
Subtasks B have a time-window during which can be active. So, applying [16] is pessimistic due to the accounting of interrupts where they cannot execute. An improvement is obtained by inserting an offset ΦH_i as in $\sum_{i=1}^{m} \lceil \frac{L-\Phi H_i}{TH_i} \rceil CH_i$ to represent the fact an interrupt cannot happen before $Bmin$. Algorithm 1 has complexity $O(H^2)$. Unfortunately, in the worst case (where all periods are prime numbers) the hyper-period is the product of all periods $\prod_{i=1}^{n} T_i$. Therefore, in practical situations the algorithm can be applied only when task periods result in a small hyper-period.

Algorithm 1. Feasibility test - first test

```
for all t1 such that 0 ≤ t1 ≤ 2H + Φ do
  for all t2 such that t1 ≤ t2 ≤ 2H + Φ do
    g(t1,t2) = Σ_{i=1}^{n} max{0, ⌊(t2+Ti−Di−Φi)/Ti⌋ − ⌈(t1−Φi)/Ti⌉}Wi
    F(t1,t2) = f(t2) − f(t1)
    if g(t1,t2) > (t2 − t1) − F(t1,t2) then
      return nonfeasible
    end if
  end for
end for
{It is feasible. Apply the second test}
return feasible
```

Feasibility Test Based on Response-Time. Differently from subtasks A_i and C_i which are scheduled by EDF, our scheduling approach assigns a **fixed** priority to all subtasks B_i according to a heuristic rule. The heuristic rule has two metrics, **criticality** and **slide factor**. In the first metric, tasks can be strict or cumulative. Subtasks with the strict criticality correspond to a group of subtasks with higher priorities than subtasks with cumulative criticality. Inside each group, priorities are given inversely proportional to the sliding factor computed as $sf_i = \frac{\psi}{W_{B_i}}$. The sliding factor is related to the capacity to slide inside the ideal time-interval and obtain the highest QoS value.

We intend to verify the schedulability of B_i computing its response-time (rt), assuming that all subtasks B_i are always released at ds_j as shown in Fig. 6. In



$rt(B_i)$	QoS
$rt \leq \psi$	100%
$rt \geq \beta + W_{B_i}$	0%
$\psi < rt < \beta + W_{B_i}$	$QoS(B_i, rt(B_i) - W_{B_i}, rt(B_i))$%

Fig. 6. QoS Value According to the rt

the same figure, we use β to describe the time-interval between the release at ds_j up to e_j. In subtasks with cumulative criticality (as shown in Fig. 2) it is possible to finish after the ideal time-interval, resulting in a lower QoS value. In contrast, subtasks with a strict criticality (Fig. 3) demand the execution inside the ideal time-interval i.e: it is necessary to verify if in the worst possible scenario $rt(B_i) \leq \psi$. Note that in a strict subtask B_i, $s_j = ds_j$, $de_j = e_j$.

The response-time can be divided into worst-case response-time ($wcrt$) and best-case response-time ($bcrt$). The $wcrt$ provides the worst possible scenario for the execution of B_i and in this sense the QoS value is the minimum possible. On the other hand, the $bcrt$ provides the best possible scenario for B_i resulting in the maximum QoS value. Computing the $wcrt$ and the $bcrt$ of subtask B_i makes it is possible to obtain a QoS value as shown in Fig. 6. Therefore, applying the $wcrt$ of a subtask B_i as a response-time in Fig. 6 results in the minimum possible QoS value. In contrast, applying the $bcrt$ as a response-time results in the maximum possible QoS value. The first line in the table inside Fig. 6 covers the case where all B_i runs inside the ideal time-interval. The second line covers the case where the execution takes place outside the time-interval and the third line covers the case where part of B_i runs inside the time-interval.

Computing the Response-Time

The worst-case response time of non-preemptive sporadic subtasks can be determined by the sum of three terms.

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (W_{B_j}) + \sum_{j \in hp(i)} W_{B_j} \tag{4}$$

The first term in (4) is the worst-case execution time of subtask B_i. The second term is the maximum blocking time due to subtasks running at moment B_i is released. We account this value as the maximum execution time among the subtasks B_j with a lower priority (lp) than B_i, leaving the interference of higher priority (hp) subtasks for the next term. The last term is the maximum blocking time due to subtasks B_j with higher priorities. We account this value adding all subtasks B_j with higher priorities than B_i.

Unfortunately, in some situations the time-windows, in which B_i and B_j can be activate may not overlap. In this case, it is impossible for B_j to produce interference upon B_i, even though it has a higher priority. For instance in B_i ($W = 2$, $T = 50$, $Bmin = 10$, $Bmax = 20$, $D = 30$, $prio = 1$) and B_j ($W = 5$, $T = 50$, $Bmin = 35$, $Bmax = 45$, $D = 55$, $prio = 2$). The time-windows do not

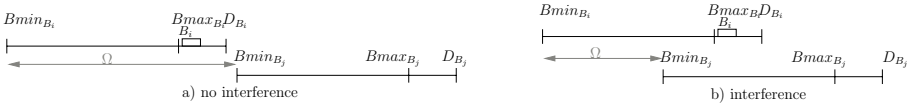


Fig. 7. Interference of B_j upon B_i

overlap, so there is no interference between B_j and B_i (Fig. 7 item *a*). However, if we change $Bmin_{B_j} = 15, Bmax_{B_j} = 35, D_{B_j} = 45$ the time-windows overlap and there is interference between B_i and B_j to account (Fig. 7 item *b*).

We extend (4) to take into account only the subtasks which produce interference (represented as I) upon B_i (5). The Ω in (6) gives the smallest time-length between the earliest release time of B_i and B_j. If Ω is smaller than the distance between $[D_{B_i}, Bmin_{B_i}]$, the time-windows overlap resulting in interference accounted as the worst-case execution time of B_j.

$$wcrt_{B_i} = W_{B_i} + \max_{j \in lp(i)} (I_{(B_j,B_i)} \cdot W_{B_j}) + \sum_{j \in hp(i)} I_{(B_i,B_j)} \cdot W_{B_j} \tag{5}$$

$$I_{(B_i,B_j)} = \begin{cases} 1, & if(\Omega < (D_{B_i} - Bmin_{B_i})) \\ 0, & otherwise \end{cases}$$

$$\Omega = Bmin_{B_j} - Bmin_{B_i} + \left\lceil \left(\frac{Bmin_{B_i} - Bmin_{B_j}}{gcd(T_{B_i}, T_{B_j})} \right) \right\rceil \cdot gcd(T_{B_i}, T_{B_j}) \tag{6}$$

The best-case response time for subtasks B_i occurs when B_i does not suffer any interference from other subtasks B_j. As a result, $bcrt_{B_i} = W_{B_i}$.

4 Experimental Evaluation

In this section we illustrate the effectiveness of the proposed feasibility test comparing its result with a simulation performed on the same task set.

Our experiment is composed by three tasks (τ_1,τ_2,τ_3), each of them subdivided into three subtasks. The worst-case execution times, periods, deadlines and offsets are presented in Table 1. Also, it show the specific parameters of subtasks B such as criticality, priority, ρ,ψ,$Bmin$ and $Bmax$. The results of the offline test can be seen in Table 2. The subtask B_2 (with strict criticality) always runs inside the ideal time-interval, resulting in the maximum QoS. The other two subtasks have cumulative criticality and present a minimum QoS of 41.6% and 25.0% respectively. Due to a pessimistic test, the *wcrt* shown in Table 2 is an overestimation of the actual values. Therefore, we should expect that the actual minimum QoS might be higher than the values given by the offline test.

We simulate the same task set for 10.000 time units, assuming the release time uniformly chosen between $Bmin$ and $Bmax$ (Table 3). Subtasks B_i and C_i are required in 90% of τ_i activations. The simulation shows a consistent result where the minimum QoS values are equal or higher than the values given by the

Table 1. Example With Three Tasks

τ	subtask	W_i	D_i	T_i	Φ_i	criticality	prio	ρ	ψ	Bmin	Bmax
	A_1	4	10	40	0						
τ_1	B_1	6	31	40	11	cumulative	3	12	10	10	20
	C_1	2	40	40	31						
	A_2	3	20	40	0						
τ_2	B_2	2	34	40	20	strict	1	8	8	20	26
	C_2	2	40	40	34						
	A_3	2	15	60	0						
τ_3	B_3	6	31	60	18	cumulative	2	14	8	15	20
	C_3	1	60	60	31						

Table 2. Offline Feasibility Results

subtask	wcrt	bcrt	min QoS	max QoS
B_1	14	6	41.6%	100.0%
B_2	8	2	100.0%	100.0%
B_3	14	6	25.00%	100.0%

Table 3. Simulation Results

subtask	wcrt	bcrt	min QoS	max QoS
B_1	14	6	41.6%	100.0%
B_2	7	2	100.0%	100.0%
B_3	13	6	41.6%	100.0%

offline test. Thus, the offline test can guarantee that during its execution no task will ever obtain a lower QoS than computed using the offline test.

5 Conclusions and Future Work

This paper presented a useful model for a class of real-world problems which by the lack of theoretical study, are implemented with conventional schedulers resulting in lack of predictability. We applied some approaches from the real-time literature with adaptations to our scheduling problem to create an offline test which providing an accept/reject answer for tasks with critical benefit constraints and a minimum/maximum expected benefit for non-critical tasks. As a future work, we intend to investigate the scheduling problem where the B segments are preemptive and need exclusive access upon resources.

References

[1] Leung, J.Y.T., Whitehead, J.: On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Performance Evaluation 2, 237–250 (1982)
[2] Layland, J., Liu, C.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM 20(1), 46–61 (1973)
[3] Ravindran, B., Jensen, E.D., Li, P.: On Recent Advances In Time/Utility Function Real-Time Scheduling And Resource Management. In: 8^{th} IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (2005)
[4] Locke, C.D.: Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives. Real-Time Systems 4, 37–53 (1992)

[5] Tokuda, H., Wendorf, J.W., Wan, H.: Implementation of a Time-Driven Scheduler for Real-Time Operating Systems. In: 8^{th} RTSS, pp. 271–280 (1987)
[6] Burns, A., Prasad, D., Bondavalli, A., Giandomenico, F.D., Ramamritham, K., Stankovic, J., Strigini, L.: The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. Journal of Systems Architecture 46, 305–325 (2000)
[7] Buttazzo, G.C., Spuri, M., Sensini, F.: Value vs. Deadline Scheduling in Overload Conditions. In: 16^{th} IEEE Real-Time Systems Symposium, pp. 90–99 (1995)
[8] Liu, J., Shih, W.K., Lin, K.J., Bettati, R., Chung, J.Y.: Imprecise Computations. In: Proceeding of the IEEE (1994)
[9] Jensen, E., Locke, C., Tokuda, H.: A Time-Driven Scheduling Model for Real-Time Operating Systems. In: Real-Time Sytems Symposium (1985)
[10] Wu, H., Ravindran, B., Jensen, E.D., Balli, U.: Utility Accrual Scheduling under Arbitrary Time/Utility Functions and Multi-unit Resource Constraints. In: Proceedings of the 10^{th} RTCSA (2004)
[11] Lipton, R.J., Tomkins, A.: Online Interval Scheduling. In: 5^{th} annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, pp. 302–311 (1994)
[12] Hassin, R., Shani, M.: Machine Scheduling with Earliness, Tardiness and Non-Execution Penalties. Computers and Operations Research 32, 683–705 (2005)
[13] Velasco, M., Martí, P., Fuertes, J.M.: The Self Triggered Task Model for Real-Time Control Systems. In: 24^{th} RTSS (WiP) (2003)
[14] de la Rocha, F.R., de Oliveira, R.S.: Time-Interval Scheduling and its Applications to Real-Time Systems. In: 27^{th} Real-Time Systems Symposium (WiP) (2006)
[15] Gutierrez, J.P., Harbour, M.G.: Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF. In: 15^{th} ECRTS (2003)
[16] Jeffay, K., Stone, D.L.: Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. In: 14^{th} RTSS (1993)
[17] k.Baruah, S., Howell, R.R., Rosier, L.E.: Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. Real-Time Systems 2, 301–324 (1990)
[18] Leung, J., Merill, M.: A Note on the Preemptive Scheduling of Periodic, Real-Time Tasks. Information Processing Letters 11, 115–118 (1980)
[19] Goossens, J.: Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints. PhD thesis, Université Libre de Bruxelles (1999)

Towards a Software Framework for Building Highly Flexible Component-Based Embedded Operating Systems

Dong Xu, Hua Wang, Qiming Teng, and Xiangqun Chen

Operating System Laboratory, Department of Computer Science and Technology
School of EECS, Peking University, Beijing, 100871
{xudong,wanghua,tqm}os.pku.edu.cn, cherry@cs.pku.edu.cn

Abstract. Emerging new computing models make embedded systems become more ubiquitous and pervasive. To adapt the dynamic computing environment, future embedded operating system (EOS) is required to be highly flexible: the static image composition can be configured and the runtime structure can dynamically evolve. In this paper, we present a software framework for building such an EOS through a component-based approach. One unique feature of our framework is its ability of supporting black-box software reuse. This capability permits components from third-party systems to be reused, and frees component developers from the burden of meeting certain implementation constraints imposed by the component model. Based on a flexible binding model, the component runtime service that resides in the nucleus of this framework provides reconfiguration functions to support runtime changes in components and connectors at different levels. To evaluate this framework, we have reorganized uC/OS-II into a component-based one, and we also have implemented a prototype system named as TICK which consists of both native components and reused components. Experiment results show the performance cost induced by our framework is controllable and acceptable.

1 Introduction

As the rapid expansion of the application domain of computing technology, and the growing maturity of network infrastructure, emerging computing models make embedded systems become more ubiquitous and pervasive[1,2]. This trend forces embedded systems to put more focus on the issues of minimization, self adaptability, personalization and etc. To adapt the dynamic computing environment, future embedded systems should be highly flexible, which indicates that the static image composition can be configured and the runtime structure can dynamically evolve. For example, a smart computing node, whose static image composition is configured with its only required components, knows to load TCP/IP protocol stack when it enters a wired network environment, while replacing the TCP/IP stack with Bluetooth protocol when switching to wireless network. The whole process should be done without human administration.

Building flexible EOS from components has been an active area of operating system research. Previous work such as OSKit [3], eCos [4], PURE [5] provide configuration capability by encapsulating functionalities into components, eCos and

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 170–181, 2007.

PURE also provide tools to aid the configuration process. Systems like Choices [6], 2K [7], Pebble [8] and MMLite [9] provide reconfiguration capabilities by different approaches, but few of them provide reconfigurable ability for system level OS components. Moreover, the component-based approaches of these systems lie in the predefined ways component can interact and bound together.

Another problem in existing component-based EOSes is that there are no rules or tools for supporting extract or produce components from third party systems. Component developers cannot directly reuse components from the system adopting a different component model, and also, without supporting tools, people cannot directly reuse legacy codes in existing non component-based EOS.

This paper presents a software framework for building EOSes from binary components. The binary components can be purposely made or extracted from pre-existing ELF files and composed at build, boot, or runtime to build a system.

The primary contributions of this paper are as follows:

1) Within our knowledge, our framework is the first one that supports black-box software reuse. We propose a unique approach to extract and produce reusable binary components from existing systems, and then reuse them in the framework.

2) We build a component runtime service as the runtime infrastructure that manages system components and user components in a unified way. It enables reconfiguration ability on both system/user components and connectors, and makes the runtime structure evolvable.

3) Our supporting toolkit covers almost the whole construction process.

To evaluate our framework, we have successfully reorganized uC/OS-II into a component-based one. We have also built a prototype system TICK[1] which consists of native components and reused components. Experiment results show the performance cost induced by our framework is controllable and acceptable.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 details our software framework. Section 4 presents the experimental study and section 5 concludes this paper.

2 Related Work

There have been several works in the past decade on building component-based EOS. OSKit [3], an early component system, thinks that application orientation and reusability can be achieved by providing a “box of building blocks”, but it has no considerations on the building rules. Systems like eCos [4], PURE [5], Choices [6], Pebble [8], 2K [7] are built based on certain frameworks in which the kernel design specified fixed ways components can interact and be bound together. This issue has been addressed by THINK [10]. Its proposed software framework allows OS architects to assemble components in varied ways, but it does not support dynamic reconfiguration functionalities. All these systems adopt different component models, but none of them provide means to reuse components from each other, and even from the third party systems.

[1] TICK stands for Tick Is Component Kernel.

Our framework does not impose a particular kernel design on the OS architect. In this respect, our framework is similar to THINK. Unlike THINK, our framework provides reconfiguration functions to support run-time changes in components and their connectors at both user level and system level. Other differences between our framework and THINK include:

1) Component model: Compared with THINK's component model ODP [11], our component model is a conceptual model, it does not put any constraints on component implementations.

2) Software reusability: Our framework supports black-box software reuse. Codes in either component-based or non component-based systems can be reused, regardless of the different component models.

3) Binding model: Unlike THINK, our binding model dynamically generates connectors from a pre-translated binary binding routine repository, with which synchronization and mutual exclusion issues can be made transparent to components.

3 An Overview of Our Software Framework

In this section, we first introduce the overall structure of our framework. Then the approaches of supporting black-box software reuse and runtime structure evolution are presented. We at last discuss the construction process and supporting tools.

3. 1 The Overall Structure

As illustrated in Figure 1, our framework models the target system as a collection of components and connectors that are collaboratively running with the supports from the underlying nucleus. A component in our framework is a functional entity that encapsulates data and behavior. Our framework supports black-box software reuse, so there are two kinds of components: the native ones are developed from scratch, and the others are directly reused from existing systems. Connectors are a sort of abstraction of the connection relationships amongst components. They can be a few of

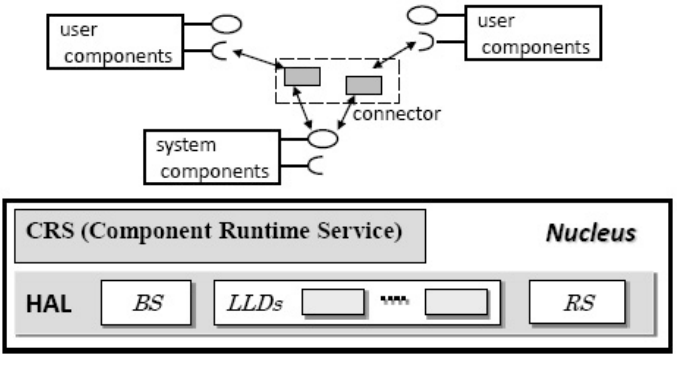


Fig. 1. The overall structure of our framework. BS stands for Bootstrap Service, LLD stands for Low Level Driver and RS stands for Runtime Service.

instructions that perform a direct call, or be coarse-grained components that encapsulate complicate inter-component communication semantics.

The nucleus, which comprises a HAL and a component runtime service (CRS), provides an abstract interface to hardware and management facilities for upper-level components. HAL provides portability for upper-level components, while the CRS plays a role as runtime infrastructure that supports run-time changes in both components and connectors at different levels. Readers can refer to [12] for an explanation of HAL design. As to CRS, we discuss it in section 3.3.

All ingredients in this framework are statically configurable, including the upper-level components, connectors and the underlying nucleus.

3.2 Supports for Black-Box Software Reuse

In our framework, we build components directly from ELF object files [13]. Choosing the binary object files as raw component candidates is reasonable:1)Building binary components can benefit from the fact that programming language used to write the source component is not restricted, developers can write component codes without meeting certain implementation constraints imposed by the component model.2) Building components directly from object files makes it possible to reuse components using different component models without source-level re-encapsulating.3)ELF standard is widely accepted in UNIX family operating systems and other systems, there potentially exist lots of high quality legacy codes to be examined.

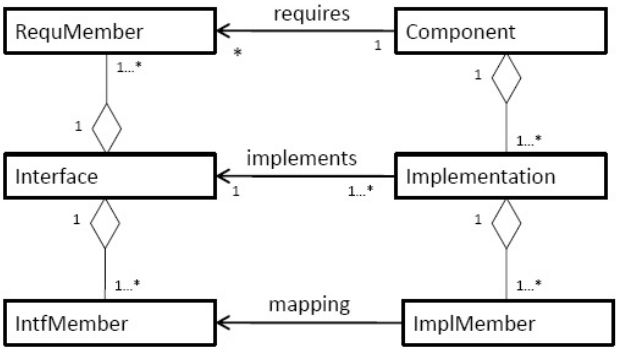


Fig. 2. Conceptual diagram of the component model. We look upon a component as a container for one or more implementation(s) of certain interface(s).

Figure 2 depicts the conceptual diagram of our component model. We look upon a component as a container for one or more implementation(s) of certain interface(s). To correctly operate in a target context, a component may require facilities from other entities in the system. These required (imported) items are referred to as *RequMember*, which are modeled as a tuple (*Intf*, *Impl*, *Member*). When requiring a portal from outside of the component, the component can optionally specify the *Impl* element. A NULL *Impl* element indicates that any implementation that implements the specified interface is acceptable. A component must have at least one implementation of an

interface, otherwise the component is regarded useless. For trusted components, the interface member, viz. *IntfMember*, can be either a variable or a function. Untrusted components can have only functions as interface members. Since the intended domain is an embedded system that might have no memory protection supports, the assumption of direct bindings among components are reasonable. However, for those systems that do support memory protection or privilege modes, no interface can export or import data members directly. In this case, all bindings among components that reside in different protection domains should be done using special facilities (e.g. trappings, upcalls).

Following steps comprise our component production process:

1) For developers who develop components from scratch can directly go to step 3 after compiling the component source codes into object files.

2) Identify and extract candidate object files from legacy systems. We developed a tool named DEREF which can automatically extract and visualize the architecture structure of legacy systems based on cross references among object files [14].With the assistance of related documents and domain knowledge, people can identify and extract required software modules easily.

3) Transform selected object files into reusable, self-contained components. The conversion process is constituted of the following steps: First, we present functions/variables in the object file to users for syntactic information recovering. All function/variable names are reprogrammed upon their signatures, which comes from header files or related documents. By this way, ambiguity caused by duplicate names in ELF files is eliminated. Second, we specify the properties of interface members, such as the reentrancy, visibility and so on. Third, we extract codes and data in the related sections of object files into implementation members, thus the implementations are separated from interfaces. At the same time, we also extract relocation information from object files into the component files. Fourth, in order to prevent hostile modifications, we sign the component with a MD5 fingerprint. Finally, we attach the component description information in the component header.

Since the binary object file is no longer restrained by the component model, it is feasible to re-produce binary components with different component models into our component format. Reusing legacy codes in existing systems can also benefit from this approach.

3.3 Supports for Runtime Structure Evolution

The key nucleus component for supporting target system runtime structure evolution is the CRS. In this section, we primarily focus on the binding model used by the CRS, we refer reader to [15] for the details of all the parts of CRS.

Connectors generated by CRS represent some sorts of inter-component communication semantics. Most of them are implemented into two code snippets named prolog and epilog, respectively. Figure 3 depicts our binding model [16] adopted by CRS: supposing that two components *A* and *B* are loaded into memory by CRS, and that the *functionA* in *A* requires a service *functionB* from *B*, a connector is needed. The connector is dynamically generated according to the user specified binding type. In this case, *functionB* in *B* is specified to be accessed exclusively. The corresponding prolog and epilog thus incorporate *seizeMutex* and *freeMutex* logic

copied from the *Binding Routines* repository. The CRS redirects function call (a relocatable call instruction) in the caller's code to the prolog in which the last *goto* instruction points to *functionB*. The CRS also modifies the calling stack frame of *functionA*, in which the return address of *functionB* ('loc_call' in Figure 3) is replaced by the start address of the epilog, so that it appears to *B* that *functionB* was called by the epilog. When *component B* finishes servicing *A*, the return statement actually transfers the execution to the epilog, which finally goes back to the 'loc_call'. Thus *component A* is ignorant of the intercepting procedure happened: it seems to *A* that the function call behaves just like usual cases, and everything is done as expected.

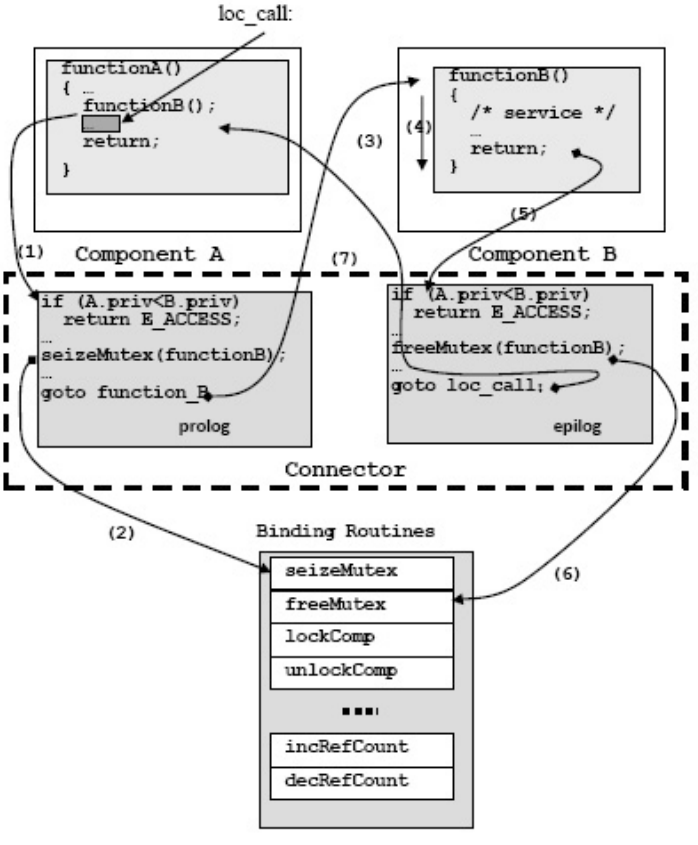


Fig. 3. The binding model. Binding routines are pre-translated binary relocatable code snippets shipped along with the CRS component. This repository is customizable when building the target image.

An interesting point of this binding model is that reference counting, component locking, and cross-domain interactions can be implemented in the same way. Supporting more complicated inter-component communication semantics such as

RPC is possible by encapsulating them into coarse grained connector components, thus they can be bound with other components in the same way. By abstracting the OS specific communication semantics into connectors, issues such as synchronization and mutual exclusion can be made transparent to connected components.

We enable reconfiguration ability of components by adopting techniques like reference count and r/w locks in the connectors. The replacement of connectors is similar to that of components since we can view connector as a special kind of component. We do not support state transfer like the K42 [17] does. State transfer usually adds constraints on component interface semantics, which limits the range of components to be used.

The CRS manages the system/user components and connectors in a unified way, since the OS mechanisms and policies are all encapsulated in upper-level components. By enabling reconfiguration functions, the runtime changes in components and connectors at different levels are supported. As a result, the runtime structure can dynamically evolve.

3.4 Construction Process and Supporting Tools

The construction process for a particular EOS based on our framework is similar to common component-based software development approaches, which follows two major steps:

1) Component acquisition/production: components in target system can either be bought from third-party vendors, or be developed from scratch. They can also be extracted and produced directly from legacy systems (e.g., RTEMS, eCos etc.). All components produced in this phase are collected into a component library.

2) Configuration and composition: at the configuration stage, OS architects select components from the component library, specify the connectors, and configure the components and the nucleus. Once the checking process of composition relationships and constraints passed, the target can be composed at build, boot or runtime.

Tools developed to aid this process include:

- Component repository system: a component library management tool, features including component storage, search etc.
- DEREF: this tool extracts and visualizes architecture structure of legacy systems which helps OS architect identify raw black-box components.
- Component maker: a semi-automatic tool used to product components from ELF object files.
- Composition tool: a visual environment that supports the target system configuration and the component composition.

4 Experimental Study

In this section, we have reorganized the non component-based real-time kernel uC/OS-II into a component-based one. We also have built a prototype system named TICK based on our framework. All measurements given below are performed on SAMSUNG S3C44B0X board with a 66MHZ ARM7TDMI processor.

4.1 Reorganize uC/OS-II into a Component-Based One

We have extracted 14 object files directly from the kernel of uC/OS-II for S3C44B0. We also have encapsulated the kernel C library references into a Clib object file. We have written an application as our benchmark which covers most of uC/OS-II functionalities, and it is used to verify the correctness of the reorganized system. Table 1 compares the size of object files and their corresponding components. We can see that the size of uC/OS components is smaller than that of the object files. This is because the uC/OS object files contain many debugging sections which are not extracted into the corresponding components. The Clib object file does not contain such sections, the attached self description information in Clib component makes it bigger than the original object file. We have built a HAL which only contains a BS module, and all the components are configured to be connected by direct call connectors. The target system composition is set to be done at boot-time.

Table 1. Size comparison between object file and component

	obj. file size(bytes)	**comp. file size(bytes)**
Clib	23,696	44,184
uC/OS	46,646	42,724

Table 2. Time cost of boot-time components loading and binding

instructions	**time(μs)**	**cycles**
5,489	271.65	17,929

The experiment shows that the application successfully starts to run. It proves that our component production approach is feasible to support the black-box software reuse. The time cost of boot-time components loading and binding is shown in Table 2. This cost is acceptable because it has no impacts on the performance of the target system itself. As compared to the statically linked uC/OS-II image which is 67,308 bytes sized, we have introduced more than 19,600 bytes extra space cost. However, this space cost is controllable: because there are no components need to be reconfigurable, the extra space cost induced by component files can be reclaimed after the CRS completes the target system composition. Under the circumstance that the total component size exceeds the memory limitation, we can compose the system at build time rather than at boot-time, thus there is no space cost induced.

4.2 TICK, a Component-Based EOS

We have built a component-based EOS named TICK based on our software framework. TICK's structure is shown in Figure 4. TICK consists of both native components and reused components. Five types of connectors are configured to connect components. The inter-component communication semantics in these connectors include direct call, system call, mutual exclusion, enable/disable interrupt and lock/unlock component.

4.2.1 Reuse Black-Box Components from Other Systems

The memory management component in TICK is reused from ThreadX [18] which implements a kind of linear memory management algorithm. Also, we extracted 8 object files from VxWorks I/O subsystem, and developed an 'adapter component' to convert their external dependencies into TICK's components, interfaces of TICK components involved in this adaptation including semaphore and memory management.

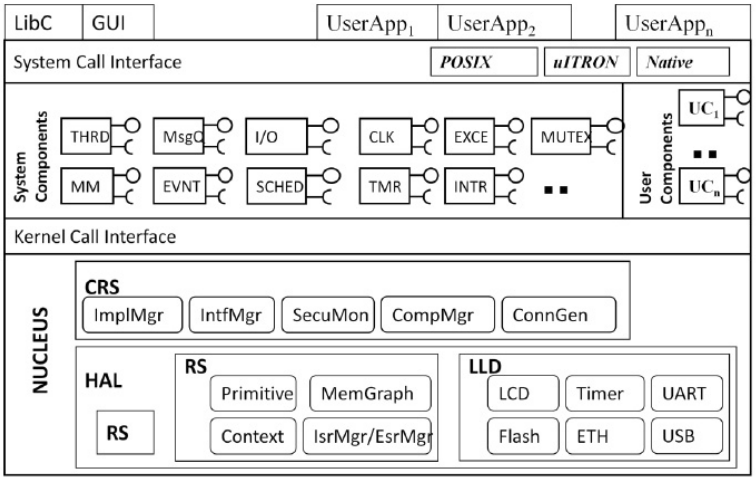


Fig. 4. The structure of TICK.CLK is a component for clock management. TMR is timer component and EXCE is exception management component. ETH stands for the low-level driver of Ethernet adapter.

4.2.2 Reconfigurable Ability

In TICK, a prolog/epilog template is used to generate connectors:

```
UINT32 bindConnectCode[] = {
    /*prolog:*/
    0xe92d100f,     /* stmfd sp!, {r0-r3,ip},   push registers */
    0xe59f0038,     /* write Param1 of preStub */
    0xe59f1038,     /* write Param2 of preStub */
    0xe59f2038,     /* write Param3 of preStub */
    0xe59f3038,     /* write Param4 of preStub */
    0xeb000000,     /* call preStub,need to be   relocated */
    0xe8bd100f,     /* ldmfd sp!, {r0-r3, ip},   pop registers */
    0xeb000000,     /*branch required_func,need   to be relocated*/
    /* epilog: */
    0xe92d100f,     /* stmfd sp!, {r0-r3, ip} */
    0xe59f001C,     /* write Param1 of postStub */
    0xe59f101C,     /* write Param2 of postStub */
    0xe59f201C,     /* write Param3 of postStub */
```

```
0xe59f301C,     /* write Param4 of postStub */
0xeb000000,     /* call postStub,need to be relocated */
0xe8bd100f,     /* ldmfd sp!, {r0-r3, ip} */
0xea000000,     /*back to the instruction after required_func */
0x00000000,     /* value of preStub Param#1,to be relocated */
0x00000000,     /* value of preStubParam#2, to be relocated* /
0x00000000,     /* value of preStub Param#3,to be relocated */
0x00000000,     /* value of preStub Param#4,to be relocated */
0x00000000,     /*value of postStub Param#1,to be relocated*/
0x00000000,     /*value of postStub Param#2,to be relocated*/
0x00000000,     /*value of postStub Param#3,to be relocated*/
0x00000000,     /*value of postStub Param#4,to be relocated*/
};
```

Based on this template, we induced extra 96 bytes, 64 bytes of which are used for 16 instructions, and the other 32 bytes are used by data for at most 8 parameters. The cost of this template is given in Table 3.

The 'preStub' and 'postStub' in this template refers to the routines that implemented certain communication semantics. Table 4 summaries performance of connectors in TICK. A direct call connector is generated by filling the callee's address into the caller's function call instruction, so there is no extra cost in this case. System call connector does not use the prolog/epilog code in this template. We registered a 'system call binding service' as a system call routine that will call the function specified in its parameters in supervisor mode. When a system call binding is required, the connector code will put the required function's address and parameters in general purpose registers, and then directly call the software interrupt instruction to request the system service.

Table 3. Performance cost of prolog/epilog template

Instructions	time(μs)	cycles
16	0.93	62

Table 4. Performance of connectors in TICK

Communication semantics	instructions	time(μs)	cycles
system call	61	2.15	142
mutual exclusion	260	9.01	595
enable/disable interrupt	26	1.21	80
lock/unlock component	761	24.98	1,649

The performance cost of each connector equals to the cost of prelog/prolog template plus the cost of corresponding preStub/postStub. Table 3 tells us that the cost of prolog/epilog is negligible. The cost of each preStub and postStub which implement certain communication semantics depends on its implementation, and this kind of cost is inherent in the interacting scheme.

To evaluate the performance of dynamic reconfiguration, we built two components A and B which encapsulated different priority-based scheduling policies. A is only

bound with scheduler component, and B is stored in host machine. Besides of the access semantics, the connectors between A and scheduler component encapsulate logic of reference counting and read lock acquisition. When dynamic reconfiguration event (we implemented it as a kind of user-defined event) is triggered, the CRS will download component B from host machine. After loading component B into memory, CRS will add a write lock on component A, and will rebind the scheduler component with component B once the reference count on component A becomes zero.

Table 5. Performance evaluation on a dynamic reconfiguration case

Step	instructions	time(μs)	cycles
load and internal relocate	1,644	49.43	3,263
bind	35	1.13	75

Table 5 shows the performance of this dynamic reconfiguration case. The cost of internal relocation is requisite when loading relocatable component into memory. The cost of runtime component binding depends on two factors: 1) the number of required functions/variables in components. In current implementation, each required function/variable needs to take 15 cycles when connecting itself with the corresponding provided one. Since the number of required functions/variables is configurable, this kind of cost is controllable.2) the performance of connectors. As we analyzed before, the extra cost induced by connectors is negligible. In real application scenario, only a few components (e.g., those encapsulate system policies) often need to be replaced, so the performance cost of dynamic reconfiguration is actually small.

5 Conclusion and Future Works

We have presented a software framework for building highly flexible component-based EOS that the static image composition can be configured and the runtime structure can dynamically evolve. We have discussed the component production approach as well as the component model, with which the black-box software reuse is supported. We have detailed the runtime infrastructure primarily on the binding model, which provides reconfiguration ability on the runtime structure. Also we have briefly introduced the construction process and supporting tools. To evaluate this framework, we have reorganized uC/OS-II into a component-based one, and we also have built a prototype system TICK using our framework. The evaluation results show that our approach is feasible and the performance penalties are controllable and acceptable.

Future works are to be carried out in the following aspects:

- Investigating method to verify the correctness of the binary component composition.
- Optimizing the CRS code, as well as the component organization format, thus the space cost can be lowered.
- Developing more kinds of connectors to support more complicated inter-component communications such as RPC.
- Exploiting existing EOS to enhance the component library.

Acknowledgments. Special thanks to Yi ZHAO who helped us tremendously improve on an earlier version of this paper. This work is sponsored by the National High-Tech Research and Development Plan of China under Grant No. 2002AA1Z2301 and 2004AA1Z2400.

References

1. Weiser, M.: Hot Topics: Ubiquitous Computing. IEEE Computer 26(7), 71–72 (1993)
2. Hendrickson, B., MacBeth, A., Gribble, S.: Systems Directions for Pervasive Computing. In: Proc. of the 8th Workshop on HOTOS, May 20-22, p. 147 (2001)
3. Ford, B., Back, G., Benson, G., et al.: The Flux OSKit: a substrate for kernel and language research. In: Proc. of the 16th ACM SOSP, October 05-08, pp. 38–51 (1997)
4. eCos. http://sources.redhat.com/ecos/
5. Beuche, D., Guerrouat, A., Papajewski, H., et al.: The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. In: Proc. of the 2nd IEEE international Symposium on Object-Oriented Real-Time Distributed Computing, May 02-05, 1999, pp. 45–53. IEEE Computer Society Press, Washington, DC (1999)
6. Campbell, R., Islam, N., Madany, P., et al.: Designing and Implementing Choices: An Object-Oriented System in C++. Communications of the ACM 36(9), 117–126 (1993)
7. Kon, F., Singhai, A., Campbell, R.H., et al.: 2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments. In: Demeyer, S., Bosch, J. (eds.) Object-Oriented Technology. ECOOP 1998 Workshop Reader. LNCS, vol. 1543, pp. 388–389. Springer, London (1998)
8. Gabber, E., Small, C., Bruno, J., et al.: The pebble component-based operating system. In: Proc. of USENIX Annual Technical Conference, CA, pp. 20–20 (1999)
9. Helander, J., Forin, A.: MMLite: A Highly Componentized System Architecture. In: Proc. of 8th ACM SIGOPS on Operating Systems European Workshop, pp. 96–103. ACM Press, New York (1998)
10. Fassino, J.-P., Stefani, J.-B., Lawall, J., et al.: THINK: A Software Framework for Component-based Operating System Kernels. In: USENIX 2002 Annual Technical Conference, CA, pp. 73–86 (2002)
11. ITU-T Recommendation X.903 | ISO/IEC International Standard 10746-3. ODP Reference Model: Architecture.ITU-T | ISO/IEC (1995)
12. Teng, Q., Wang, H., Chen, X.: A HAL for Component-Based Embedded Operating Systems. In: Proc.of the COMPSAC 2005, pp. 23–24. IEEE CS, Washington, DC (2005)
13. Qiming, T., Xiangqun, C., Xia, Z.: On Building Reusable EOS Components from ELF Object Files. Journal of Software 15(12a), 157–163 (2004)
14. Teng, Q., Chen, X., Zhao, X., et al.: Extraction and Visualization of Architectural Structure Based on Cross References among Object Files. In: Proc. of COMPSAC 2004, pp. 508–513. IEEE Computer Society, Washington, DC (2004)
15. Xu, D., Teng, Q., Chen, X.: Supports for Components Loading and Binding at Boot-time for Component-based Embedded Operating Systems. In: Proc. of ICESS 2005, December 2005, pp. 46–52. IEEE CS Press, Los Alamitos (2005)
16. Teng, Q., Chen, X., Zhao, X.: On Generalizing Interrupt Handling into A Flexible Binding Model for Kernel Components. In: Wu, Z., Chen, C., Guo, M., Bu, J. (eds.) ICESS 2004. LNCS, vol. 3605, pp. 323–330. Springer, Heidelberg (2005)
17. Baumann, A., Heiser, G., Appavoo, J., et al.: Providing dynamic update in an operating system. In: Proc. of USENIX 2005 Annual Technical Conference, CA, pp. 32–32 (2005)
18. ThreadX. http://www.rtos.com/

A Study on Asymmetric Operating Systems on Symmetric Multiprocessors

Yu Murata, Wataru Kanda, Kensuke Hanaoka,
Hiroo Ishikawa, and Tatsuo Nakajima

Department of Computer Science, Waseda University
{murata,kanda,kensk,ishikawa,tatsuo}@dcl.info.waseda.ac.jp

Abstract. This paper proposes a technique to achieve asymmetric multiple OSes environment for symmetric multiprocessors. The system has a host OS and guest OSes: L4 microkernel and their servers run as the host OS, and modified Linux runs as the guest OS. OS manager which is one of the servers on the host OS manages the guest OSes. Our approach avoids a lot of execution overheads and modification costs of the guest OSes because the each OS can control hardware directly without any virtualization. The results of the evaluation show that our system is much better than the existing virtual machine systems in point of the performance. In addition, a guest OS in our system requires a few amount of modification costs. Consequently, the experiments prove that our system is a practical approach for both performance and engineering cost sensitive systems.

Keywords: Operating Systems, Symmetric Multiprocessors, Multiple OSes Environment, InterProcessor Interrupts, InterOS Communications.

1 Introduction

Virtualization technology has become popular again because of the hardware improvement. The remarkable feature of virtual machine technologies is multiple operating systems (OSes) can run on a single machine. A virtualization technology principally consists of a virtual machine monitor (VMM) and some virtual machines (VMs). A VMM provides VMs which virtualize hardware for OSes run on them. If a VMM provides multiple VMs, multiple OSes can run on them concurrently. In general, an OS which runs on a VM is called a guest OS.

On the other hand, virtual machine approaches require higher hardware performance. Some techniques have been proposed to reduce overheads by modifying a guest OS for a specific VMM such as Xen, but they need expensive cost for modification. Therefore, it is significant to consider the trade-off between performance and engineering cost.

Various approaches have been proposed to achieve multiple OSes environment on a single machine. One of the advantages of running multiple OSes is efficient use of hardware resources. It is usually more efficient to run two OSes on a single machine than to run them on two machines. Because recent hardware has

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 182–195, 2007.

a amount of resources such as memory and hard disk, an OS can't always use all resources in a machine. Running more than two OSes on a single machine, such unused resources can be assigned to another OS. As a result, you can save the number of hardware. In addition, running multiple OSes concurrently improves the reliability of systems. If there are multiple OSes running on a system, you can access their services even when one of the OSes doesn't work. Thus, multiple OSes environment makes the system reliable.

We developed a system which achieves multiple OSes environment on a single machine. The system is named SIGMA system. SIGMA system doesn't require any specific hardware but only requires a symmetric multiprocessor (SMP) which is generally used in a personal computer. SIGMA system assigns one OS to one of the processors. We can say this environment is asymmetric because different kinds of OSes run on each processor in SIGMA system. One of the most important characteristics is that an OS of SIGMA system achieves not only the minimum engineering cost but also the little performance degradation although performance and engineering cost are trade-off relationships in the virtual machine technology.

We ran the benchmark software on its OS to evaluate the performance of SIGMA system and proved its advantage in performance. The evaluation result is actually significant because evaluation of asymmetric environment has hardly been performed before.

The remainder of this paper is structured as follows. Section 2 describes three virtualization techniques as related work which achieve multiple OSes on a single machine. Section 3 explains overview of SIGMA system and detail of its implementation. Section 4 shows evaluations in point of the performance and engineering cost. Section 5 discusses use cases of SIGMA system. Finally, Section 6 summarizes our research.

2 Related Work - Virtualization

The essence of the virtualization technology is a VMM which changes a single hardware interface to multiple interfaces. The interface is duplication of real hardware and equips all instructions (both privileged and unprivileged) of processors and hardware resources (memory and I/O devices, etc...) [3]. Such interface is called a VM, which is the virtualized hardware environment created by a VMM. The VM is controlled by the VMM and provides the same environment as the original hardware. Basically, a program runs on a VM should behave as it runs on an original hardware except some exceptions such as differences caused by the availability of system resources and timing dependencies [14].

Generally, virtualization technologies are categorized into three approaches: full-virtualization, para-virtualization, pre-virtualization.

2.1 Full-Virtualization

Full-virtualization is achieved by complete emulation of all instructions and hardware. The most significant advantage of full-virtualization is that almost all OSes

can be run on its VM without requiring any modification to them. In addition, since it emulates hardware completely, the approach does not degrade the reliability of OSes run on the VM. On the other hand, full-virtualization has performance problems. The VMMs developed for full-virtualization are VMware [15] [16] [19], Virtual PC, real hardware.

2.2 Para-virtualization

Para-virtualization is achieved by modifying a guest OS for a specific hypervisor. The performance of para-virtualization is superior to that of full-virtualization because the OS is optimized for the hypervisor. Though, such modification extinguishes some advantages achieved in full-virtualization. For a examples, a para-virtualized OS can't run on different hypervisors because the OS is created for only a single hypervisor. The examples of para-virtualization hypervisors and its guest OSes are Xen and XenoLinux [2], L4 microkernel and L4Linux [4].

Microkernel-based para-virtualization: A microkernel is one of kernels which has only core functions such as interrupt management, process management, an interprocess communication (IPC). The other functions are implemented as user process modules outside of the kernel. The kernel structure is simple and easy to manage. On the other hand, an IPC is generated very frequently to communicate with the modules and it leads context switches. The performance degradation of a microkernel is mainly caused from those switches. Although some approaches has been proposed to decrease those overheads for some architecture such as arm and x86 [12] [17] [20], the performance problems in microkernel approaches are the acute issues.

L4Linux is one of the para-virtualized Linux which is ported to run on a L4 microkernel. L4Linux is executed with other servers concurrently in the unprivileged mode. Wombat [9] is also one of the para-virtualized Linux which is ported to run on L4/Iguana system. L4/Iguana consists of NICTA::L4-embedded [8] and Iguana [5]. NICTA::L4-embedded is one of the L4 microkernel modified for embedded systems. Iguana is basic software provides OS services such as memory protection mechanisms and device driver frameworks for embedded systems. NICTA::L4-embedded, Wombat and Iguana have been developed at National ICT Australia (NICTA).

2.3 Pre-virtualization

Pre-virtualization is achieved by modifying assembler codes of a guest OS [10]. The modification consists of multi-phase process called afterburning. Since the afterburning is processed automatically, the engineering cost is relatively small. Pre-virtualization has both benefits of full-virtualization and para-virtualization, the low engineering cost and the high performance. The pre-virtualized OS supports multiple hypervisors such as Xen and L4 [18]. At present, a single compiled

image of a pre-virtualized OS can be run on multiple types of hypervisor. Users can select hypervisors for any purpose [11].

3 Design and Implementation

This section explains the design and implementation of SIGMA system, which provides multiple OSes environment especially for multiprocessor architecture. Although SIGMA system looks like a virtual machine monitor in terms of multiple OSes environment, it doesn't provide virtualized hardware interfaces to OSes running on it. An OS in SIGMA system runs on a physical hardware directly. The principal mechanisms to achieve SIGMA system are an interprocessor interrupt (IPI) [6] and an interOS communication (IOSC). The system is implemented on IA-32 architecture.

3.1 Approach and Overview

SIGMA system assigns each processor to a guest OS so that the OS can use all of the CPU resources exclusively. Consequently, this makes SIGMA system simple: it doesn't require a resource management mechanism such as the one implemented in a VMM. OSes of SIGMA system treat those operations by themselves. Those are significant advantages. The first reason is that the OSes hardly require to be modified because the OSes act as if they run on single normal hardware. The second is that the OS in SIGMA system can run in the privileged mode. That is, the OS can handle most of the privileged instructions. As a result, SIGMA system shows as high performance as a native system.

In the Intel architecture, one of the processor is selected as the bootstrap processor (BSP) and others are the application processors (APs) at system initialization by the system hardware [7]. After configuring the hardware, the BSP starts the APs. We define that an OS runs on the BSP is a host OS, and that an OS runs on the AP is a guest OS in SIGMA system. Especially, we call the guest Linux, SIGMA Linux, which is little modified its source codes for SIGMA system. In SIGMA system, NICTA::L4-embedded, Iguana and Wombat are run on the BSP, and SIGMA Linux is run on the AP. The architecture of the system is illustrated in Figure 1.

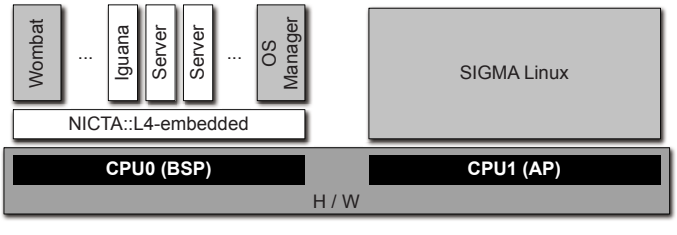


Fig. 1. The structure of SIGMA system

3.2 Boot Sequence of SIGMA Linux

All modules of SIGMA system including a SIGMA Linux kernel are attached to the memory at initial boot time. The OS manager that is one of the modules running on the host OS takes in charge of starting guest OSes. Before SIGMA Linux boots, the OS manager acquires the kernel information from L4. L4 has modules information such as the address in the memory, the image size and other parameters. In addition, to define the memory regions which are available from the guest OS, the OS manager creates new e820 memory maps. The detail of the memory management will be described in next paragraph. The OS manager also writes the information which SIGMA Linux refers such as kernel parameters to the specified memory address. Finally, OS manager sends startup IPI to the AP and SIGMA Linux starts booting. It seems that SIGMA Linux is booted by the general bootloader such as grub on the standalone system. For the guest OS, the OS manager is something like grub.

3.3 Memory Management

Although a normal Linux kernel acquires a memory map through a bios call to identify free and reserved regions in the main memory, a Linux kernel running on SIGMA system obtains the memory map from the OS manager.

Before booting SIGMA Linux, the OS manager executes the program which invokes the e820 function on the AP and acquires the e820 memory map. The e820 function is one of the bios service to check memory size and memory maps. Then, the OS manager modifies the memory map to create memory spaces for SIGMA Linux. The modification is based on the pre-defined memory regions which can be available for SIGMA Linux. After the modification, the OS manager overwrites the modified map to the memory and boots SIGMA Linux. Although the original native Linux includes the instructions which call the e820 function, those are removed from SIGMA Linux to prevent the modified memory map from being overwritten again. Finally, SIGMA Linux reads the modified memory maps and starts booting within the restricted memory areas. This approach is faced with a security problem that SIGMA Linux can ignore and even change this memory restriction. The problem is discussed in Section 5.

3.4 InterOS Communication

The IOSC is a mechanism to communicate between OSes run on the different processors in SIGMA system. The IOSC is similar in mechanism to an interprocess communication (IPC) of a general OS. The IOSC is processed by following steps.

(a) A source OS writes a message to a shared memory region.
(b) The source OS sends an IPI to destination processors.
(c) The destination OS receives the IPI and reads the message.

If the IOSC is called frequently, the system performance can be affected. This resembles IPCs of the microkernel-based system. Though, the IOSC hardly affects the performance of the system because the opportunity to use the IOSC

is limited and the frequency is too low as compared to that of IPC in the microkernel-based system.

3.5 Device Management

Device management is mainly achieved by the host OS. Although both the host and guest OSes can use the device directly in fact, if some OSes use the same device at the same time, this device access becomes controversial collision. SIGMA system avoids these collision by restricting the device access to the host OS.

Device drivers are implemented in the device servers of the host OS. A device access of a process in the host OS is performed by just calling the device server. On the other hand, when a process of a guest OS tries to use a device, the process has to send a request to the device server of the host OS by calling IOSC service. This process is performed by a stub driver of the guest OS, called a pseudo device driver. Since the interface of the pseudo device driver is same as common device drivers, the process can use the devices without knowing the location of the device and its driver.

4 Performance Evaluation

This section describes the evaluation of our prototype system. We ran a collection of benchmark programs to evaluate the performance of the guest OS. We also examined the collision on shared memory between the host OS and the guest OS. All experiments described in this paper were performed on a PC with a 2.4GHz Intel Core2 Duo with 32KB L1 cache for each core, 4MB shared L2 cache, 512MB of RAM, and an ATA disk drive. We used Linux kernel version 2.6.12, and 2.6.16 for Xen.

4.1 LMbench

We used the LMbench [13] to measure their performance. LMbench is the micro benchmark software to measure the performance of Linux. LMbench measures the performance of system calls, context switches, latencies of communications and so on. Since LMbench uses regular Linux system calls, it can be suitable to compare many Linux systems.

SIGMA Linux is compared to three systems: vanilla Linux kernel running on a single core (Native Linux), vanilla Linux kernel running on two cores (Native SMP Linux), and modified Linux kernel running on Xen virtual machine monitor (XenoLinux).

Figure 2 shows the cost of the interactions between a process and the guest kernel. The benchmark examines Linux system call and signal handling performance. Any of the items of SIGMA Linux is the same as that of native Linux, while XenoLinux takes longer time than native Linux in most of the cases. Some experiments of SIGMA Linux made better results than native Linux. This can be caused by the small difference of the environment: SIGMA Linux ignores some interrupts, but native Linux doesn't.

We evaluated the cost of a process context switch in the same set (Figure 3). The cost of a context switch on SIGMA Linux is also almost the same as that of the native Linux and less than that of the native SMP Linux and XenoLinux in any case.

The last benchmark evaluates the cost of the file operations (Figure 4). Xen is as fast as native Linux and SIGMA Linux in file create/delete operations. On the other hand, native SMP Linux is slower than the others. The result shows that SIGMA Linux performs as well as native Linux. It means that there are very few overheads in SIGMA Linux comparing to native Linux.

From the all results, SIGMA Linux presented as the almost same performance as native Linux. In addition, compared to native SMP Linux and XenoLinux, SIGMA Linux shows much better performance than them.

If multiple threads are appropriately assigned to processors and there are no dependencies between the threads, native SMP Linux can achieve efficient processing. Although it may be suitable for math calculation fields and image processing to introduce native SMP Linux, it is not always efficient way for general purpose use. To create a program which is optimized to parallel processing, a special compiler may be required such as automatic parallelizer and it may be also required to use a parallel programming language such as Fortress [1]. As described above, native SMP Linux often shows slower performance than native Linux on a single processor. There are some possible reasons. The first is lock contention: if one processor has already acquired the lock of the critical sections, the other processors have to wait until it will be unlocked to access the regions. In this case, the advantage of parallel execution cannot be made the best use of. The second is the memory access collision issues, which will be described in 4.2. The third is cache inconsistency. In a multiprocessor system, when one processor changes a page table or page directory entry, the change must be propagated to all the other processors to keep the cache consistency. This process is commonly referred to as TLB shootdown and performed by IPI [7]. TLB shootdown in IA-32 architecture must guarantee either of the following two conditions: different TLB mappings are not used on different processors during updating the page table, the OS is prepared to deal with the case where processors use the stale mapping during the table update. A processor can be stalled or must invalidate the modified TLB entries in order to keep the TLB consistency. As a result, this may degrade the system performance.

XenoLinux shows more than two times slower than SIGMA Linux and native Linux at worst. The most substantial element is the emulation of privileged instructions handled by Xen hypervisor. Many of IA-32 instructions related to memory and interrupt and I/O management must be performed in the privileged mode. To perform those operations, XenoLinux invokes hypervisor, which actually handles such instructions. Such invocations are called hypercalls. The hypercall is very expensive processing. Although some optimizations are implemented into Xen to reduce the cost of the hypercall, for a example, some hypercalls are handled together, it can't be enough to improve the performance degradation substantially.

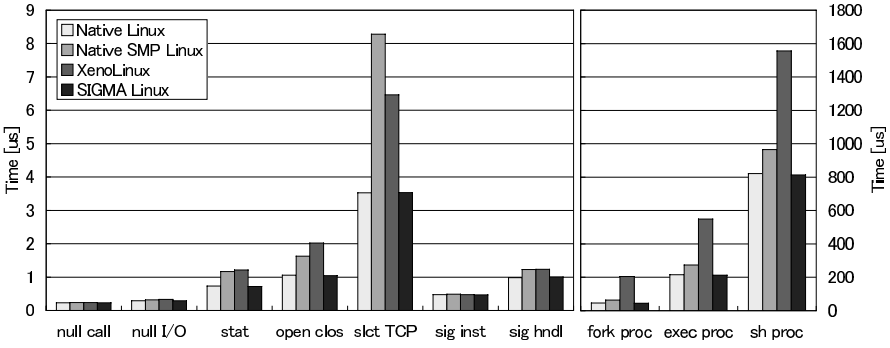


Fig. 2. LMbench: Processes - time in microseconds

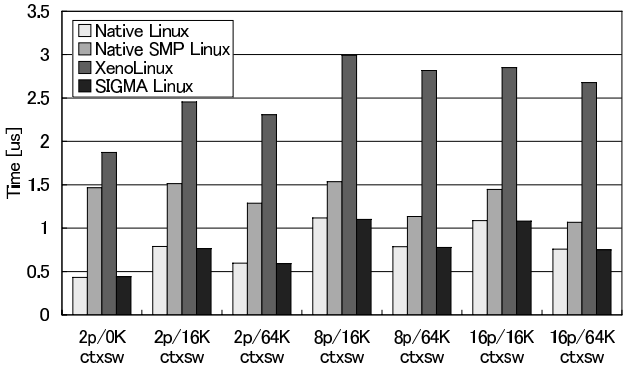


Fig. 3. LMbench: Context switching - time in microseconds

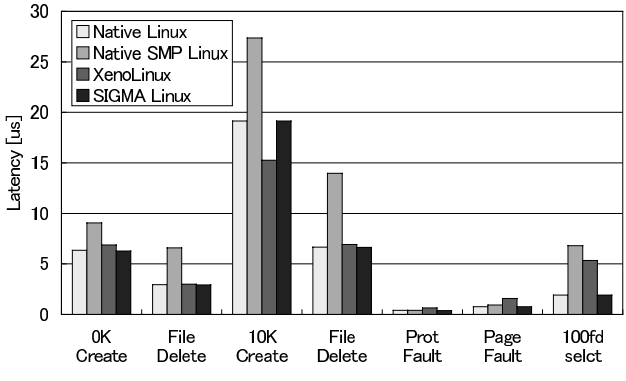


Fig. 4. LMbench: File & VM system latencies in microseconds

On the other hand, SIGMA Linux shows as fast performance as native Linux. The most significant advantage of avoiding performance degradation is that the guest OS can handle the privileged instructions directly. The configurations of page tables and interrupt vector tables, enabling/disabling interrupts, device controls require privileged instructions and those are usually performed by emulations in general virtualization approaches. Though, SIGMA Linux hardly requires such emulations except in using device drivers run on the other OSes. In addition, the VMM of general virtualization manages the guest OSes and usually handles scheduling of the guest OSes. In other words, the VMM must decide one of the guest OS to run and save and restore all OS contexts. This context switches between OSes are similar to that of processes in general OS and may degrade system performance. In SIGMA system, multiple OSes run concurrently and scheduling of OSes is not required at all. The superiority of SIGMA Linux in performance aspects is based on the architecture of SIGMA Linux which is almost same as that of native Linux.

4.2 Memory Access Collision

Memory accessing can conflict at a memory bus in shared memory multiprocessor architecture. The conflict causes the performance degradation even in SIGMA system. When more than two OSes access to the memory concurrently, while one OS performs memory related operations, the others can be waited for it to become available. We call such conditions memory access collisions and the performance may become slow in such cases. Fortunately, many kinds of multiprocessors have cache memory in each processor and a processor can access cache memory much faster than memory. If a process accesses data exists in cache memory, the read/write operation becomes faster. On the other hand, if the data isn't in cache memory, the process has to access main memory. Such operations are expensive compared to the cache memory. Thus, the results of memory latencies depend on whether the target data is in cache or not. We measured the latencies with memory access collision in five different conditions described in Table 1. The main purpose of the evaluation is to survey how much affection there are from activities of other OSes which share same memory bus.

The five conditions are categorized by cache availability and memory stress. "No Cache" and "Both No Cache" condition mean that caches in each processor

Table 1. The measurements condition of memory access collision in SIGMA system

Condition	Host OS		Guest OS
	Cache	Stress	Cache
Normal	enabled	no	enabled
No Cache	no	no	enabled
No Cache with Stress	no	stressed	enabled
Both No Cache	no	no	no
Both No Cache with Stress	no	stressed	no

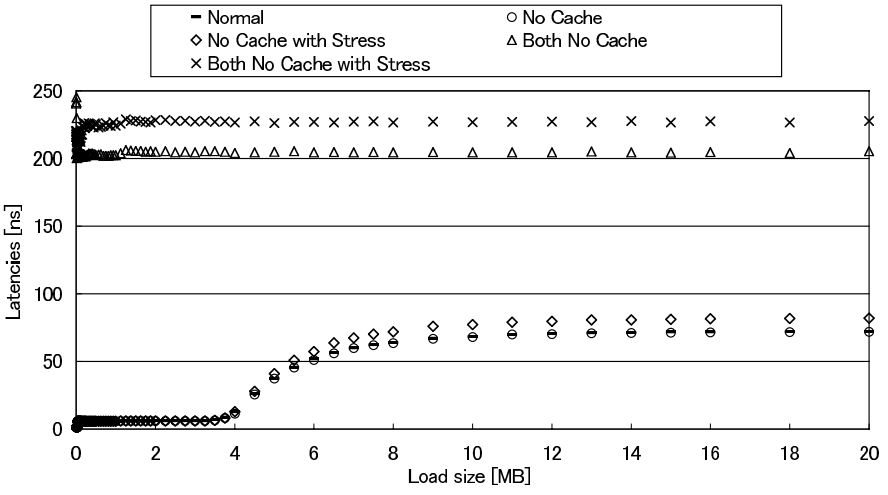


Fig. 5. LMbench: lat_mem_rd: Memory read latencies in the guest OS with memory collisions

are disabled. In "Stress", to generate loading condition, the program which only repeat read and write to memory is run on the host OS, Wombat. In that case, the memory access of the guest OS in "No Cache" condition can be sometimes affected. In "No Cache with Stress" condition, the delay is more increased because the many access from the host OS occurs. In "Both No Cache" and "Both No Cache with Stress" condition, the memory collision is assumed to be much more increased because all processors try to use the memory bus in reading or writing to the memory. The evaluations are performed on the guest OS and we used LMbench as an evaluation tool. The results are illustrated in Figure 5.

Figure 5 shows that, in both cache enabled and disabled configurations, the performance of the guest OS in loading condition degrades ten to fifteen percents compared to that in unloading. In addition, when loading size is below four megabytes, the latencies of "No Cache" and even "No Cache with Stress" conditions are same as the "Normal". That is, no performance degradation occurs below four megabytes when the caches are enabled on the target processor.

In general IA-32 architecture, processors and memory are connected by the 32-bit system bus, and the bus is shared by all processors on the system. Therefore, if multiple processors try to access the memory, the memory access may conflict. In the architecture, memory reading/writing is performed in accordance with memory ordering model of the processor, which is the order in which the processor issues reads and writes through the system bus to system memory. For a example, in write ordering of multiprocessor system, although each processor is guaranteed to perform writes in program order, writes from all processors are not guaranteed to occur in a particular order [7]. Therefore, if the system memory becomes loaded condition, multiple processors may access to the memory at

the same time and actually waiting time can be increased for each processor. As a result, those conflicts caused such delay shown in Figure 5.

On the other hand, the reason why such delay didn't occur in loading four megabytes or less is related to L2 cache size of the processor. If the loading data is in the cache, the processor need not to access the memory and the accessing does not affected by the memory ordering. Therefore, if the each processor has more caches, the possibility of the cache hit increases and it minimizes the delay by the memory access competition, as a result.

4.3 Engineering Cost

We compared the number of modified lines of each guest OS to measure the development cost quantitatively. One of the principle purpose of SIGMA system is to minimize the engineering cost of the guest OS as small as possible. The results are shown in Table 2.

Table 2. The engineering cost of the guest OSes

System	Modified Lines
SIGMA Linux	25
XenoLinux	1441
L4Linux	6500

Table 2 indicates that the modification cost of SIGMA Linux is much smaller than that of XenoLinux and L4Linux. A significant reason is that SIGMA Linux has the almost same structure as native Linux. That is, SIGMA Linux can execute the instructions that control the hardware directly, and such instructions need not to be substituted with emulated codes and hypervisor calls. The required modifications in SIGMA Linux are the memory related issues described in 3.3 and interrupt configurations and hardware checking parts.

5 Discussion

5.1 Limited Memory Protection

SIGMA system can't support memory protection among host and guest OSes because those OSes run in the privileged mode. This means that each OS can change memory mapping to the MMU of each processor and invade the memory regions where the other OSes manage. Since a VMM of virtualization technologies emulates updating memory mappings, the guest OS can be restricted not to update them by itself to preventing such illegal accessing. On the other hand, SIGMA system doesn't have any methods to control this kinds of illegal access because all OSes run in the privileged mode.

5.2 Estimated Use Cases

The advantages of the SIGMA system are that multiple OSes can run on it concurrently and their performance is as fast as native Linux. The main disadvantage is that SIGMA system has a security problem that its OSes can't be protected each other. Then, we will introduce two systems considered for both the advantages and the disadvantages.

For an Embedded System. If an embedded system is once shipped as a product, it is difficult to update its software such as OS and applications. Generally speaking, software of the embedded system is reliable because it is thoroughly verified. Therefore, even if memory protection is not supported sufficiently like SIGMA system, it is possible to introduce SIGMA system to some embedded fields. If there are bugs in OSes or device drivers and the system becomes unstable or crashes at worst, it is also critical condition for existing embedded systems. In other words, even general embedded OS can't ensure that the system continues to work properly in such condition. It is practical to adopt SIGMA system for embedded systems as long as their software is tested sufficiently. The advantages to introduce the SIGMA system to embedded systems are following: inexpensive general-purpose multiprocessors are available in the system, the system makes it easy to reuse the hardware and software by introducing general-purpose hardware platforms.

For a Server System. When SIGMA system is used as a server, it is possible to achieve secure and high performance server environment by being incorporated with secure hypervisors.

As illustrated in Figure 6, a secure hypervisor and its guest OS are performed on some processors. The OS has network interfaces and communicates with the other computers. The other OSes which don't depend on the hypervisor run on other processors. Those guest OSes can handle performance-oriented applications without accessing networks.

The OS runs on the hypervisor performs in the unprivileged mode. The OS can never affect the other OSes in the system because the OS is completely managed by the hypervisor. Therefore, it is suitable to run unreliable applications or

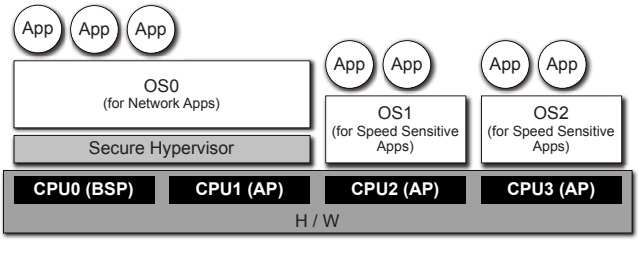


Fig. 6. One of estimated use cases - for a server system

networking applications on the OS which are always faced with the attack from malicious users. That is, even if the OS are accessed illegally by hitting its bugs or the administrator account of the OS is taken over, the OS can't access the other OSes beyond its own region.

The guest OSes which run on the hardware directly without any hypervisors, OS1 and OS2 in Figure 6, show fast performance, though they have security problems. In other words, such OSes can spoil the other OSes or the hypervisor. Therefore, unreliable and external applications should not be executed on such OSes and it is desirable to reboot the guest OSes periodically to keep them stable. Some applications estimated to be executed on the guest OSes are relatively heavy numeric calculation programs and database management programs and so on. It is possible for an OS runs on the hypervisor to ask the guest OSes to execute such heavy applications by using IOSCs.

As described above, combing SIGMA system with a secure hypervisor which are used in virtualization technologies, both software estimated to be used in reliable condition and performance-sensitive software can be available on the system.

6 Conclusion

We proposed asymmetric multiple OSes environment named SIGMA system on symmetric multiprocessors. Asymmetric means that there are two types of OS run on different processors: L4 microkernel and their servers including OS manager are performed as a host OS, and modified Linux named SIGMA Linux is performed as a guest OS. SIGMA system can avoid performance degradation of SIGMA Linux because SIGMA Linux can control hardware directly without any virtualization.

The evaluation clearly showed that SIGMA system performed much faster than the virtualization techniques such as Xen and almost same performance as native Linux. SIGMA system can be achieved with minimum overheads because no emulations are required to handle privileged processing. We also measured the latency of SIGMA Linux in memory loading condition. The results showed that SIGMA Linux was little affected in the condition. In addition, its engineering cost is much smaller than that of other para-virtualization approaches.

Consequently, the experiments proved that SIGMA system was a practical approach for a performance and engineering cost sensitive system. We believe that SIGMA system is adoptable to a server system and an embedded system.

References

1. Allen, E., Chase, D., Hallett, J., Luchangco, V., Maessen, J.-W., Ryu, S., Steele Jr., G.L., Tobin-Hochstadt, S.: The Fortress Language Specification Version 1.0 beta. Sun Microsystems, Inc. (March 2007)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the 19th ACM symposium on Operating systems principles, pp. 164–177. ACM Press, New York (2003)

3. Goldberg, R.: Survey of Virtual Machine Research. IEEE Computer 7(6), 34–45 (1974)
4. Härtig, H., Hohmuth, M., Liedtke, J., Schönberg, S., Wolter, J.: The Performance of Microkernel-Based Systems. In: SOSP 1997: Proceedings of the 16th ACM symposium on Operating systems principles, pp. 66–77. ACM Press, New York (1997)
5. Heiser, G.: Iguana User Manual, 4 (2005)
6. Intel Corporation. MultiProcessor Specification Version 1.4 (May 1997)
7. Intel Corporation. IA-32 Intel Architecture Software Developer's Manual (June 2006)
8. Kuz, I.: L4 User Manual NICTA L4-embedded API (October 2005)
9. Leslie, B., van Schaik, C., Heiser, G.: Wombat: A Portable User-Mode Linux for Embedded Systems. In: Proceedings of the 6th Linux.Conf.Au, Canberra (2005)
10. LeVasseur, J., Uhlig, V., Chapman, M., Chubb, P., Leslie, B., Heiser, G.: Pre-Virtualization: Slashing the Cost of Virtualization. Technical Report 2005-30, Fakultät für Informatik, Universität Karlsruhe (TH) (November 2005)
11. LeVasseur, J., Uhlig, V., Leslie, B., Chapman, M., Heiser, G.: Pre-Virtualization: Uniting Two Worlds (October 23–26, 2005)
12. Liedtke, J.: Improved Address-Space Switching on Pentium Processors by Transparently Multiplexing User Address Spaces. Technical Report 933, GMD - German National Research Center for Information Technology (September 1995)
13. McVoy, L.W., Staelin, C.: lmbench: Portable Tools for Performance Analysis. In: USENIX Annual Technical Conference, pp. 279–294 (1996)
14. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Commun. ACM 17(7), 412–421 (1974)
15. Rosenblum, M., Garfinkel, T.: Virtual Machine Monitors: Current Technology and Future Trends. Computer 38(5), 39–47 (2005)
16. Sugerman, J., Venkitachalam, G., Lim, B.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In: USENIX Annual Technical Conference, pp. 1–14 (2001)
17. Uhlig, V., Dannowski, U., Skoglund, E., Haeberlen, A., Heiser, G.: Performance of Address-Space Multiplexing on the Pentium. Interner Bericht 2002-01, Fakultät für Informatik, Universität Karlsruhe (2002)
18. University of Karlsruhe Germany and University of New South Wales and National ICT Australia. Afterburning and the Accomplishment of Virtualization (April 2005)
19. Waldspurger, C.: Memory Resource Management in VMware ESX Server. ACM SIGOPS Operating Systems Review 36(si), 181 (2002)
20. Wiggins, A., Tuch, H., Uhlig, V., Heiser, G.: Implementation of Fast Address-Space Switching and TLB Sharing on the StrongARM Processor. In: Proceedings of the 8th Asia-Pacific Computer Systems Architecture Conference, Aizu-Wakamatsu City, Japan, September 23–26 (2003)

An Efficient Code Generation Algorithm for Code Size Reduction Using 1-Offset P-Code Queue Computation Model

Arquimedes Canedo, Ben A. Abderazek, and Masahiro Sowa

Graduate School of Information Systems
University of Electro-Communications
1-5-1 Chofugaoka, Chofu-Shi 182-8585
Tokyo, Japan

Abstract. Embedded systems very often demand small memory footprint code. A popular architectural modification to improve code density in RISC embedded processors is to use a dual instruction set. This approach reduces code size at the cost of performance degradation due to the greater number of reduced width instructions required to execute the same task. We propose a novel alternative for reducing code size by using a single reduced instruction set queue machine. We present a efficient code generation algorithm to insert additional instructions to be able to execute programs in the reduced instruction set. Our experiments show that the insertion of additional instructions is minimal and we demonstrate improved code size reduction of 16% over MIPS16, 26% over Thumb, and 50% over MIPS32 code. Furthermore, we show that our compiler without any optimization is able to extract about the same parallelism than fully optimized RISC code.

1 Introduction

One of the major concerns in the design of an embedded RISC processor is the code density. Code density is tightly related to performance, energy consumption, and ROM memory utilization. A well known design strategy to reduce code size in RISC architectures is allowing the processor to execute two different instruction sets [5,8,17]. These dual instruction set architectures provide a 32-bit instruction set, and a *reduced instruction set* of 16-bit. The idea is to shorten the instruction length to improve the code size as well as fetching efforts as two short instructions can be read instead of one at the same cost. As the reduced instruction set has width limitation, more 16-bit instructions are needed to execute a given task. Since the 16-bit code requires more instructions than the 32-bit code to execute the same task, the performance of the reduced instruction set code is lower than the 32-bit code. The ARM-Thumb [5] and MIPS16 [8] are examples of dual instruction sets. In [5], a 30% of code size reduction with a 15% of performance degradation is reported for the ARM-Thumb processor.

Compiler support for dual instruction set architectures is crucial to maintain a balance between code size reduction and performance degradation due to the

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 196–208, 2007.

increase in number of instructions. Different researches have been proposed to cope with this problem. Profile guided compiler heuristics at function level granularity have been proposed in [9] to determine which code, from the two available, maximizes the code size reduction without causing a significant loss in performance. Halambi et.al. proposed in [6] a compiler algorithm at instruction level granularity based on a profitability analysis function discerning between code size and performance. A similar approach based on a path-based profitability analysis is proposed in [15]. Kwon et.al. in [12], proposed a compiler-architecture interaction scheme to compress even more the ARM-Thumb instructions by using a partitioned register file and an efficient register allocation algorithm. In [10,11], Krishnaswamy and Gupta proposed a compiler-architecture approach where the compiler identifies pairs of 16-bit instructions that can be combined safely into a single 32-bit instruction and replaces it with augmenting instructions that are later coalesced by the hardware at zero performance penalty.

An attractive alternative to reduce code size is using a queue-based computation model. The queue computation model uses a FIFO data structure, analogous to a register file, to perform computations [3]. All accesses to the FIFO queue, named operand queue, follow the rules of enqueueing and dequeueing. These accesses are done at the rear, and head of the queue, respectively. Thus, instructions are free of location names from where to dequeue or enqueue their operands. Having only the instruction itself allows the instruction set to be short.

In our previous work we have investigated and designed a Parallel Queue Processor (PQP) based on the Queue Computation Model [16,1,2]. The PQP breaks the rule of dequeueing to make a better utilization of the data in the operand queue. This feature leads to better performance and shorter programs. PQP instructions allows operands to be dequeued from a specified position with respect of the head of the queue. Thus, PQP instruction set format requires the encoding of the offset references making it longer than the instruction set for a traditional queue machine. We found in [4], that for a set of scientific programs the instructions that require two offsets are almost nonexistent. Below 10% of the instructions require only one offset reference, and above 90% of remaining instructions require no offsets to be specified. As queue programs require at most one offset, we propose to reduce the number of offsets encoded in the instructions of the PQP from two to one. Having only one offset reference to be encoded in the instruction set saves bits in the instruction format making instructions shorter. In this paper, we present a compiler and architecture support to reduce code size using a reduced PQP instruction set. An efficient code generation algorithm for 1-offset P-Code QCM is described in detail. Our algorithm is able to reduce code size while keeping the instruction count increase very low since the queue instructions are free of false dependencies. To our best knowledge, code generation algorithms for queue machines have not been proposed by previous published work.

The remainder of this paper is organized as follows: Section 2 gives an overview of the queue computation model, the queue compiler infrastructure,

and describes the 1-offset P-Code QCM. Section 3 describes in detail the phases of the code generation algorithm for 1-offset P-Code QCM. We present the results of our approach in Section 4. We discuss our results in Section 5, and conclude in Section 6.

2 1-Offset P-Code Queue Computation Model

Queue Computation Model (QCM) refers to the evaluation of expressions using a first-in first-out queue, called *operand queue*. This model establishes two rules for the insertion and removal of elements from the operand queue. Operands are inserted, or enqueued, at the rear of the queue. And operands are removed, or dequeued, from the head of the queue. Two references are needed to track the location of the head and the rear of the queue. The *Queue Head*, or `QH`, points to the head of the queue. And *Queue Tail*, or `QT`, points to the rear of the queue.

A program for the QCM is obtained from a breadth-first traversal of the expressions' parse tree [3]. Figure 1 shows the parse tree of expression $x = (a + b)/(a - b)$ and the resulting *queue program* after a breadth-first traversal of the parse tree. The first four instructions enqueue the operands a, b, a, b from memory into the operand queue. At this stage `QH` points to the first loaded operand (a), and `QT` points to an empty location after the last operand (b). The addition instruction `add` requires two operands to be dequeued (a and b). `QH` now points to the third operand originally loaded (a). After the addition is computed the result is written to `QT`. The rest of the instructions have the same behavior until the expression has been completely evaluated and the result (x) is dequeued to memory.

In our early work [4], we have proposed a *2-offset P-Code* QCM to allow programs to be generated directly from their directed acyclic graphs (DAGs). 2-offset P-Code QCM, strictly follows enqueueing rule but has flexibility in the dequeueing rule of operands. Operands can be taken from `QH` or from a different position other than `QH`. The desired position of the datum that should be dequeued is specified as an offset with respect of the position of `QH`.

Figure 2 shows the DAG and the 2-offset P-Code program for the same expression shown in Figure 1, $x = (a + b)/(a - b)$. Notice that the program

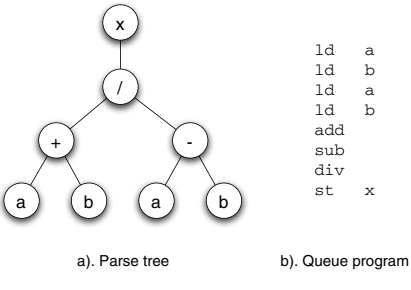


Fig. 1. Queue program generation from an expression's parse tree

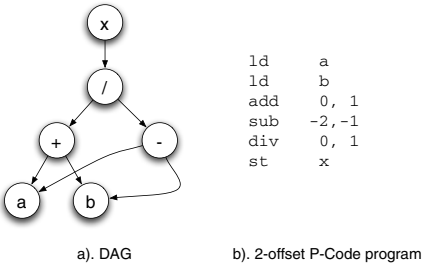


Fig. 2. 2-offset P-Code program generation from a directed acyclic graph

in Figure 2.b contains only one enqueue operation for operand a, and one for operand b instead of two as in Figure 1. After operands have been enqueued the queue contents are: a, b. The "`add 0, 1`" instruction has two offset references, one for each of its operands. The first offset, `0`, indicates that the first operand should be taken from a zero distance from `QH`, or `QH`+0. That is, from `QH` itself. The second offset reference, `1`, indicates that the second operand should be taken from a distance one from `QH`, or `QH`+1. After the `add` instruction dequeues its first operand, a, the `QH` is updated and points to the next queue location, in this case, to the operand b. After the second operand for instruction `add` is dequeued, operand b, `QH` is updated and points to an empty location after operand b. The `add` instruction is performed with operands a, b and the result, $a + b$, is written back to `QT`, and `QT` is updated. At this stage, `QH` points to $a + b$, and `QT` to an empty location after $a + b$. Next, the "`sub -2, -1`" instruction takes its first operand from a distance of `-2` from `QH`, or `QH`−2. A negative offset indicates that the operand should be taken from the data in the operand queue that has been used before by other operations. For this case `QH`−2 points to the already used operand a. The second operand for `sub` is taken from `QH`−1, the used operand b. This instruction takes its two operands from a different location than `QH`, therefore, `QH` pointer is not updated and remains pointing to $a + b$ after the `sub` instruction is executed. At this point, the contents of the queue are: $a + b, a - b$. The following instructions are executed in similar fashion until the final expression (x) has been computed entirely.

Together with the 2-offset P-Code QCM we developed the compiler infrastructure and code generation algorithms [4]. Our queue compiler uses GCC 4.0.2 front-end. The custom back-end of our compiler takes GIMPLE trees [13,14] as input and generates assembly code for the PQP. GIMPLE intermediate representation is first expanded into an unrestricted trees named QTrees. The difference between GIMPLE and QTrees is that QTrees do not have limitation in the number operands an expression can hold [13]. QTrees are very similar to GENERIC trees [13] but QTrees are generated after all GCC's tree optimizations have been applied to GIMPLE form. QTrees are then passed to a leveling function that creates a *leveled DAG* (LDAG) [7]. A LDAG is a data structure chosen to be the input to the code generation algorithms for the queue compiler due to its expressiveness in the relations hip between operations, operands, and

the queue computation model. The code generation algorithm takes LDAGs to perform the offset calculations for the operations in the program. After all P-Code offset references are computed, the code generation algorithm produces a linear low level intermediate representation named QIR. QIR is a one operand intermediate representation suitable for generating final assembly code for the PQP. The last phase of the compiler takes QIR and generates the final assembly code for the PQP. Figure 3 shows the structure of the queue compiler.

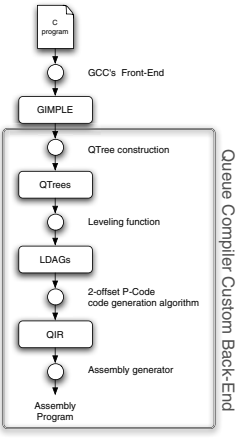


Fig. 3. Queue Compiler Infrastructure

We conducted experiments to measure the distribution of P-Code instructions required by some scientific test programs and the 2-offset P-Code QCM [4]. We found that the instructions that require the two operands to be dequeued from a different position than `QH` (2-offset instructions) appear rarely in programs. Instructions that require only one operand to be dequeued from an offset reference from `QH` (1-offset instructions), and instructions that take their operands directly from `QH` (0-offset instructions) are the most common instructions in programs. Based on these experiment results, we propose a 1-offset P-Code QCM. 1-offset P-Code QCM restricts all operations to have at most one offset reference. This limitation forces binary operations to take one of the operands always from `QH`, and the other operand can be taken from a location different from `QH`. Unary operations do not suffer from this limitation since they follow the rule to have at most one offset reference and the only operand can be taken with an offset reference.

It is impossible for the 1-offset P-Code QCM to execute an instruction where both operands are away from `QH`. To solve this problem, a special operation named `dup` instruction is added to the 1-offset P-Code's instruction set. The semantics of the `dup` instruction is to copy the value of an operand in the operand queue to `QT`. `dup` instruction takes one operand which is an offset reference. The offset reference is used to indicate the position with respect of `QH` of the operand

to be copied to QT. Figure 4.a shows the DAG for the expression $x = (a+b)/(a*a)$. 1-offset P-Code QCM cannot execute the multiplication node since both of its operands are away from QH by the time the operation is evaluated. Figure 4.b shows the DAG for the same expression augmented with dup instruction node. The effect of the dup node is to create a copy of operand a to the place where the dup node appears as shown with the dashed arrow. Figure 4.c lists the resulting 1-offset P-Code program. Operands a, b are enqueued. The offset for the dup instruction is zero indicating that the operand that has to be copied is placed in QH itself. After dup is executed, the contents of the operand queue are: $a, b, a\prime$. Where $a\prime$ is the copy of a produced by dup. QH points to the first operand, a, and QT points to an empty location after the operand $a\prime$. add 0 instruction takes both operands from QH. Then mul instruction takes its first operand from QHthat at this stage points to $a\prime$, and its second operand, a, from a distance of -2 with respect of QH. The rest of the program is executed similarly.

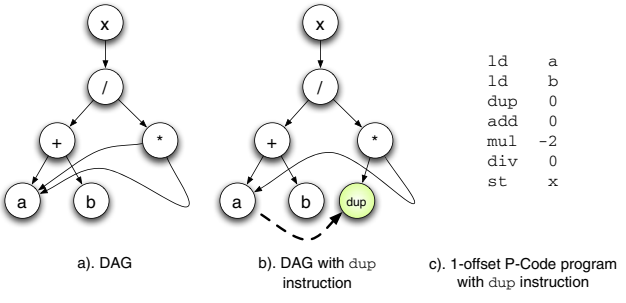


Fig. 4. 1-offset P-code program generation from a directed acyclic graph

1-offset P-Code QCM requires a new algorithm capable to determine the correct location of dup instructions. In the next section we will describe in detail the code generation algorithm for 1-offset P-Code.

3 Code Generation Algorithm

The algorithm works in two stages during code generation. The first stage converts QTrees to LDAGs augmented with *ghost nodes*. A ghost node is a node without operation that serves as a mark for the algorithm. The second stage takes the augmented LDAGs and assigns dup instructions to the ghost nodes when necessary. Finally, a breadth-first traversal of the LDAGs with dup nodes computes the offset references for all instructions and generates QIR as output. Figure 5 shows the two stages for the proposed code generation algorithm represented by the numbered circles. The elements inside the dashed area are the modified modules with respect of the original queue compiler [4] shown in Figure 3.

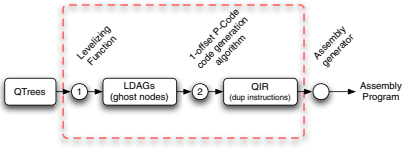


Fig. 5. 1-offset P-Code code generation compiler phases

3.1 Augmented LDAG Construction

QTrees are transformed into LDAGs by the leveling function. Algorithm 1 lists the leveling function that outputs an augmented LDAG with ghost nodes. The algorithm makes a post-order depth-first recursive traversal over the QTree. All nodes are recorded in a lookup table when they first appear, and are created in the corresponding level of the LDAG together with its edge to the parent node. Two restrictions are imposed over the LDAGs for the 1-offset P-Code QCM.

Definition 1. *The sink of an edge must be always in a deeper or same level than its source.*

Definition 2. *An edge to a ghost node spans only one level.*

When an operand is found in the lookup table the Definition 1 must be kept. Line 5 in Algorithm 1 is reached when the operand is found in the lookup table and it has a shallow level compared to the new level. The function `dag_ghost_move_node()` moves the operand to the new level, updates the lookup table, converts the old node into a ghost node, and creates an edge from the ghost node to the new created node. The function `insert_ghost_same_level()` in Line 8 is reached when the level of the operand in the lookup table is the same to the new level. This function creates a new ghost node in the new level, makes an edge from the parent node to the ghost node, and an edge from the ghost node to the element matched in the lookup table. These two functions build LDAGs augmented with ghost nodes that obey Definitions 1 and 2. Figure 6 illustrates the result of leveling the QTree for the expression $x = (a * a)/(-a + (b - a))$. Figure 6.b shows the resulting LDAG augmented with ghost nodes.

3.2 `dup` Instruction Assignment and Ghost Nodes Elimination

The second stage of the algorithm works in two passes as shown in Lines 4 and 7 in Algorithm 2. Function `dup_assignment()` decides whether ghost nodes are substituted by `dup` nodes or eliminated from the LDAG. Once all the ghost nodes have been transformed or eliminated, the second pass performs a breadth-first traversal of the LDAG, and for every instruction the offset references with respect of `QH` are computed in the same way as in [4]. The output of the code generation algorithm is QIR for 1-offset P-Code.

The only operations that need a `dup` instruction are those binary operations whose both operands are away from `QH`. The augmented LDAG with ghost nodes

Algorithm 1. dag_levelize_ghost (tree t, level)

```
1: nextlevel ⇐ level + 1
2: match ⇐ lookup (t)
3: if match ≠ null then
4:    if match.level < nextlevel then
5:       relink ⇐ dag_ghost_move_node (nextlevel, t, match)
6:       return relink
7:    else if match.level = lookup (t) then
8:       relink ⇐ insert_ghost_same_level (nextlevel, match)
9:       return relink
10:   else
11:      return match
12:   end if
13: end if
14: /* Insert the node to a new level or existing one */
15: if nextlevel > get_Last_Level() then
16:   new ⇐ make_new_level (t, nextlevel)
17:   record (new)
18: else
19:   new ⇐ append_to_level (t, nextlevel)
20:   record (new)
21: end if
22: /* Post-Order Depth First Recursion */
23: if t is binary operation then
24:   lhs ⇐ dag_levelize_ghost (t.left, nextlevel)
25:   make_edge (new, lhs)
26:   rhs ⇐ dag_levelize_ghost (t.right, nextlevel)
27:   make_edge (new, rhs)
28: else if t is unary operation then
29:   child ⇐ dag_levelize_ghost (t.child, nextlevel)
30:   make_edge (new, child)
31: end if
32: return new
```

facilitate the task of identifying those instructions. All binary operations having ghost nodes as their left and right children need to be transformed as follows. The ghost node in the left children is substituted by a `dup` node, and the ghost node in the right children is eliminated from the LDAG. For those binary operations with only one ghost node as the left or right children, the ghost node is eliminated from the LDAG. Algorithm 3 describes the function `dup_assignment()`. The effect of Algorithm 3 is illustrated in Figure 7. The algorithm takes as input the LDAG with ghost nodes shown in Figure 6.b and performs the steps described in Algorithm 3 to finally obtain the LDAG with `dup` instructions as shown in Figure 7.a. The last step in the code generation is to perform a breadth-first traversal of the LDAG with `dup` nodes and compute for every operation, the offset value with respect of `QH`. `dup` instructions are treated as unary instructions by

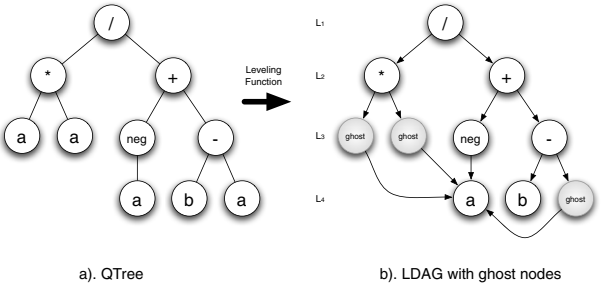


Fig. 6. Leveling of QTree into augmented LDAG for expression $x = \frac{a \cdot a}{-a+(b-a)}$

Algorithm 2. codegen ()

```
1: for all basic blocks BB do
2:    for all expressions W_k in BB do
3:       for all instructions I_j in TopBottom (W_k) do
4:          dup_assignment (I_j)
5:       end for
6:       for all instructions I_j in BreadthFirst (W_k) do
7:          p_qcm_compute_offsets (W_k, I_j)
8:       end for
9:    end for
10: end for
```

the offset calculation algorithm. The final 1-offset P-Code QIR for the expression $x = (a * a)/(-a + (b - a))$ is given in Figure 7.b.

3.3 Increase in Number of Instructions

A single `dup` instruction is inserted for every binary operation whose both operands are away from `QH` (β).

$$dup_i = \beta_i \tag{1}$$

The increase in number of instructions (Δ) for the 1-offset P-Code compared to 2-offset P-Code is given by the addition of `dup` instructions in the program.

$$\Delta = \sum_{i=1}^{n} (dup_i) \tag{2}$$

Thus, the total number of instructions for 1-offset P-Code ($Total$) is given by the total number of instructions for 2-offset P-Code (T_old) plus the inserted `dup` instructions (Δ):

$$Total = T\_old + \Delta \tag{3}$$

Algorithm 3. dup_assignment (i)

```
1: if isBinary (i) then
2:    if isGhost (i.left) and isGhost (i.right) then
3:       dup_assign_node (i.left)
4:       dag_remove_node (i.right)
5:    else if isGhost (i.left) then
6:       dag_remove_node (i.left)
7:    else if isGhost (i.right) then
8:       dag_remove_node (i.right)
9:    end if
10:   return
11: end if
```

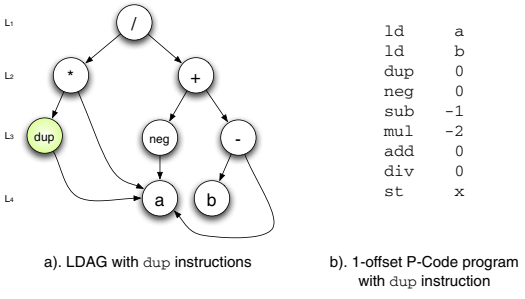


Fig. 7. 1-offset P-Code code generation from a LDAG

The length of the instruction set of the PQP is 2 byte [1]. The PQP has a special instruction, `covop`, which extends the value of the operand of the following instruction. The `covop` instructions are used to extend immediate values that are not representable with a single PQP 16-bit instruction. Thus, the code size for a 1-offset P-Code is obtained from the Equation 3 as:

$$Code\_Size = 2 * (Total + covop) \tag{4}$$

4 Experiments

We measured the code size for some benchmarks using our queue compiler for the PQP, and using GCC version 4.0.2 for MIPS32, MIPS16, 32-bit ARM, and Thumb architectures. We chose a set of recursive and iterative numerical computation benchmarks including the fast fourier transform, livermore loops, linpack, matrix multiplication, Rijndael encryption algorithm, etc. All programs were compiled without code reduction optimizations to estimate the real overhead of using a reduced instruction set architecture to reduce code size and compare it with our solution. Compiler based optimization techniques for improving code size on reduced instruction set architectures remains out of the scope of this paper.

Table 1. Code size comparison using GCC for different RISC architectures and the queue compiler for the PQP

Benchmark	ARM32	MIPS16	Thumb	PQP
quicksort.c	0.95	0.40	0.63	0.43
nqueens.c	0.85	0.53	0.78	0.52
md5.c	0.74	0.39	0.76	0.48
matrix.c	0.93	0.42	0.63	0.68
fft8g.c	1.02	0.92	0.60	0.54
livermore.c	1.16	0.74	0.80	0.58
vegas.c	1.11	0.89	0.73	0.51
whetstone.c	1.15	0.73	0.73	0.34
linpack.c	0.97	0.58	0.81	0.52
aes.c	0.83	0.51	0.67	0.38

For each benchmark we take the code size for MIPS32 as the baseline. Table 1 shows the normalized code size for all compiled benchmarks for ARM32 in column 2, MIPS16 in column 3, Thumb in column 4, and PQP in column 5 as compared to the baseline code size. GCC generates about the same code size for MIPS32 and ARM32 architectures with an average difference of 3%. For the MIPS16 and Thumb architectures, gcc reduces the code size for all benchmarks compared to the baseline MIPS32 code size. In average, for the MIPS16 it produces 42% smaller code size. For the Thumb it produces 24% smaller code size. We compiled the set of benchmarks for the PQP processor using our queue compiler. The presented results for the PQP take into account the extra `dup` instructions. Most of the programs require zero or one extra `dup` instruction except for `linkpack.c` which required six extra `dup` instructions. In average, our compiler technique produces 51% smaller code than the baseline code size. Our compiler is able to generate in average 16% smaller code than gcc for MIPS16, and 36% smaller code than gcc for Thumb architecture. For three of the benchmarks, `quicksort.c`, `md5.c`, and `matrix.c`, our compiler generated larger code compared to MIPS16. An inspection to the source of the programs revealed that these programs have a common characteristic of having functions with arguments passed by value. Our queue compiler handles these arguments sub-optimally as they are passed in memory. Therefore, additional instructions are required to copy the values to local temporary variables.

To compare the effect of breadth-first scheduling on parallelism, we compiled the benchmark programs and analyzed the compile-time parallelism exposed by the compiler. Our compiler at the moment does not include any optimization of any kind. For the RISC code we selected GCC-MIPS compiler and we enable all optimizations (-O3). Figure 8 shows the results of the experiment. Our compiler is able to extract about the same parallelism than fully optimized RISC code, in average, 1.07 times more. Our current and future work include the addition of optimization phases on the queue compiler infrastructure.

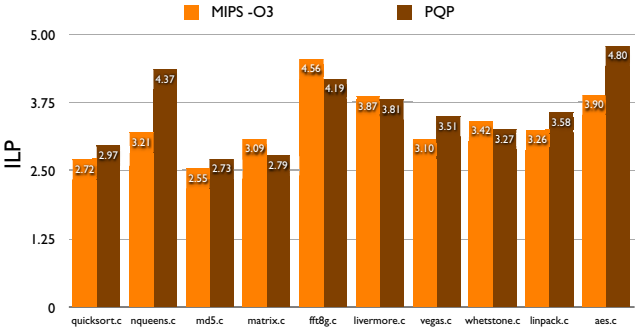


Fig. 8. Compile-time extracted instruction level parallelism

5 Discussion

The presented code generation algorithm efficiently reduces code size by using a 1-offset P-Code queue computation model processor. From the presented results, we observed that our technique reduces code size while keeping the instruction number increase very low. Techniques to reduce code size by using a dual instruction set [5,8] have the tradeoff that the increase in number of instructions, to relieve register pressure, leads to performance degradation of about 15%. In our technique, the increase in number of instructions comes from the insertion of `dup` instructions by the code generation algorithm. As presented in the experiments, and as found on our previous work [4], the additional instruction count is very low. The instructions of the PQP do not have register references making the programs free of false dependencies and, as a consequence, the need of spill code disappears. As future work, we will conduct experiments to measure the performance, in terms of execution time, of our technique. We shown here that the code for 1-offset P-Code QCM is about 50% denser than a full 32-bit RISC processor, and we expect it to be about the same performance since the increase of instructions is minimal. The benefits of 1-offset P-Code QCM are not just limited to code size. Since our technique introduces very small number of additional instructions, and the width of the instruction set is 16-bit, we expect less power consumption while accessing the memory to fetch instructions when compared to a fixed width 32-bit instruction set such as MIPS32.

6 Conclusion

In this paper we presented a code generation algorithm together with a 1-offset P-Code QCM for reducing code size. Our code generation algorithm has been integrated to the Queue compiler and the presented results have demonstrated the efficiency of the algorithm. The contributions of this paper can be summarized as follows: (1) the development of a new code generation algorithm for a 1-offset P-Code QCM using `dup` instructions and its integration to the queue compiler infrastructure; (2) the utilization of a 1-offset P-Code QCM to reduce

code size; (3) evidence that the queue-based computers are a practical alternative for systems demanding small code size and high performance. Our technique is able to generate in average 16% denser code than MIPS16, 26% denser code than Thumb, and 50% denser code than MIPS32 and ARM architectures. Without optimizations, the queue compiler is able to extract about the same parallelism than fully optimized code for a RISC machine.

References

1. Abderazek, B., Yoshinaga, T., Sowa, M.: High-Level Modeling and FPGA Prototyping of Produced Order Parallel Queue Processor Core. Journal of Supercomputing, 3–15 (2006)
2. Abderazek, B., Kawata, S., Sowa, M.: Design and Architecture for an Embedded 32-bit QueueCore. Journal of Embedded Computing 2(2), 191–205 (2006)
3. Bruno, R., Carla, V.: Data Flow on Queue machines. In: 12th Int. IEEE Symposium on computer Architecture, pp. 342–351 (1985)
4. Canedo, A.: Code Generation Algorithms for Consumed and Produced Order Queue Machines, University of Electro-Communications, Master Thesis (2006), http://www2.sowa.is.uec.ac.jp/~canedo/master_thesis.pdf
5. Goudge, L., Segars, S.: Thumb: Reducing the Cost of 32-bit RISC Performance in Portable and Consumer Applications. In: Proceedings of COMPCON 1996, pp. 176–181 (1996)
6. Halambi, A., Shrivastava, A., Biswas, P., Dutt, N., Nicolau, A.: An Efficient Compiler Technique for Code Size Reduction using Reduced Bit-width ISAs. In: Proceedings of the Conference on Design, Automation and Test in Europe, p. 402 (2002)
7. Heath, L., Pemmaraju, S., Trenk, A.: Stack and Queue Layouts of Directed Acyclic Graphs. SIAM Journal of Computing 28(4), 1510–1539 (1999)
8. Kissel, K.: MIPS16: High-density MIPS for the embedded market, Technical report, Silicon Graphics MIPS Group (1997)
9. Krishnaswamy, A., Gupta, R.: Profile Guided Selection of ARM and Thumb Instructions. In: ACM SIGPLAN conference on Languages, Compilers, and Tools for Embedded Systems, pp. 56–64 (2002)
10. Krishnaswamy, A., Gupta, R.: Enhancing the Performance of 16-bit Code Using Augmenting Instructions. In: Proceedings of the, SIGPLAN Conference on Language, Compiler, and Tools for Embedded Systems, 2003, pp. 254–264 (2003)
11. Krishnaswamy, A.: Microarchitecture and Compiler Techniques for Dual Width ISA Processors, University of Arizona, Ph.D Dissertation (2006), http://cs.arizona.edu/~gupta/Thesis/arvind.pdf
12. Kwon, Y., Ma, X., Jae Lee, H.: PARE: instruction set architecture for efficient code size reduction. Electronics Letters, 2098–2099 (1999)
13. Merrill, J.: GENERIC and GIMPLE: A New Tree Representation for Entire Functions. In: Proceedings of GCC Developers Summit, pp. 171–180 (2003)
14. Novillo, D.: Design and Implementation of Tree SSA. In: Proceedings of GCC Developers Summit, pp. 119–130 (2004)
15. Sheayun, L., Jaejin, L., Min, S.: Code Generation for a Dual Instruction Processor Based on Selective Code Transformation. Lectures in Computer Science, pp. 33–48 (2003)
16. Sowa, M., Abderazek, B., Yoshinaga, T.: Parallel Queue Processor Architecture Based on Produced Order Computation Model. Journal of Supercomputing, 217–229 (2005)
17. SuperH RISC Engine, http://www.superh.com

Interconnection Synthesis of MPSoC Architecture for Gamma Cameras

Tianmiao Wang[1], Kai Sun[1], Hongxing Wei[1], Meng Wang[2], Zili Shao[2,⋆], and Hui Liu[3]

[1] Robot Research Institute, Beihang University, Beijing 100083, China
{wtm,mounthorse,whx}@me.buaa.edu.cn
[2] Department of Computing, The Hong Kong Polytechnic University, Hong Kong
{csmewang,cszlshao}@comp.polyu.edu.hk
[3] Software Engineering Institute, Xidian University, Xi'an, China
liuhui@xidian.edu.cn

Abstract. MPSoC (Multi-Processor System-on-Chip) architecture is becoming increasingly used because it can provide designers much more opportunities to meet specific performance and power goals. In this paper, we use MPSoC architecture to solve real-time signal processing problem in gamma camera. We propose an interconnection synthesis algorithm to reduce the area cost of the Network-on-Chip for an MPSoC architecture we propose in [14]. We implement our interconnection synthesis algorithm on FPGA, and synthesize Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and satisfy the requirements of real-time signal processing for 256 × 256 image construction.

1 Introduction

MPSoC (Multi-Processor System-on-Chip) architecture is becoming increasingly used. It can provide high throughput while keeping power and complexity under control and give designers more opportunities to meet specific performance and power goals. With a heterogeneous MPSoC architecture, different types of cores can be built on the same die, and each type of core can effectively and efficiently process specific tasks. These flexible combinations make MPSoC systems very powerful and be able to implement complex functions with high performance and low power by integrating multiple heterogeneous processors, hierarchy memory systems, custom logic, and on-chip interconnection. With such advantages, MPSoC architecture has been widely applied in various fields such as network[10], multimedia[11] and HDTV[6]. We focus on the MPSoC design and synthesis for real-time signal processing for biomedical applications in this paper.

In biomedical applications, most medical electronic devices heavily rely on image processing techniques to process large scale data. Therefore, more powerful

⋆ The corresponding author.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 209–218, 2007.

techniques are needed in order to improve processing speed and precision. MPSoC architecture brings these systems more opportunities to achieve their goals. For example, in [8], Khatib et al. propose an application-specific MPSoC architecture for real-time ECG (Electrocardiogram) analysis. The advanced industrial components for MPSoC design (multi-issue VLIW DSPs, system interconnect from STMicroelectronics, and commercial off-the-shelf biomedical sensors) are employed in their architecture so real-time ECG analysis can be achieved with high sampling frequencies.

In this paper, we solve the real-time digital signal processing for gamma cameras, most commonly used medical imaging devices in nuclear medicine. In a gamma camera, images are generated by detecting gamma radiation. One of the key components in a gamma camera is PMT (PhotoMultiplier Tube). Basically, PMT is used to detect fluorescent flashes generated by a crystal and produce current. Then the corresponding voltage signals are converted to digital signals by ADC (Analog to Digital Converter) behind a PMT, and finally the digital signals are processed to generate images by a digital signal processing system. To generate images, multiple PMTs are placed in hexagon configurations behind the absorbing crystal. In a typical scheme, a PMT array consisting of more than 30 PMTs is used in a gamma camera. Using a serial 2D images obtained by gamma cameras from the different angles, 3D information can be acquired by SPECT (Single Photon Emission Computed Tomography).

To process the data generated by the PMT array, DSP (Digital Signal Processing) boards based on PC platforms are widely used in current gamma cameras. With such platforms, typically, it takes about 15 - 30 seconds to generate one 64 $\times$ 64 image and 15 - 20 minutes to finish a complete scan in SPEC. The platforms can not efficiently produce higher-quality images such as 256$\times$256. And their slow processing speed and big size limit the effective use of gamma cameras, in particular, for portable gamma cameras [12,13] that work with new room-temperature nuclear radiation detector. To improve image construction speed, a technique called PMT-PSPMT (Position Sensitive PhotoMultiplier Tube) [7] is proposed. PMT-PSPMT is very effective in optimizing image construction times. But it reduces the image quality and cannot construct 256 $\times$ 256 image dynamically.

To solve these problems, we propose an MPSoC architecture for PMT data processing in a gamma camera in [14]. Our MPSoC architecture consists of the following four parts: one general-purpose embedded processor, a high speed data interface (HSDI), application-specific DSP cores and a Network-on-Chip with an interconnection bus. In this paper, we develop an interconnection synthesis algorithm to reduce the area cost of the Network-on-Chip for the MPSoC architecture in [14]. We implement our interconnection synthesis algorithm with FPGA, and synthesize DSP cores and Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and implement communication with small area cost. It can satisfy the requirements of real-time signal processing for 256 $\times$ 256 image construction.

The rest of this paper is organized as follows: in Section 2, we introduce necessary backgrounds related to gamma camera technique. In Section 3, we present the MPSoC system architecture. Section 4 provides the experimental results and discussions. In Section 5, we conclude the paper.

2 Background

Gamma camera is a commonly used medical imaging device in nuclear medicine. In a gamma camera, images are generated by detecting gamma radiation. Basically, the counts of gamma photons that are absorbed by a crystal are accumulated, and the crystal produces a faint flash of light at the same time. The PMT array behind the crystal detects the fluorescent flashes and generates current. The current signal generated by the PMT is captured by the ADC, and two corresponding voltage signals are converted into digital signals. Digital signals are used to calculate the coordinate and energy of the gamma photons. With these coordinate and energy data, the final image can be produced.

During the whole medical imaging procedure, three algorithms, the integral, coordinate and amendment algorithms, are applied to the collected data.

The integral algorithm is to calculate the energy of the voltage signal. In this algorithm, the serial data of each PMT is accumulated based on system status conditions. The coordinate algorithm includes the calculation for two parts, position and energy. With the position and energy data, the gamma photon pulse can be determined. The amendment algorithm is used to amend energy and position data with three table-lookup operations. This algorithm consists of two parts, energy and linearity emendation.

To reduce the image construction time, we can increase the pulse frequency of gamma photons. With the limitation of the device, the maximum pulse frequency currently we can achieve is 500KHz-1 MHz. Correspondingly, we have to improve the speed of digital signal processing in order to generate image with such high pulse frequency. In this paper, our goal is to design an MPSoC architecture that can generate one 256×256 image in less than one second for gamma cameras with 1 MHz pulse frequency.

3 MPSoC System Design

In this section, we first introduce the MPSoC architecture in Section 3.1. Then we propose our interconnection synthesis in Section 3.2, respectively.

3.1 Architecture Overview

Our MPSoC architecture is a typical heterogeneous multi-core architecture targeting on the application of gamma camera. It is specially designed for processing PMT data in parallel with multi-processors. In order to achieve these goals, an MPSoC architecture, as shown in Figure 1 is proposed to speed up the image generation and improve image quality.

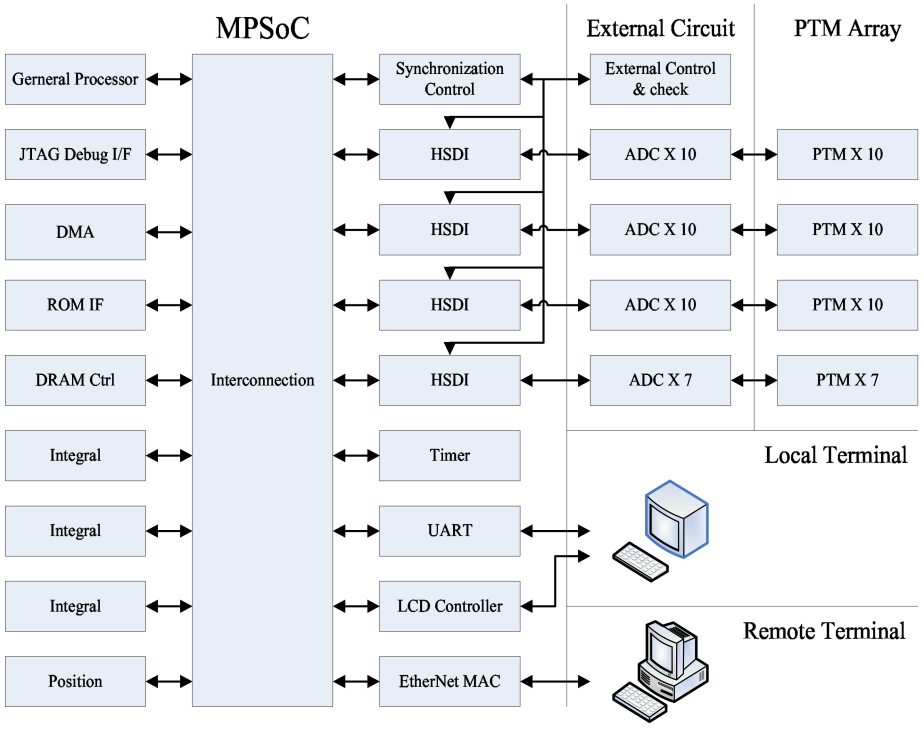


Fig. 1. The MPSoC Architecture

As shown in Figure 1, our MPSoC architecture consists of four parts: general processor, HSDI (High Speed Data Interface), DSP, and interconnection synthesis. In this architecture, the processor speed and the 32-bit on-chip interconnection are 200MHz, which are compatible with the 0.18um ASIC technology and the 32-bit bus interface IP cores. Next, we present the design issues for each key part of MPSoC architecture.

In the general processor part, there are one general purpose processor and some necessary IP cores, such as timer, UART, and SPI etc. Among these IP cores, the most important components are the on-chip RAM, SRAM/Flash controller, SDRAM controller and Ethernet MAC controller. The amendment algorithm and other general purpose computing are implemented in the general processor.

The customized DSPs used in our MPSoC architecture are designed for implementing the integral algorithm and the coordinate algorithm. We design two types of DSP, integral and coordinate, to implement the integral and coordinate algorithm, respectively. The corresponding block diagrams of the integral DSP and coordinate DSP are shown in Figure 2(a) and Figure 2(b), respectively.

The integral DSP has two bus interfaces, *Master* and *Slave*. The *Master* interface implements the data load/store, and the *Slave* interface implements the control and status logic accessing from other devices. The main components of

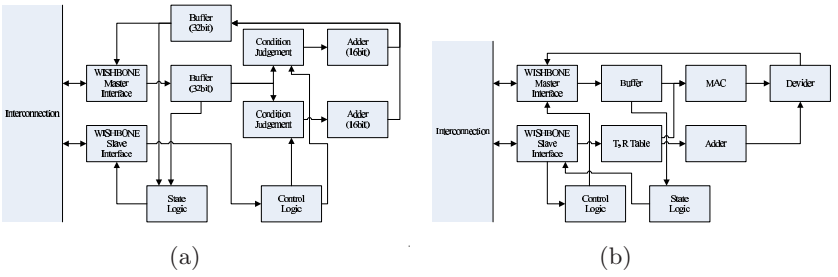


Fig. 2. The block diagrams of two DSP Cores a) The Integral arch structure (b) The Coordinate arch structure

the coordinate DSP are *MAC* (Multiply Accumulate) and *Divider*. The coordinate DSP has two bus interfaces, *Master* and *Slave*, which are as same as those of the integral DSP.

3.2 Interconnection Synthesis

In this section, we first compare and analyze several bus structures, and then propose an algorithm to get a better interconnection synthesis for MPSoC architecture.

In MPSoC architecture, since the customized DSP and other components have enough buffer or cache to debase the infection of the bus latency, we can ignore the effect of the buffer to the system. Furthermore, since the communication throughput of the *Slave* interface is very low for the integral DSP, coordinate DSP and DMA controller, in this section, we only focus on the *Master* interface when considering the design of interconnection synthesis.

In most on-chip bus standards, such as AMBA [2], CoreConnect [1], STBus [5] and WISHBONE [4], the share structure is used in the embedded processor as shown in Figure 3(b). In this structure, the total bandwidth of the interconnection is limited to the bandwidth of each node since all buses are connected to one node and only one master can access the interconnection simultaneously.

In order to fulfill the bandwidth requirement, we employ the crossbar structure in which different masters can access the slaves at the same time as shown in Figure 3(c). This structure improves the capability of the interconnection, and it is suitable for our architecture to process the integral algorithm in parallel. However, the area cost is very high in this structure. We have implemented these two structures in Wishbone protocol based on the open source IP core [3], and the results show that the crossbar structure uses more than 8 times area compared with that using the share structure. To reduce the area, the reduced crossbar structure which reduces the unnecessary connections between masters and slaves (in Figure 3(d)) is employed in our implementation. The results show that the reduced structure uses more than about four times more area compared with that using the share structure as shown in Table 5.

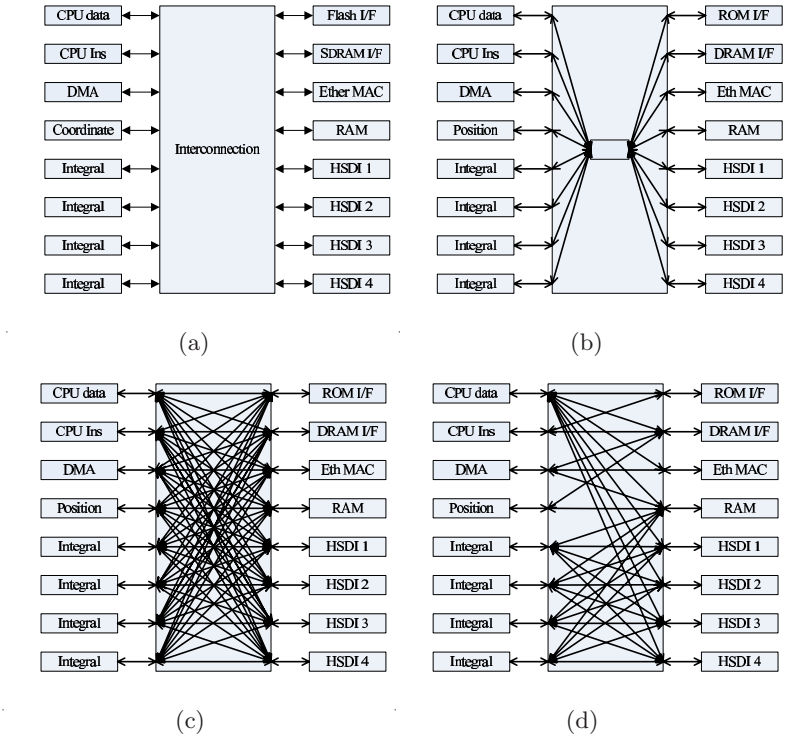


Fig. 3. Interconnection Structure (a) The main components (b) The share structure (c) The crossbar structure (d) The reduced crossbar structure

In order to further reduce the area cost, we design a novel algorithm which can reduce a generic-reduced matrix, thus to achieve our goals to reduce the area cost. The basic idea is to combine the *Slaves* and let the combined *Slaves* use only one *Slave* interface with the conditions that the total bandwidth of the combined *Slaves* can not exceed the bandwidth of every single bus, and the maximum number of buses can be reduced after the combination. We do not combine the *Masters* since such combination may cause more complicated problems [9]. We ignore the communication conflicts as the bandwidth of every single bus is low and the DSP and HSDI have enough buffers. Our MBRA algorithm (The Maximum-Bus-Reduction Algorithm) is shown in Figure 3.1.

In the inputs of the algorithm, M and S are the numbers of the master and slaves in the network, MAX is the maximum bandwidth of a single bus, and MS is the communication matrix where $MS[i][j]$ denotes the communication between master i and slave j. We place the communication bandwidth between the *Masters* and *Slaves* into $MS[i][j]$, in which if $MS[i][j] = 0$, it denotes that there is no bus between master i and slave j. The output of the algorithm is the optimized bus architecture by combining slaves as much as possible.

Algorithm 3.1 The Maximum-Bus-Reduction Algorithm (MBRA)

Require: $M \leftarrow$ The number of masters; $S \leftarrow$ The number of slaves; $MAX \leftarrow$ The upper-bound of the bandwidth of a single bus; $MS[M][S]$: The communication matrix where $MS[i][j]$ denotes the communication between master i and slave j ($MS[i][j] == 0$ denotes no bus);

Ensure: The communication matrix with the minimum number of slaves after the slave combination.

1: **//Assign a flag/number to each slave to denote if it has been combined/to which .**
2: For $i = 0$ to S-1, Flag_Slave[i]=NO; To_Slave[i]=-1;
3: **//Find the pair of slaves with the maximum cost (the reduced bus number) and combine the pair.**
4: **while** {1} **do**
5: **//Find the pair of slaves with the maximum cost.**
6: max_cost = -1;
7: **for** i=0 → S-2 **do**
8: **for** j=i+1 → S-1 **do**
9: combined_bus=total_bus=0; combined_flag=YES
10: **if** Flag_Slave[i] == NO and Flag_Slave[j] == NO **then**
11: **for** k=0 → M-1 **do**
12: If $MS[k][i] > 0$, total_bus ++; If $MS[k][j] > 0$, total_bus ++;
13: If $MS[k][i] > 0$ or $MS[k][j] > 0$, combined_bus ++;
14: If $MS[k][i] + MS[k][j] > MAX$, combined_flag=NO;
15: **end for**
16: **if** combined_flag==YES **then**
17: cost ← total_bus - combined_bus;
18: **if** cost > max_cost **then**
19: max_cost ← cost; combined_slave[0]=i; combined_slave[1]=j;
20: **end if**
21: **end if**
22: **end if**
23: **end for**
24: **end for**
25: **//Combine the slaves with the maximum cost (always combine Slave j to salve i).**
26: **if** min_cost != -1 **then**
27: i = combined_slave[0]; j=combined_slave[1];
28: For k=0 to M-1, $MS[k][i] += MS[k][j]$;
29: Flag_Slave[j]=YES; To_Slave[j]=i;
30: **else**
31: Break;
32: **end if**
33: **end while**

In the algorithm, for each pair of slaves, we first calculate the total numbers of buses from all masters to this slave pair before and after combining slaves. We then check if the combination is possible by comparing the combined combination

with the upper bound of bandwidth of a single bus. Next, if the combination is possible, we calculate the cost that is defined as the reduced number of buses after the combination. The cost is compared with the current recorded maximum cost. If it is larger than the current maximum cost, we record the slave pairs $< i, j >$ into an array. After all possible slave pairs have been checked, we combine the slave pair with the maximum cost, which means that we can reduce the maximum number of buses by combining this slave pair. Then we record which slave has been combined and combined into which one. We set a flag for the slave that has been combined into others so it will not be considered in the further combination. The above procedure will be repeated until we could not find any possible combination. The MBRA algorithm is a polynomial-time algorithm. It takes at most $O(|S|^3|M|)$ to finish where S is the number of slaves and M is the number of masters.

4 Experimental Results and Discussions

To compare our MPSoC architecture with the general architecture, we have implemented our interconnection with WISHBONE protocol and our bus interconnection synthesis algorithm. We compare our technique with the crossbar and the reduce crossbar structure in terms of the area cost.

The communication array without optimization between the masters and slaves in the interconnection is shown in Table 1. In this array, for example, the number 60 in column 3 and row 4 denotes that the communication request between master 4 and slave 3 is 60MBps. After applying our MBRA algorithm in 3.1, the original 8 slaves are combined into 4 slave groups, and the area is reduced accordingly. The reduced array is shown in Table 2. In this reduced array, the columns have been reduced from 8 to 4, and the slaves 1, 2, 5, and slaves 3, 8, 4 have been partitioned into 2 groups, which means that slaves of the two groups can be put into one single bus. With the reduced array, the final structure is shown in Figure 4.

The three interconnection structures are coded in Verilog HDL, and are synthesized to gate-level circuits using Synopsys Design Compiler and a UMC 0.18um standard cell library. The area cost comparison of the cross, the reduced cross and the final structure is show in Figure 5. The results show that the algorithm 3.1 reduces 12% of the area.

Table 1. The unreduced array

	s1	s2	s3	s4	s5	s6	s7	s8
m1	50	25	100	1	2	2	2	2
m2	10	50	0	0	0	0	0	0
m3	0	50	55	50	0	0	0	0
m4	0	4	60	0	0	0	0	0
m5	0	0	19	0	150	150	150	105
m6	0	0	19	0	150	150	150	105
m7	0	0	19	0	150	150	150	105
m8	0	0	19	0	150	150	150	105

Table 2. The reduced array

	s1,2,5	s3,8,4	s6	s7
m1	79	105	4	4
m2	60	0	0	0
m3	50	105	0	0
m4	4	6	0	0
m5	150	124	150	150
m6	150	124	150	150
m7	150	124	150	150
m8	150	124	150	150

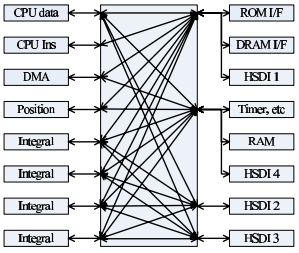


Fig. 4. The final crossbar structure

Structure	Area(K um^2)
Crossbar	1608
Shared crossbar	786
Our MBRA Algorithm	694

Fig. 5. Area Comparison

5 Conclusion

In this paper, we have proposed an interconnection synthesis algorithm for an MPSoC architecture for implementing real-time signal processing in gamma camera in [14]. We synthesized DSP cores and Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and satisfy the requirements of real-time signal processing for 256 × 256 image construction.

Acknowledgments

The work described in this paper was partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU A-PH13, PolyU A-PA5X, PolyU A-PH41, and PolyU B-Q06B), the National Nature Science Foundation of China (60525314), the 973 Program of China (2002CB312204-04) and the 863 Program of China (2006AA04Z206).

References

1. Ibm on-chip coreconnect bus architecture. www.chips.ibm.com
2. Arm amba specification (rev2.0) (2001), http://www.arm.com
3. Wishbone interconnect matrix ip core. rev. 1.1 (2002), http://www.opencores.org
4. Wishbone system-on-chip (soc) interconnection architecture for portable ip cores revision: B.3 (2002), http://www.opencores.org
5. Stbus communication system: Concepts and definitions, reference guide. STMicroelectronics (2003)
6. Beric, A., Sethuraman, R., Pinto, C.A., Peters, H., Veldman, G., van de Haar, P., Duranton, M.: Heterogeneous multiprocessor for high definition video. In: Consumer Electronics, 2006. ICCE 2006. 2006 Digest of Technical Papers. International Conference on, pp. 401–402 (2006)
7. Jeong, M.H., Choi, Y., Chung, Y.H., Song, T.Y., Jung, J.H., Hong, K.J., Min, B.J., Choe, Y.S., Lee, K.-H., Kim, B.-T.: Performance improvement of small gamma camera using nai(tl) plate and position sensitive photo-multiplier tubes. Physics in Medicine and Biology 49(21), 4961–4970 (2004)

8. Khatib, I.A., Poletti, F., Bertozzi, D., Benini, L., Bechara, M., Khalifeh, H., Jantsch, A., Nabiev, R.: A multiprocessor system-on-chip for real-time biomedical monitoring and analysis: architectural design space exploration. In: Sentovich, E. (ed.) DAC, pp. 125–130. ACM, New York (2006)
9. Pasricha, S., Dutt, N.D., Ben-Romdhane, M.: Constraint-driven bus matrix synthesis for mpsoc. In: Hirose, F. (ed.) ASP-DAC, pp. 30–35. IEEE, Los Alamitos (2006)
10. Paulin, P.G., Pilkington, C., Bensoudane, E., Langevin, M., Lyonnard, D.: Application of a multi-processor soc platform to high-speed packet forwarding. In: DATE 2004: Proceedings of the conference on Design, automation and test in Europe, p. 30058. IEEE Computer Society, Washington, DC, USA (2004)
11. Reyes, V., Kruijtzer, W., Bautista, T., Alkadi, G., Nnuez, A.: A unified system-level modeling and simulation environment for mpsoc design: Mpeg-4 decoder case study. In: DATE 2006: Proceedings of the conference on Design, automation and test in Europe, Leuven, Belgium, Belgium, 2006. European Design and Automation Association, pp. 474–479 (2006)
12. Sanchez, F., Benlloch, J.M., Escat, B., Pavon, N., Porras, E., Kadi-Hanifi, D., Ruiz, J.A., Mora, F.J., Sebastia, A.: Design and tests of a portable mini gamma camera. In: Medical Physics, pp. 1384–1397 (2004)
13. Sanchez, F., Fernandez, M.M., Gimenez, M., Benlloch, J.M., Rodriguez-Alvarez, M.J., De Quiros, F.G., Lerche, C.W., Pavon, N., Palazon, J.A., Martinez, J., Sebastia, A.: Performance tests of two portable mini gamma cameras for medical applications. Medical Physics 4210 (2006)
14. Sun, K., Wang, M., Shao, Z.: Mpsoc architectural design and synthesis for real-time biomedical signal processing in gamma cameras. In: International Conference on Biomedical Electronics and Devices (2008)

Integrated Global and Local Quality-of-Service Adaptation in Distributed, Heterogeneous Systems

Larisa Rizvanovic[1], Damir Isovic[1], and Gerhard Fohler[2]

[1] Department of Computer Science and electronics, Mälardalen University, Sweden
{larisa.rizvanovic,damir.isovic}@mdh.se
http://www.mrtc.mdh.se
[2] Department of Electrical and Computer Engineering,
University of Kaiserslautern, Germany
fohler@eit.uni-kl.de
http://www.eit.uni-kl.de/

Abstract. In this paper we have developed a method for an efficient Quality-of-Service provision and adaptation in dynamic, heterogeneous systems, based on our Matrix framework for resource management. It integrates local QoS mechanisms of the involved devices that deal mostly with short-term resource fluctuations, with a global adaptation mechanism that handles structural and long-term load variations on the system level. We have implemented the proposed approach and demonstrated its effectiveness in the context of video streaming.

Keywords: Quality-of-Service adaptation, distributed resource management, heterogenous systems, networked architectures, resource limitations and fluctuations.

1 Introduction

In distributed heterogeneous environments, such as in-home entertainment networks and mobile computing systems, independently developed applications share common resources, e.g., CPU, network bandwidth or memory. The resource demands coming from different applications are usually highly fluctuating over time. For example, video processing results in both temporal fluctuations, caused by different coding techniques for video frames, and structural fluctuations, due to scene changes [1]. Similarly, wireless networks applications are exposed to long-term bandwidth variations caused by other application in the system that are using the same wireless network simultaneously, and short-term oscillations due to radio frequency interference, like microwave ovens or cordless phones. Still, applications in such open, dynamic and heterogeneous environments are expected to maintain required performance levels.

Quality-of-Service (QoS) adaptation is one of the crucial operation to maximize overall system quality as perceived by the user while still satisfying individual application demands. It involves monitoring and adjustment of resources and data flows in order to ensure delivering of certain performance quality level to the application. This can be done *locally* on a device, i.e., local resource adaptation mechanisms on devices detect changes in resource availability and react to them by adjusting local resource consumption on host devices, or *globally*, on the system level, i.e., the QoS adaptation is performed by a global QoS manager with a full knowledge of the system resources. The

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 219–233, 2007.

first approach has the advantage that the application can use domain specific knowledge to adapt its execution to the available resources. For example, in a video streaming application, this could be achieved by decreasing the stream bit rate or skipping video frames. On the other hand, a global resource management is aware of the demand of other applications and it has an overview of the total resource availability on the system level. In this way, it may reassign budgets, or negotiate new contracts to maximize system overall performance.

While most of the existing approaches provide mechanisms for either local or global adaptation, we believe that both methods should be used together in order to respond properly to both local and global fluctuations. Hence, we propose an *integrated* global and local QoS adaptation mechanism, where the structural and long-term load variations on the system level are object for global adaptation, while the temporal load and short-term resource variations are taken care locally on devices. The task of the local adaptation mechanism is to adjust resource usage locally on a device as long as the fluctuation is kept within a certain QoS range. If the resource usage exceeds the range's threshold, the global adaptation mechanism takes over and performs resource reallocation on the system level.

QoS-aware applications are usually structured in such a way that they can provide different discrete quality levels, which have associated estimations of the required resources. We use the notion of abstract quality levels for defining QoS ranges, such as *high*, *medium* and *low* resource availability. This provides a general QoS framework that is not application or device specific. As long as an application or a device can express its resource demand and consumption in terms of an abstract level, it can benefit from our method.

Implementation of the proposed integrated QoS mechanism is enabled by our previous work, the Matrix framework for efficient resource management in distributed, heterogeneous environments [2]. It provides a global abstraction of device states as representation of the system state for resource management and it decouples device scheduling and system resource allocation. In this work, we use the Matrix as the infrastructure to develop global and local adaptation mechanisms and to integrate them into a single QoS adaptation unit in a system. While some parts of the resource management mechanism are adopted from the original Matrix approach and further developed here in terms of the newly proposed global adaptation mechanism, the local adaptation and the integrated mechanism are entirely new contributions. Furthermore, the original parts of this paper include a new module in the Matrix, the application adapter used for application adaptation, the quality level mapping and interfacing, the closed-loop control model for resource monitoring and adaptation, as well as the deployment of our approach in the context of video streaming and video stream adaptation.

The rest of this paper is organized as follows. In the next section we give an overview of the related work. In Section 3 we describe the extended Matrix framework used in our approach. In Section 4 we describe the global and the local adaptation mechanism and show how to integrate them in a single approach. In the same section, we present an example of how our method can be used in the context of media streaming. In Section 5 we describe the current implementation status, followed by Section 6, which concludes the paper.

2 Related Work

Comprehensive work on distributed QoS management architectures has been presented in [3,4,5,6,7]. However, those architectures are mostly designed to work over networks like ATM or the Internet with Integrated Services (IntServ) and Differentiated Services (DiffServ) support, i.e., networks that can provide guarantees on bandwidth and delay for data transfer. In our work, we do not make any assumptions that the underlying system (OS or network) can offer any QoS guarantees. We consider a distributed, heterogeneous environment where applications share resources, such as CPU and network bandwidth. Applications can either execute on a single device, or on several devices (e.g., a video streaming application that involves reading a stream on a server and sending it through a network to a hand held device to be decoded and displayed). Furthermore, we assume that handovers and other network related issues are done by lower level in the system architecture, and those are not task of our research.

While architectures like [8] give an overall management system for end-to-end QoS, covering all aspects from a user QoS policies to network handovers, in our work we focus on QoS management and resource adaptation in application domain. Our work is related to [9,10], which present application-aware QoS adaptation. Both of them make a separation between the adaptations on the system and application levels. While in [9] the application adjustment is actively controlled by a middleware control framework, in [10] this process is left to the application itself, based on upcalls from the underlying system.

Our work differs in how an application adaptation is initiated and performed. We do adaptation on different architectural levels, but unlike the mentioned work, we address global and local adaptation and the integration of both approaches.

We perform global adaptation of all resources within the system, while the work above focus on adjustment of resources on the end system, where the adaptation is based on the limited view of the sate of one device. We also provide an application independent approach, i.e., it can be used with different types of applications. Furthermore our approach can support component-based, decoupled approaches, where different components, like CPU or network schedulers can easily be replaced.

More recently, control theories have been examined for QoS adaptation. The work presented in [11] shows how an application can be controlled by a task control model. Method presented in [12] uses control theory to continuously adapt system behaviour to varying resources. However, a continuously adaptation maximizes the global quality of the system but it also causes large complexity of the optimization problem. Instead, we propose adaptive QoS provision based on a finite number of quality levels.

3 Resource Management Framework

Here we present the resource management framework used in our QoS adaptation method. First we give an overview of our previous work on distributed resource management, and then we extend it to suit the needs of the integrated QoS adaptation approach that will be presented in the next section.

3.1 Matrix Framework

The Matrix is an adaptive framework for efficient management of resources in distributed, heterogeneous environments. Figure 1 shows the data flow (information flow) between the Matrix components. The *Resource Manager* (RM) is used to globally schedule and reserve resources in the system, i.e., it makes decisions for resource usage for all devices in the system. Likewise, each time a new application is about to enter the system, the RM performs admission control.

For example, in a video streaming application, if the display device, e.g., a PDA, cannot manage to decode and display all video frames on time, the Resource Manager will notice this and instruct the sender device to send a less demanding version of the stream (e.g., with lower resolution).

In order to deal with resource reservation, the Resource Manager has to have knowledge about currently available resources in the system. This is provided in the *Status Matrix* (SM). For the example above, the Status Matrix will contain the information that CPU availability on the PDA is low while the bandwidth for the wireless link between the streaming server and the PDA device is high. The SM also provides information about active applications resource requirements, priorities, sink and source destinations.

Based on the information stored in the Status Matrix, the Resource Manager will make decisions for resource reallocation in the system, and store the orders for devices the *Order Matrix* (OM). An example of such an order could be one given to the streaming server to decrease the quality of streamed video.

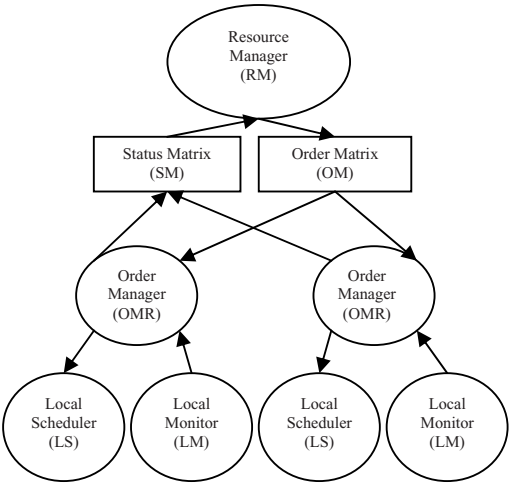


Fig. 1. The Matrix: Information flow

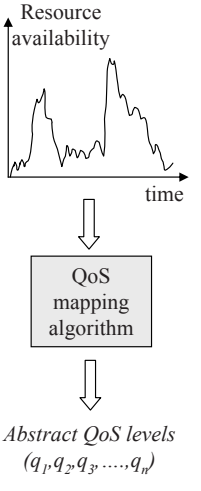


Fig. 2. Abstract QoS levels

The resource status information in the Status Matrix is provided by the *Order Managers* (OMR), located on the devices. For each type of shared resources, there is an Order Manager responsible for publishing the current resource availability on the device in the Status Matrix. This information is provided to the Order Manager through *Local*

Monitors (LM), that are responsible for continuous monitoring of a resource availability on a device, e.g., the available CPU or the network bandwidth. The accuracy of the information depends on a chosen temporal granularity.

Furthermore, an Order Manager receives orders from the Order Matrix and makes sure to adjusts local resource usage according to them. This is done through *Local Schedulers* (LS), which are responsible for scheduling of a local resources, e.g., a network packets scheduler that can adjust the packet sending rate according to available bandwidth.

For further details on Matrix framework we refer to our previous work [2,13].

3.2 QoS Levels

We want to use the minimum relevant information about devices states as needed for resource management, in order to reduce the system state presentation, and to abstract over fluctuations, which could overload scheduling of resources. Thus, we use the notion of a few *abstract QoS levels* that represent a resource's availability and an application's quality. For example, the variations in the quality of network link connection between two devices can be represented by e.g., three abstract QoS level values, (L)ow, (M)edium and (H)igh. H means that the data can be transmitted through the link with full available capacity, while L indicates severe bandwidth limitations. Likewise, quality of each application using certain resources is mapped to a finite number of application QoS levels.

In general, the availability of each resource is represented in our approach as a vector of discrete range of *n* QoS performance levels $\{q_1, q_2, ...q_k, q_{k+1}, ..., q_n\}$, see Figure 2. The value range of a QoS level q_k is defined by its threshold values $[q_k^{min}, q_k^{max}]$.

In this work, we apply linear mapping between the resources and the QoS levels, e.g., based on experimental measurements [14]. For example, one simple mapping for the CPU bandwidth based on the CPU utilization U could be e.g., $0 \leq U \leq 0.3 \Rightarrow H$, $0.3 < U \leq 0.6 \Rightarrow M$, $0.6 < U \leq 1.0 \Rightarrow L$. A more advanced mapping could, for instance, use fuzzy logic to provide a larger number of QoS levels with finer granularity, but QoS mapping is an ongoing work and it is out of the scope of this paper.

3.3 Application Adapter

The Matrix is an application independent framework, and application adaptation is not the main focus of our work. However, in order to advance the usage of the Matrix along with various types of applications, we have extended the original Matrix architecture with an additional component, the *Application Adapter* (AA). The Application Adapter performs the mapping of QoS levels to the application specific parameters, and vice versa. For example, the AA for a video streaming application could map abstract quality levels, such as H, M and L, into real possible frame-per-second (fps) values for the stream, e.g., for a 30 fps MPEG-2 stream high quality could mean the fps-interval between 24 and 30 fps, medium quality is 16 to 23 fps and low quality could be defined as 10 to 15 fps.

Since this process is application specific, our ambition was to provide an interface for this component, and than is up to the application designer to implement it. If there is

a way in an application to map its resource fluctuations into some abstract levels, then it can be used with our design. Also, upon resource reallocation, the Application Adapter will receive orders about new abstract levels from the Order Manager, which must be translated into some concrete actions on the application level.

4 Integrated QoS Adaptation Approach

In this section we present our integrated global and local adaptation mechanism that uses the Matrix framework. In our approach, global adaptation is performed by the Resource Manager, while the local adaptation is taken care of locally on the devices.

Consider the following motivating example: A person uses a PDA to watch a video stored on a local video server, which is delivered to the PDA through a wireless network. As the person moves around with the PDA, at some point it becomes almost out of range for the server, which results in video interruption due to packet losses. A local adaptation on the PDA does not really help in this case, since the video disruption is caused by the buffer underflow in PDAs decoder (in the case of buffer overflow, this could be treated locally on the PDA by e.g., speeding up the video decoding task). However, if there is a mechanism at the system level that can detect the lower bandwidth of the wireless link, i.e., the Matrix framework described in previous section, it could instruct the video server to stream a lower quality video stream that takes less network bandwidth.

Expressed in more general terms, resource consumption is adjusted locally on devices as long as the fluctuation stays within the range of requested QoS. For example, the Local Monitor detects a change in available CPU for a certain application, but this change is not large enough to enforce a different quality level to the application. Instead, the Local Scheduler could perform some local countermeasures, e.g., prioritize the application on the cost of some other application running on the same device. However, if the resource availability passes the defined thresholds (abstract QoS levels), the entire system gets involved via the global adaptation mechanism. The whole idea is illustrated in Figure 3.

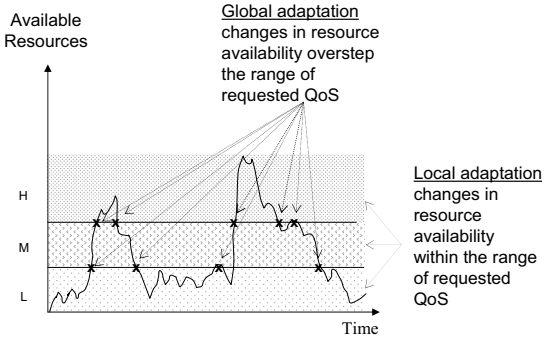


Fig. 3. Different types of resource variations handled on different architectural levels

4.1 Local Adaptation Mechanism

Local adaptation involves detecting the changes in resource availability and reacting to those via the local scheduler. The ideas from control theory can be used to achieve this. We use the *closed loop* model, i.e., a control model that involves feedback to ensure that a set of conditions is met. It involves the Local Monitor, the Local Scheduler, and the Order Manager, see Figure 4. Expressed by terminology of the control theory, we use the following terms for inputs and outputs variables in our control model; *control variable*, v_{ctrl}, is the value observed by the local monitor (e.g. network packet loss, CPU utilization), *reference variable*, v_{ref}, is concrete performance specification for Local Schedulers made by the order manager, *error* ε is the difference between the value observed by the Local Monitor and the reference variable, and *control input variable*, v_{in}, is the value calculated by the adaptation algorithm in order to adapt scheduling of the local resources. The Local Monitor continuously monitors available resources in the

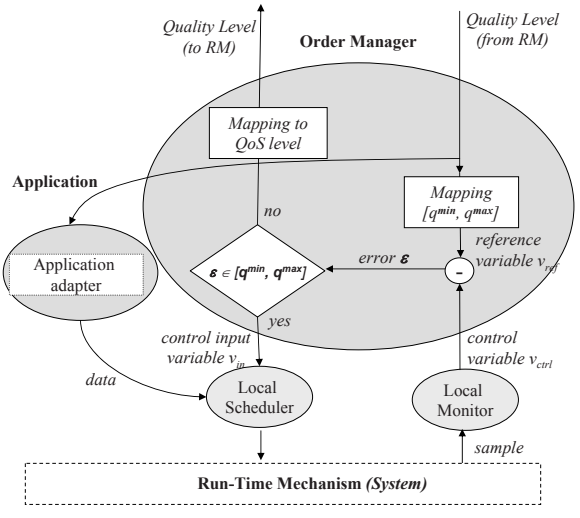


Fig. 4. Local QoS Adaptation Mechanism

system (e.g., CPU or bandwidth). Thus, in our control model it acts as an *observer* of the controlled system. It send the observed control value to the Order Manager. The Order Manager calculates the difference between the desired value, defined by the currently used QoS level, and the observed control value, i.e., it calculates the error value of the control loop. As long as resource availability stays within the boundaries for the given QoS level, i.e., the error falls in the range of the current QoS level, the output of the adaptation algorithm, control input, is passed to the Local Scheduler, i.e., the adapter part of control loop.

In the case that the error value implies a change in QoS levels, the values in the Status Matrix are updated and the Resource Manager is informed about the change. From this point, the global adaptation mechanism takes over, which we describe next.

4.2 Global Adaptation Mechanism

Whenever a local mechanism detects that a local resource availability has exceeded the current QoS level, a global adaptation mechanism will be initiated. The objective of the global adaptation is to adjust the resource usage among all involved applications. If the resource availability has increased, it will be given to involved applications (in terms of increased quality levels). Similarly, if the resource availability has decreased, the quality levels of the consumer applications will be decreased.

We support user defined *priorities* to be used when redistributing resources, i.e., the higher the priority of an application, the faster the quality increase of the application. However, it is up to the user to use priorities or not. Based on this, we distinguish between three reallocation policies in our approach, *fair*, *fair prioritized* and *greedy*.

Fair reallocation – If the priorities are not used, then the resources are adjusted (increased or decreased) in a strictly fair fashion: for each consumer applications the quality is adjusted step-by-step, one QoS level at the time, and then, if there are still resources to increase/decrease, we repeat the procedure for all applications once again, until the resource is consumed/replanished. For example, consider four different applications a_1,a_2, a_3 and a_4 that are using the same resource r. The current quality level for each applications is set to L. Assume that a_4 gets terminated and the resource availability of r gets increased by the portion used by a_4. The freed resource is given back to the remaining three application such that we first increase the the QoS level of a_1,a_2 and a_3 to M, and then, if there are still resources left, all QoS levels are increased to H.

Fair-prioritized reallocation – Note that in the fair approach, there is no guarantee that a certain application will change its QoS level. In the example above, there could be a case where the freed resource is entirely consumed after increasing the level of a_1 and a_2 to level M, so that a_3 will remain running on level L, despite the fact that a_3 might be the most important one in the system. However, if we use priorities, we could instruct RM to start by increasing the QoS levels of high priority applications first, i.e., a_3 in the example above. In other words, the resources are reallocated in a fair fashion, i.e., each application's quality level is changed by one step before changing any other application's level one more step, but also we use priorities to determine which applications should be served first.

Greedy reallocation – Moreover, priorities enable for an another reallocation policy, i.e., greedy redistribution. This means to increase (decrease) QoS level of an application with the highest (lowest) priority until it reaches its maximum (minimum) QoS level, before we start with the next one application (in the priority order). For the example above, we would continue increasing the QoS level of a_3 until it reaches H, before doing any QoS increase of a_1 and a_2. Furthermore, the priorities can be used when selecting which applications to drop first if that becomes necessary.

If an application is processed by several different devices, then, before changing its quality level, we need to check if the new level can be supported by all involved devices on the application's playout route. For example, in a video streaming application where a video stream is sent from a video server to a hand held device via a laptop, the

bandwidth increase between the server and the laptop does not necessarily mean that we should start streaming a higher bit rate stream, since the link between the laptop and the hand held device might not be able to support it. Likewise, we have to consider if this increased quality can be supported by all other types of resources that the application is consuming e.g., there is no point to send more data over the communication link than it cannot be timely processed at the receiver device (by the local CPU).

Our *admission control* approach for new applications is quite similar to the adaptation approach described above. Thus, each time a new application is about to enter the system, the Resource Manager has to determine if sufficient resources are available to satisfy the desired QoS of the new connection, without violating QoS of existing applications. If yes, then we accept the new application and publish orders for resource reservation/reallocation into the Order Matrix. If no, we check if there are any existing application with the the lower priority than the new one, and if so, decrease their QoS (starting with the lowest priority application) to free some resources for the new application. If there are no available resources, and no lower priority applications, the new applications is rejected.

4.3 Pseudo-Code for Integrated Approach

Here is the pseudo-code for our current implementation of the integrated local and global QoS adaptation mechanism. We introduce some additional terms, as a complement to the terms presented earlier:

- $\mathcal{A} = \{a_1, a_2, .., a_n\}$, a set of applications in the system.
- $\mathcal{R} = \{r_1, r_2, ..., r_m\}$, a set of resources in the system.
- $\mathcal{D} = \{d_1, d_2, ..., d_p\}$, a set of devices in the system.
- $\mathcal{A}(r_i) \in \mathcal{A}$, a subset of applications that currently use resource r_i.
- $\mathcal{R}(a_j) \in \mathcal{R}$, a subset of resources currently used by application a_j.
- $\mathcal{R}(d_l) \in \mathcal{R}$, a subset of resources currently consumed on device d_l.
- $\mathcal{D}(a_j) \in \mathcal{D}$, a subset of devices currently used for processing of application a_j.
- $S(r_i)$, current resource supply (availability) of resource r_i.
- $D(r_i)$, current resource demand of all applications using r_i.
- $q_k(r_i)$ and $q_k(a_j)$, the *k*-th QoS level of resource r_i, respective application a_j, as described in section 3.2.

/ For the sake of simpler explanation, we omit in the pseudo-code for the start up activities where the devices has reported the local resource availability, and the RM has published initial QoS levels in the Status Matrix */*

$\forall\ d_i \in \mathcal{D}$ */* For each device */*
  $\forall\ r_i \in \mathcal{R}(d_i)$ */* For each resource on a device */*

/ Invoke local adaptation based on the currently assigned quality level */*
map $q_k(r_i) \Rightarrow [q_k^{min}(r_i), q_k^{max}(r_i)]$
$v_{ref} = q_k^{max}(r_i)$, $\varepsilon^{max} = q_k^{max}(r_i) - q_k^{min}(r_i)$

Do
get v_{ctrl} from LM
$\varepsilon = v_{ref}$ - v_{ctrl}
calculate $v_{in}(\varepsilon)$ and send it to LS
While *(*$0 \leq \varepsilon \leq \varepsilon^{max}$*)*

/ Prepare for global adapt. when the error exceeds the limit of current QoS level */*
map $\varepsilon \Rightarrow q_l(r_i), l \neq k$
publish $q_l(r_i)$ in SM
$\Rightarrow$ *break! invoke global adaptation*

/ RM performs global adaptation based on new info in SM */*

/ Case 1: total resource supply is greater than the total demand* $\Rightarrow$ *increase QoS levels */*
If $(S(r_i) > D(r_i))$ ***Then***
Do
/ Based on the chosen realloc. policy get an application to increase its QoS level */*
If (a_j = getApplication(POLICY, INCREASE)) ***Then***

/ Check if all* a_j*'s proc. devices (other than* d_i*), support the next QoS level of* a_j **/*
If$(\forall d_j \in \mathcal{D}(a_j), d_j \neq d_i, d_j$ supports $q_{k+1}(a_j))$ ***Then***

/ Check if the new QoS level of* a_j *can be served by all other* a_j*'s resources*/*
If$(\forall r_n \in \mathcal{R}(a_j), r_n \neq r_i, r_n$ supports $q_{k+1}(a_j))$ ***Then***
increase quality of a_j to $q_{k+1}(a_j)$
/ incr/decr dem/sup for* r_i *by the amount used to jump to next QoS lev.*/*
$\Delta = q_{k+1}^{max}(r_i) - q_k^{max}(r_i)$
$D(r_i) += \Delta$; $S(r_i) -= \Delta$

While *(*$S(r_i) > D(r_i)$ *AND* $a_j \neq$ *NULL)*

/ Case 2: total resource supply is less than the total demand* $\Rightarrow$ *decrease QoS levels */*
Else
/ Similar as above, but the QoS levels are decreased. Also, we do not need to check other devices and resources, since the decr. quality will not put extra demands on them.*
...(omitted)

4.4 Example

Here we illustrate our approach in the context of video streaming. Consider the example scenario with the PDA and the streaming video server from Section 3, where the quality of the streamed video was dependent on the distance between the PDA and the server. At some point in time, the PDA is so far away from the server so it only makes sense to stream a low quality video stream, i.e., stream S_1 with the abstract quality level L and priority p_1. Assume also that there is another video stream in the system, S_2, streamed from the server to a laptop with a quality level H and higher priority p_2. The CPU

availability (bandwidth) on all devices is initially assumed to be high. The reallocation policy used is fair-prioritized. The whole situation is depicted in Figure 5. The values within the parentheses are the new QoS levels (obtained after adaptation).

Now, assume that the person with the PDA starts moving closer to the server. The local adaptation mechanism on the one of the involved devices, i.e., either on the server or on the PDA, will detect that more and more packets can be sent between them (let's assume the PDA will detect this first). As the PDA is coming closer to the server, at some point the quality of the link connection will exceed the assigned threshold for the local adaptation, and the global adaptation mechanism will take over, with the following steps involved (see Figure 5 in parallel; the numbers below correspond to the numbers in the figure; some of the steps are merged):

1. The Local Monitor on the PDA detects that the link quality between the server and the PDA has increased.
2. This is reported to the Order Manager, who will map the new values to the quality level H (we can assume a sudden large connection improvement e.g., by entering the room where the server is placed).
3. Order Manager publishes the new quality level H in the Status Matrix.
4. Assume also that there has been some change in the CPU availability on the laptop, i.e., it gets decreased from H to L due some new, CPU intensive application that has started to run on the laptop. Initially, the local adaptation mechanism on the laptop will react to the changes in the CPU load by e.g., by performing selective frame skipping in the video decoder that is processing the stream S_2. However, at some point the CPU QoS threshold will be exceeded and the new QoS value will be calculated and published in the Status Matrix for the CPU.
5. The Resource Manager is notified about the new quality level values.
6. Now, it is up to the Resource Manager to take a decision about the resource reallocation. Considering the available bandwidth and the streams priorities, one solution could be to set the quality of S_1 to M (since it has lower priority), and left the quality of S_2 unchanged. However, streaming the high quality video stream to the laptop may not be a good solution, since the CPU on the laptop is overloaded and video frames will be skipped anyway. Hence, the Resource Manager, who has the total resource usage view of the system, decides to set L for stream S_2. This decision will not only reflect the resource status on the laptop correctly, but also it will allow for S_1 to be set to H (which can be done because the quality of the connection between the server and the PDA has been changed to H).
7. The Order Managers on respective devices are informed about the new values (arrows to the OMRs of the PDA and the laptop are omitted in the figure to ease readability).
8. The Order Managers then enforce the new settings via their local schedulers and application adapters. For example, in the case of the server, the stream application adapter will make sure to decrease the quality of stream S_2. This can be done in several ways, e.g., by reading a lower quality version of S_2 that has been stored on the server in advance, or by using an online modification of original S_2 by using the quality-aware preventive frame skipping methods that we have developed in our previous work [15].

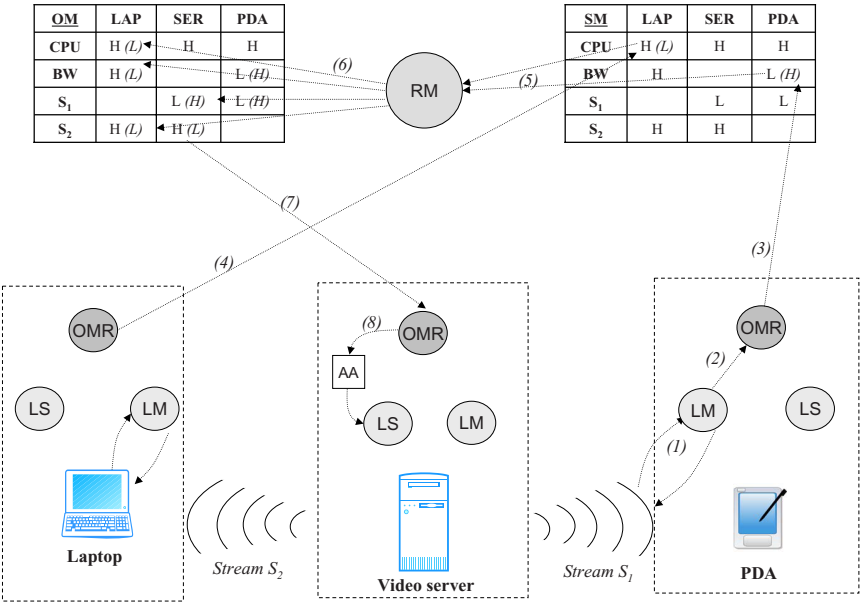


Fig. 5. Example global adaptation

5 Implementation and Evaluation

The Matrix framework is quite complex and we are still working on its full implementation. However, we have implemented a mock-up of Matrix approach [2] using HLA [16]. Moreover, some basic benefits of our method, has been demonstrated by simulations.

5.1 Implemented Modules

The hierarchical architecture and the loose coupling between system modules makes it possible to work on different parts independently. Current implementation includes Local Monitors and Schedulers for CPU and network bandwidth, and a Video Stream Adapter.

Local Network Scheduler – For network scheduling we use the traffic shaping approach, which provides different QoS by dynamically adapting the transmission rate of nodes, to match the currently available bandwidth of a wireless network. The Traffic Shaper adjusts the outbound traffic accordingly to input parameters (i.e., the amount of available bandwidth assign to the Local Scheduler). Please see [14] for full implementation details.

Local Network Monitor – For monitoring and estimation of available bandwidth (over 802.11b wireless Ethernet), we use a method that provides us with the average bandwidth that will be available during a certain time interval. The architecture consists

of a bandwidth predictor that first uses a simple probe-packet technique to predict the available bandwidth. Then, exponential averaging is used to predict the future available bandwidth based on the current measurement and the history of previous predictions, see [14] for details.

Local CPU Scheduler – The allocation of CPU to the applications depends on the scheduling mechanism that is used. We have developed a predictable and flexible real-time scheduling method that we refer to as *slot shifting* [17]. The basic idea is to guarantee a certain quality of service to applications before run-time, and then adjust it at run-time according to the current status of the system.

Local CPU Monitor – Since we use a real-time scheduling mechanism, the CPU monitoring is very simple to achieve. The *spare capacity* mechanism of slot shifting provides easy access of the amount and the distribution of available resources at run-time [17].

Video Stream Adapter – We have implemented an Application Adapter for MPEG-2 video stream adaptation, based on quality-aware, selective frame skipping. Order Manager sends allowed abstract quality level to the video adapter, which then adjusts the stream according to available resources by skipping the least important video frames. For the frame priority assignment algorithm we have proposed a number of criteria to be applied when setting priorities to the frames. Please see our previous work [15] for details.

5.2 Evaluation

We have evaluated our method in the context of video streaming. Here we present results from a 15 minutes video streaming simulation using our integrated approach for global and local adaptation. We simulate usage of 30 devices in the system and show how a MPEG-2 video stream is adapted based on current resource availability (network bandwidth). We use the following quality levels for available bandwidth (given in Mbps): $q_1(BW) = [q_1^{max}, q_1^{min}] = [1.5, 2.5]$ (L), $q_2(BW) = [q_2^{max}, q_2^{min}] = [2.5, 4]$ (M), $q_3(BW) = [q_3^{max}, q_3^{min}] = [4, 11]$ (H). Figure 6 shows that the local adaptation mechanism is deployed most of the time (77%), while the global mechanism is triggered only when necessary (23%), i.e., the QoS has changed that much that the system reallocation must take place.

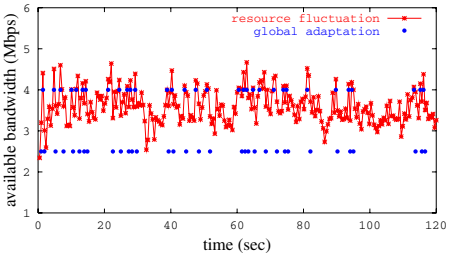


Fig. 6. Invocation of global adaptation

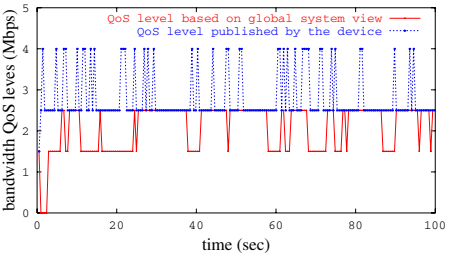


Fig. 7. Global vs Local system view

Figure 7 shows the difference between QoS levels based on one device's local view and those assigned by global adaptation, i.e. the possible spared resources (available bandwidth) on just one device due to global adaptation. It illustrates efficiency of our integrated approach where adjustment of resources is not just based on the limited local system view of one device, but also on the current available resources of all involved devices. In that way, our approach enables a system wide optimization.

6 Conclusions and Future Work

We proposed a method for efficient Quality-of-Service adaptation in dynamic, heterogenous environments.

It integrates global and local adaptation, where the first one takes care of the structural resource fluctuations on the system level, while the second one is performed locally on devices to handle short-term variations.

The idea is to perform local adaptation as long as possible, using a control model for resource monitoring and adjustment, and if a resource availability passes the range of the currently assigned QoS level, the global adaptation mechanism takes over.

Our current and future work include further developing the local control model by formally describing the system's behaviour with a set of differential equations. Furthermore, we are working on a more general model for mapping between resources demands and abstract QoS levels and exploiting the proposed framework in other application domains than in-home networks.

References

1. Otero Perez, C., Steffens, L., van der Stok, P., van Loo, S., Alonso, A., Ruíz, J.F., Bril, R.J., García Valls, M.: QoS-based resource management for ambient intelligence, Ambient intelligence: impact on embedded system design. Academic Publishers, Norwell, MA, USA (2003)
2. Rizvanovic, L., Fohler, G.: The MATRIX: A QoS Framework forStreaming in Heterogeneous Systems. In: International Workshop on Real-Time for Multimedia, Catania, Italy (2004)
3. Nahrstedt, K., Smith, J.M.: Design, Implementation an Experiences of the OMEGA End-Point Architecture, Distributed Systems Laboratory, University of Pennsylvania, Philadelphia
4. Nahrstedt, K., Chu, H., Narayan, S.: QoS-Aware Resource Management for Distributed Multimedia Applications, UIUCDCS-R-97-2030 (1997)
5. Campbell, A., Coulson, G., Hutchison, D.: A quality of service architecture, ACM SIGCOMM Computer Communication Review (1994)
6. Gopalakrishna, G., Parulkar, G.: Efficient Quality of Service in Multimedia Computer Operating Systems, Washington University (1994)
7. Shankar, M., De Miguel, M., Liu, J.W.S.: An end-to-end QoS management architecture, Real-Time Technology and Applications Symposium (1999)
8. Kassler, A., Schorr, A., Niedermeier, C., Schmid, R., Schrader, A.: MASA - A scalable QoS Framework. In: Proceedings of Internet and Multimedia Systems and Applications (IMSA), Honolulu, USA (2003)
9. Li, B., Nahrstedt, K.: A Control-Based Middleware Framework for Quality-of-Service Adaptations. Selected Areas in Communications, IEEE Journal (1999)

10. Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R.: Agile Application-Aware Adaptation for Mobility. In: 16th ACM Symposium on Operating Systems Principles, France (1997)
11. Li, B., Nahrstedt, K.: Impact of Control Theory on QoS Adaptation in Distributed Middleware Systems. In: American Control Conference (2001)
12. Stankovic, J.A., Abdelzaher, T., Marleya, M., Tao, G., Son, S.: Feedback control scheduling in distributed real-time systems. In: RTSS (2001)
13. Rizvanovic, L., Fohler, G.: The MATRIX - A Framework for Real-time Resource Management for Video Streaming in Networks of Heterogenous Devices. In: Conference on Consumer Electronics, Las Vegas, USA (2007)
14. Lennvall, T., Fohler, G.: Providing Adaptive QoS in Wireless Networks by Traffic Shaping, Resource management for media processing in networked embedded systems (RM4NES), Netherlands (2005)
15. Isovic, D., Fohler, G.: Quality aware MPEG-2 Stream Adaptation in Resource Constrained Systems, ECRTS, Catania, Italy (2004)
16. IEEE Standard for Modeling and Simulation, High Level Architecture (HLA) - Federate Interface Specification, No.:1516.1-2000
17. Isovic, D., Fohler, G.: Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints, 21st IEEE RTSS, USA (2000)

Toward to Utilize the Heterogeneous Multiple Processors of the Chip Multiprocessor Architecture

Slo-Li Chu

Department of Information and Computer Engineering,
Chung Yuan Christian University, Chung-Li, Taiwan, R.O.C.
slchu@cycu.edu.tw

Abstract. Continuous improvements in semiconductor fabrication density are supporting new classes of Chip Multiprocessor (CMP) architectures that combine extensive processing logic/processor with high-density memory in a single chip. One of the architecture, called Processor-in-Memory (PIM) can support high-performance computing by combining various processors in a single system. Therefore, a new strategy is developed to identify their capabilities and dispatch the most appropriate jobs to them in order to exploit them fully. This paper presents a novel scheduling mechanism, called Swing Scheduling to fully utilize all of the heterogeneous processors in the PIM architecture. Integrated with our Octans system, this mechanism can decompose the original program into blocks and can produce a feasible execution schedule for the host and memory processors, even for other CMP architectures. The experimental results for real benchmarks are also proposed.

Keywords: Chip Multiprocessor (CMP), Processor-in-Memory, Swing Scheduling, Octans.

1 Introduction

In current high-performance computer architectures, the processors run many times faster than the computer's main memory. This performance gap is often referred to as the Memory Wall [25]. This gap can be reduced using the System-on-a-Chip or Chip Multiprocessor [13] strategies, which integrates the processors and memory on a single chip. The rapid growth in silicon fabrication density has made this strategy possible. Accordingly, many researchers have addressed integrating computing logic/processing units and high density DRAM on a single die [5][7][8][9] [10][12][13]. Such architectures are also called Processor-in-Memory (PIM), or Intelligent RAM (IRAM).

Integrating DRAM and computing logic on a single integrated circuit (IC) die generates PIM architecture with several desirable characteristics. First, the physical size and weight of the overall design can be reduced. As more functions are integrated on each chip, fewer chips are required for a complete design. Second, very wide on-chip buses between the CPU and memory can be used, since DRAM is located with computing logic on a single die. Third, eliminating off-chip drivers reduces the power consumption and latency [12].

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 234–246, 2007.

This class of architectures constitutes a hierarchical hybrid multiprocessor environment by the host (main) processor and the memory processors. The host processor is more powerful but has a deep cache hierarchy and higher latency when accessing memory. In contrast, memory processors are normally less powerful but have a lower latency in memory access. The main problems addressed here concern the method for dispatching suitable tasks to these different processors according to their characteristics to reduce execution times, and the method for partitioning the original program to execute simultaneously on these heterogeneous processor combinations.

Previous studies of programming for PIM architectures [4][6] have concentrated on spawning as many processors as possible to increase speedup, rather than on the capability difference between the host and memory processors. However, such an approach does not exploit the real advantages of PIM architectures. This study integrates our Octans system that integrates statement splitting, weight evaluation and a scheduling mechanism. The original scheduling [2] mechanism is improved to generate a superior execution schedule to fully utilize all heterogeneous processors in the PIM architecture, using our new Swing Scheduling mechanism. A weight evaluation mechanism is established to obtain a more precise estimate of execution time, called weight. The Octans system can automatically analyze the source program, generate a Weighted Partition Dependence Graph (WPG), determine the weight of each block, and then dispatch the most suitable blocks for execution on the host and memory processors.

The rest of this paper is organized as follows: Section 2 introduces PIM architectures. Section 3 describes our Octans system and the Swing Scheduling algorithms. Section 4 presents experimental results. Conclusions are finally drawn in Section 5.

2 The Processor-in-Memory Architecture

Fig. 1 depicts the organization of the PIM architecture evaluated in this study. It contains an off-the-shelf processor, P.Host, and four PIM chips. The PIM chip integrates one memory processor, P.Mem, with 64 Mbytes of DRAM. The techniques presented in this paper is suitable for the configuration of one P.Host and multiple P.Mems, and can be extended to support multiple P.Hosts.

Table 1 lists the main architectural parameters of the PIM architecture. P.Host is a six-issue superscalar processor that allows out-of-order execution and runs at 800MHz, while P.Mem is a two-issue superscalar processor with in-order capability and runs at 400MHz. There is a two-level cache in P.Host and a one-level cache in P.Mem. P.Mem has lower memory access latency than P.Host since the former is integrated with DRAM. Thus, computation-bound codes are more suitable for running on the P.Host, while memory-bound codes are preferably running on the P.Mem to increase efficiency.

The PIM chip is designed to replace regular DRAMs in current computer systems, and must therefore conform to a memory standard that involves additional power and ground signals to support on-chip processing. One such standard is Rambus [5], so the

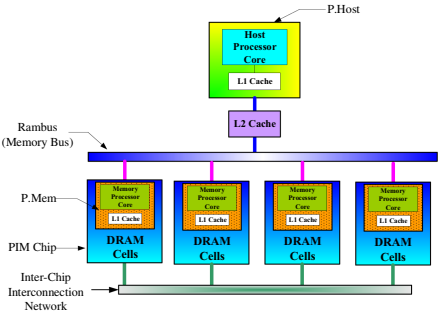


Fig. 1. Organization of the PIM architecture

Table 1. Parameters of the PIM architecture

P.Host	P.Mem	Bus & Memory
Working Freq: 800 MHz	Working Freq: 400 MHz	Bus Freq: 100 MHz
Dynamic issue Width: 6	Static issue Width: 2	P.Host Mem RT: 262. 5 ns
Integer unit num: 6	Integer unit num: 2	P.Mem Mem RT: 50. 5 ns
Floating unit num: 4	Floating unit num: 2	Bus Width: 16 B
FLC_Type: WT	FLC_Type: WT	Mem_Data_Transfer: 16
FLC_Size: 32 KB	FLC_Size: 16 KB	Mem_Row_Width: 4K
FLC_Line: 64 B	FLC_Line: 32 B	
SLC_Type: WB	SLC: N/A	
SLC_Size: 256 KB		
SLC_Line: 64 B		
Replace policy: LRU		
Branch penalty: 4	Branch penalty: 2	
P.Host_Mem_Delay: 88	P.Mem_Mem_Delay: 17	

* FLC stands for the first level cache, SLC for the second level cache, BR for branch, RT for round-trip latency from the processor to the memory, and RB for row buffer.

PIM chip is designed with a Rambus-compatible interface. The private interconnection network of the PIM chips is also provided.

3 The Octans System

Most current parallelizing compilers focus on the transformation of loops to execute all or some iterations concurrently, in a so-called iteration-based approach. This approach is suited to homogeneous and tightly coupled multi-processor systems. However, it has an obvious disadvantage for heterogeneous multi-processor platforms because iterations have similar behavior but the capabilities of heterogeneous processors are diverse. Therefore, a different approach is adopted here, using the statements in a loop as a basic analysis unit, called statement-based approach, to develop the Octans system.

Octans is an automatic parallelizing compiler, that partitions and schedules an original program to exploit the specialties of the host and the memory processor. At first, the source program is split into blocks of statements according to dependence relations. Then, the Weighted Partition Dependence Graph (WPG) is generated, and the weight of each block is evaluated. Finally, the blocks are dispatched to either the host or the memory processors, according to which processor is more suitable for executing the block. The major difference between Octans and other parallelizing systems is that it uses a statement rather than an iteration as the basic unit of analysis. This approach can fully exploit the characteristics of statements in a program and dispatch the most suitable tasks to the host and the memory processors. Fig. 2 illustrates the organization of the Octans system.

3.1 Statement Splitting and WPG Construction

Statement Splitting splits the dependence graph of the given program by the extended node partition mechanism as introduced in [2]. It divides the original program into

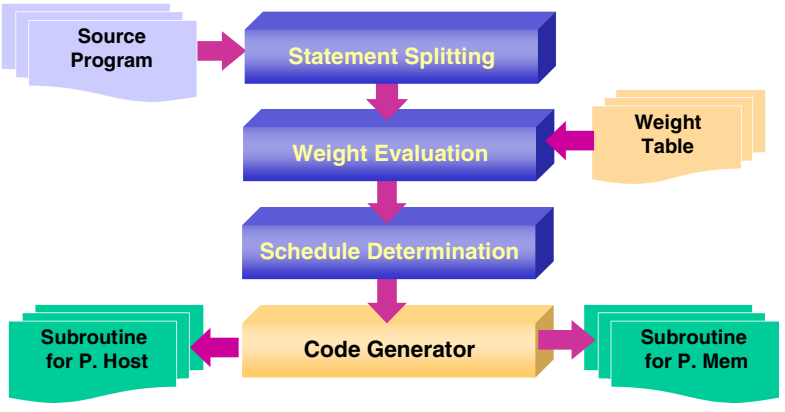


Fig. 2. The sequence of compiling stages in Octans

several small loops within the minimal statements. The detailed mechanisms can be found in literature [2]. Then WPG Construction constructs the Weighted Partition Dependence Graph (WPG), to be used in the subsequent stages of Weight Evaluation, Wavefront Generation and Schedule Determination.

3.2 Weight Evaluation

Two approaches to evaluating weight can be taken. One is to predict the execution time of programs by profiling the dominant parts. The other considers the operations in a statement and estimates the program execution time by looking up an operation weight table. The former method called code profiling may be more accurate, but the predicted result cannot be reused; the latter called code analysis can determine statements for suitable processors but the estimated program execution time is not sufficiently accurate. Hence, the Self-Patch Weight Evaluation scheme was designed to combine the benefits of both approaches. It integrates these two approaches together by analyzing code and searching weight table first to estimate the weight of a block. If the block contains unknown operations, the patch (profiling) mechanism is then activated to evaluate the weights of unknown operations. The obtained operation weights are added into the weight table for next look-up. For a detailed description of this scheme, please refer to [2].

3.3 The Swing Scheduling Mechanism

Here we propose the Swing Scheduling mechanism to achieve a good schedule for utilizing all of the memory processors in PIM architecture. At first, the redundancy and synchronization between processors are critical factors that affect the performance of job scheduling for multiprocessor platforms. A critical path mechanism is used to minimize the frequency of synchronization. Then the WPG is then partitioned into several Sections according to the nodes on the critical path and the dependence relations between these nodes. In a Section, the blocks are partitioned into several Inner Wavefronts in the following stages. Finally, the execution schedule

for all P.Host and P.Mems is obtained. If the number of occupied memory processors exceeds the maximum number of processors in the PIM configuration, then the execution schedule will be modified accordingly. Algorithm 1 presents the main steps of this scheduling mechanism.

Algorithm 1. (Swing Scheduling)

[Input]

WPG=(P,E): original weighted partition dependence graph after weight is determined.

[Output]

An critical path execution order schedule CPS, where $CPS = \{CPS_1, CPS_2, \ldots, CPS_i\}$. $CPS_i = \{CP_i, IWF_i\}$ where CP_i = {Processor(b_a)} where processor is PH or PM . $IWF_i = \{PH(b_a), PM_1(b_b), PM_2(b_c), \ldots\}$ means that in *Inner Wavefront i*, $PH(b_a)$ means that block b_a will be assigned to P.Host, $PM_1(b_b)$ means that blocks b_b will be assigned to P.Mem_1, $PM_2(b_c)$ means that blocks b_c will be assigned to P.Mem_2.

[Intermediate]

W: a working set of nodes ready to be visited.
EO_temp: a working set for execution order scheduling.
iwf_temp: a working set for *Inner Wavefront* scheduling.
max_EO: the maximum number of execution order.
min_pred_O(b_i): the minimum execution order for all b_i's predecessor blocks.
max_pred_O(b_i):the maximum execution order for all b_i's predecessor blocks.
min_succ_RO(b_i):the minimum execution order for all b_i's successor blocks.
max_succ_RO(b_i):the maximum execution order for all b_i's successor blocks.
PHW(b_i): the weight of b_i for P.Host.
PMW(b_i): the weight of b_i for P.Mem.
$\text{Rank}_u(b_i)$: the trace up value of b_i used for finding CP
$\text{Rank}_d(b_i)$: the trace down value of b_i used for finding CP

[Method]

Step 1: Call "*Initialization()"* to initialize the algorithm and determine the weights of each block.
Step 2: Call "$Rank_d$*_Exec_Order_Det()*" to determine the Rank_d and Execution order of each block.
Step 3: Call "Rank_u_Det()" to determine the Rank_u of each block.
Step 4: Call "*Critical_Path_Det()*" to determine the blocks which is belong to the Critical Path.
Step 5: Call "*Critical_Path_Block_Sch()*" to find out the *Section* and schedule the Critical Path Block in each *Section* to the a suitable processor.
Step 6: Call "*Inner_Wavefront_Sch()*" to partition the blocks which is belong to the same *Section* into several *Inner Wavefront* and schedule the blocks in the same *Inner Wavefront* to the suitable processors.
Step 7: Call "*Generate_Schedule()*" to generate the execution schedule, CPS.
Step 8: If the occupied processor number is larger than the maximum processor number, call "Modify_schedule()" to modify the original execution schedule to fit the processor number, else Stop the algorithm.

The algorithm includes eight major steps. In Step 1, the algorithm calls *"Initialization()"* to initiate the necessary variables and determine the P.Host and P.Mem weights of each blocks determined by the weight evaluation mechanism.

Swing algorithm adopts the critical path method to partition WPG into *Sections*. Therefore, the critical path and the blocks on the critical path must be determined. Then the attributes, $rand_u$ and $rank_d$, of block b_i in WPG are defined by the following equations.

$$rank_u(b_i) = PMW(b_i) + \max_{b_j \in succ(b_i)}(rank_u(b_j))$$

$$rank_d(b_i) = \max_{b_j \in pred(b_i)}\{rank_d(b_j) + PMW(b_j)\}$$

Here, $succ(b_i)$ and $pred(b_i)$ represent all of the successors and predecessors of b_i, respectively. The *critical path* is defined as the following equation.

A block b_i is on the *critical path*, if and only if $rank_u(b_i) + rank_d(b_i) = rank_u(b_s)$, where b_s is the start block of the WPG, and b_i is called the *critical path block.*

According to the above definitions, the *critical path* and the *critical path block* can be determined from Step 2 to Step 4. Step 2 calls *"Rank$_d$_Exec_Order_Det()"* to determine $rank_d$ and the execution order of each block. Step 3 calls *"Rank$_u$_Det()"* to determine $rank_u$ of each block. Then, the algorithm calls "*Critical_Path_Det()*" to determine which blocks are *critical path blocks* in Step 4.

```
Subroutine: Critical_Path_Det()
    CP = (rank_d(b_s)),where b_s is the start block of WPG
    CP_num_sec=0
    for i=1 to max_EO do
        store all of b_i whose O_i =i in EO_temp
        for each block b_i ∈ EO_temp do
            if (rank_d(b_i)+rank_u(b_i))=CP then
                CP_num_sec=CP_num_sec+1
                CP_O(CP_num_sec )= O(b_i)
                CP_temp(CP_num_sec )= b_i
        end for
end for
```

```
Subroutine: Rank_u_Det()
    W=P-{b_e},where b_e is the end block of the WPG
    RO(b_e)=1
    done = False
    while done = False AND W≠φ do
        done=True
        for each b_i ∈ W do
            if min_succ_RO(b_i)=0 then
                done=False
            else
                rank_u(b_i) = PMW(b_i) + max_{b_j∈succ(b_i)} (rank_u(b_j))
                RO_i= max_succ_RO(b_i)+1
                W=W-{ b_i }
            end if
        end for
end while
```

```
Subroutine: Critical_Path_Det()
    CP = (rank_d(b_s)),where b_s is the start block of WPG
    CP_num_sec=0
    for i=1 to max_EO do
    store all of b_i whose O_i=i in EO_temp
    for each block b_i ∈ EO_temp do
       if (rank_d(b_i)+rank_u(b_i))=CP then
          CP_num_sec=CP_num_sec+1
          CP_O(CP_num_sec )= O(b_i)
          CP_temp(CP_num_sec )= b_i
    end for
end for
```

Fig. 3 illustrates the WPG of the synthetic program, which is processing in stages stated above. In this WPG, the shadow blocks are on the critical path. When the critical path is determined in Step 5, "*Critical_Path_Block_Sch()*" is called to partition all blocks in the WPG into several *Sections*. Fig. 4 illustrates the result of the given WPG, which is partitioned into five *Section*s, *Section*1:{b1}, *Section* 2: {b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14}, *Section* 3:{b15}, *Section* 4: {b16, b17, b18, b19, b20, b21, b22, b23, b24, b25, b26, b27, b28} and *Section* 5:{b29}. The execution order of *Sections* is governed by their dependence relations. After the critical path block is identified, the remaining blocks are partitioned into several *Inner Wavefronts* according to the order of execution and the dependence relations. In Fig. 4, *Section* 2 of the WPG is used to explain how blocks are scheduled in a *Section*. Since b2 is the block on critical path in *Section* 2, "*Critical_Path_Block_Sch()*" is firstly used to schedule b2 to reduce the waiting and synchronization frequencies. The remaining blocks are partitioned into three wavefronts according to the O_i of each block, by calling *"Inner_Wavefront_Sch()"* in Step 6. Finally, iw1={b3, b4, b5, b6}, iw2={b7, b8, b9}, iw3={b10, b11, b12, b13} are determined.

```
Subroutine: Critical_Path_Block_Sch()
    i=1, k=0
    while k ≤CP_num_sec do
       k=CP_O(i)
       if PHW(CP_temp( i ))- PMW(CP_temp( i ))< 0 then
          CP_k={PH(CP_temp(i))}
          PH_Used=true
          PM1_Used=false
       else
          CP_k={PM_1(CP_temp(i))}
          PH_Used=false
          PM1_Used=true
       end if
       i=i+1
    end while
```

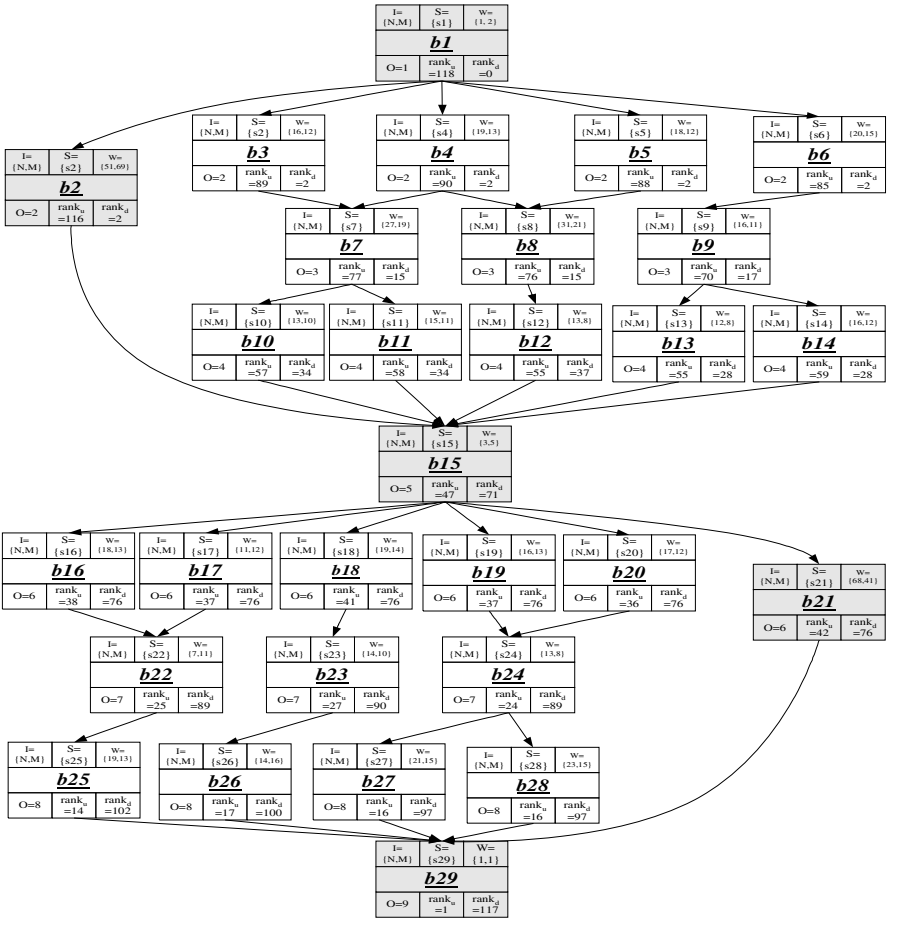


Fig. 3. WPG of a synthetic example

Section 2={b2,b3,b4,b5,b6,b7,b8,b9,,b10,b11,b12,b13}
Critical path ={b2}

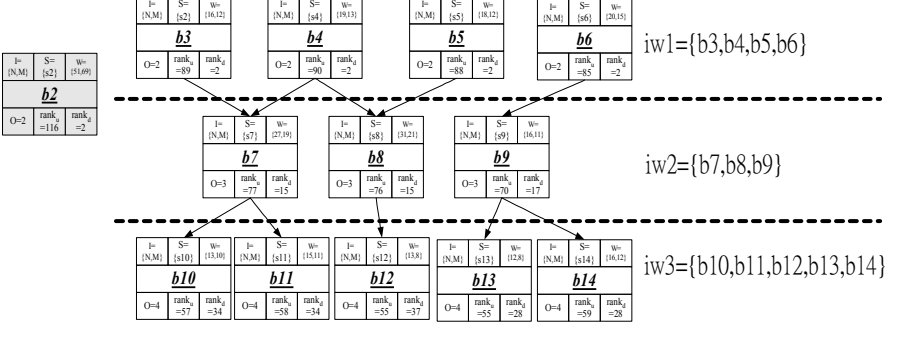


Fig. 4. Scheduled WPG of *Section* 2

CPS = {CPS1 , CPS2 , CPS3 , CPS4 , CPS5}
={{CP1 , IWF1}, {CP2 , IWF2}, {CP3 , IWF3}, {CP4 , IWF4}, {CP5 , IWF5}}
CPS_1 : ***/*Section 1*/***
CP_1={PH(b1)},
IWF_1={ϕ}
CPS_2 : ***/*Section 2*/***
CP_2={PH(b2)},
IWF_2={iwf1, iwf2, iwf3} ={{PM_1(b3), PM_2(b4), PM_3(b5), PM_4(b6)}, {PM_1(b7), PM_2(b8), PM_3(b9)}, {PM_1(b10), PM_2(b11), PM_3(b12), PM_4(b13), PM_5(b14)}}
CPS_3 : ***/*Section 3*/***
CP_3={PH(b15)},
IWF_3={ϕ }
CPS_4 : ***/*Section 4*/***
CP_4={PM_1(b21)},
IWF_2={iwf1, iwf2, iwf3} ={{PH(b16), PM_1(b17), PM_2(b18), PM_3(b19), PM_4(b20)}, {PH(b22), PM_1(b23), PM_2(b24)}, {PH(b25), PM_1(b26), PM_2(b27), PM_3(b28)}}
CPS_5 : ***/*Section 5*/***
CP_5={b29}, IWF_5={ϕ}

Fig. 5. Output of the Swing scheduling algorithm

In Step 7, the execution schedule is generated by "*Generate_Schedule()*", as shown in Fig. 5. and Fig. 6 shows the graphical execution schedule. The shaded blocks the Fig. 5 represent the execution latency. The blank blocks indicate that the processor is waiting for other processors to synchronize. The bold and dotted lines determine the point of synchronization of *Section* and *Inner Wavefront* respectively.

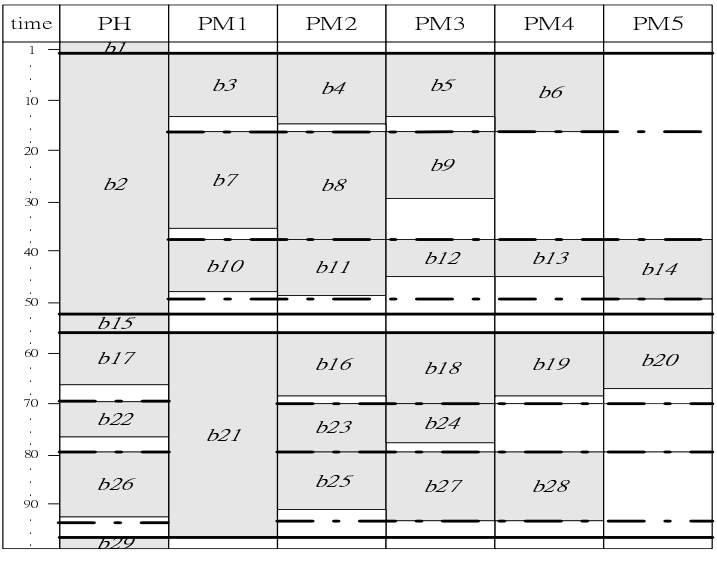


Fig. 6. Graphical execution schedule of the given example

Sometimes, the execution schedule may occupy more processors than are present in the architectural configuration. Therefore, Step 8 calls *"Modify_schedule()"* to modify the execution schedule as necessary.

4 Experimental Results

The code generated by our Octans system is targeted on our PIM simulator that is derived from the FlexRAM simulator developed by the IA-COMA Lab. at UIUC [13]. Table 1 lists the major architectural parameters. In this experiment, the configuration of one P.Host with many P.Mem processors is modeled to reflect the benefits of the multiple memory processors.

This experiment utilizes multiple P.Mem processors in the PIM architecture to improve performance. The evaluated applications include five benchmarks: *cg* is from the serial version of NAS; *swim* is from SPEC95; *strsm* is from BLAS3; *TISI* is from Perfect Benchmark, and *fft* is from [45].

Table 2 and Fig. 7 summarize the experimental results. "Standard" denotes that the application is executed in P.Host alone. This experiment concerns a general situation of a uniprocessor system, and is used to compare speedup. "1-P.Mem" implies that the application is transformed and scheduled by the simplified Swing Scheduling for the one-P.Host and one-P.Mem configuration of the PIM architecture. "n-P.Mem" implies that the application is transformed and scheduled by Swing Scheduling mechanism for the one P.Host and multiple P.Mem configuration of the PIM architecture.

Table 2 and Fig. 7 indicate that *swim* and *cg* have quite a good speedup when the Swing Scheduling mechanism is employed because these programs contain many memory references and few dependence relations. Therefore, the parallelism and memory access performance can be improved by using more memory processors. Applying the 1-P.Mem scheduling mechanism can also yield improvements. *strsm* exhibits an extremely high parallelism but a rather few memory access, so the Swing Scheduling mechanism is more suitably adopted than the 1-P.Mem scheduling mechanism. *TISI* cannot generate speedup when the 1-P.Mem scheduling mechanism is applied, since it is a typical CPU bounded program, and involves many dependencies. The Swing Scheduling mechanism can be suitably used to increase speedup. Finally, in *fft*, the program is somewhat computation-intensive and

Table 2. Execution cycles of five benchmarks

Bench-mark	Standard	1-P.Mem Scheduling	n-P.Mem Scheduling	Speedup		
				1-P.Mem Scheduling	n-P.Mem Scheduling	n(Occupied P.Mem)
swim	228289321	116669760	52168027	1.96	4.38	6
cg	91111840	51230772	32124287	1.78	2.84	4
strsm	703966766	489967053	187989176	1.44	3.74	5
TISI	133644087	173503404	91098174	0.77	1.47	2
fft	117998621	101841407	110399171	1.16	1.07	2

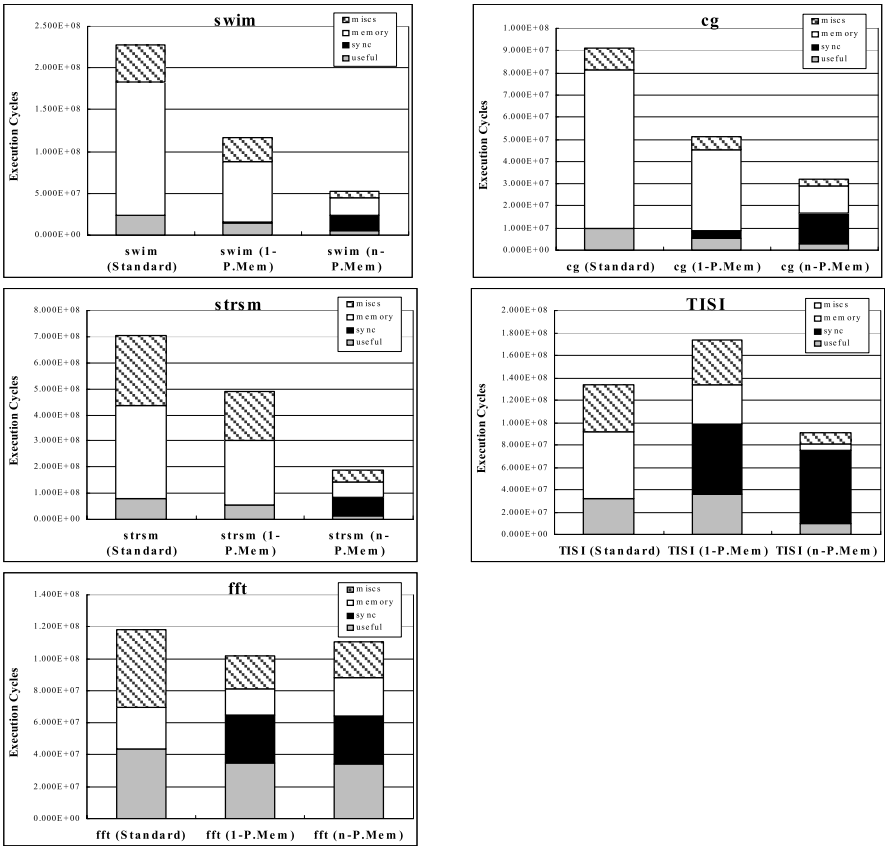


Fig. 7. Execution times of five benchmarks obtained by Standard, 1-P.Mem and n-P.Mem settings.

sequential, and therefore only a little speedup can be improved after the 1-P.Mem scheduling mechanism is applied. However, an additional overhead is generated when the Swing Scheduling mechanism is applied. Accordingly, 1-P.Mem and Swing scheduling mechanisms are suitable for different situations. Choosing the 1-P.Mem or Swing scheduling mechanism more heuristically in the scheduling stage of the Octans system will improve performance.

5 Conclusions

This study proposes a new scheduling mechanism, called Swing Scheduling, with Octans system for a new class of high-performance chip multiprocessor architectures, Processor-in-Memory, which consists of a host processor and many memory processors. The Octans system partitions source code into blocks by statement splitting; estimates the weight (execution time) of each block, and then schedules each block to the most suitable processor for execution. Five real benchmarks, swim, TISI,

strsm, cg, and fft were experimentally considered to evaluate the effects of the Swing Scheduling. In the experiment, the performance was improved by a factor of up to 4.38 while using up to six P.Mems and one P.Host. The authors believe that the techniques proposed here can be extended to run on DIVA, EXECUBE, FlexRAM, and other high-performance chip multiprocessor architectures by slightly modifying the code generator of the Octans system.

Acknowledgements

This work is supported in part by the National Science Council of Republic of China, Taiwan under Grant NSC 96-2221-E-033 -019-

References

[1] Blume, W., Eigenmann, R., Faigin, K., Grout, J., Hoeflinger, J., Padua, D., Petersen, P., Pottenger, B., Rauchwerger, L., Tu, P., Weatherford, S.: Effective Automatic Parallelization with Polaris. International Journal of Parallel Programming (May 1995)

[2] Chu, S.L.: PSS: a novel statement scheduling mechanism for a high-performance SoC architecture. In: Proceedings of Tenth International Conference on Parallel and Distributed Systems, pp. 690–697 (July 2004)

[3] Crisp, R.: Direct Rambus Technology: the New Main Memory Standard. In: Proceedings of IEEE Micro, pp. 18–28 (November 1997)

[4] Hall, M., Anderson, J., Amarasinghe, S., Murphy, B., Liao, S., Bugnion, E., Lam, M.: Maximizing Multiprocessor Performance with the SUIF Compiler. IEEE Computer (December 1996)

[5] Hall, M., Kogge, P., Koller, J., Diniz, P., Chame, J., Draper, J., LaCoss, J., Granacki, J., Brockman, J., Srivastava, A., Athas, W., Freeh, V., Shin, J., Park, J.: Mapping Irregular Applications to DIVA, a PIM-Based Data-Intensive Architecture. In: Proceedings of 1999 Conference on Supercomputing (January 1999)

[6] Judd, D., Yelick, K.: Exploiting On-Chip Memory Bandwidth in the VIRAM Compiler. In: Proceedings of 2nd Workshop on Intelligent Memory Systems, Cambridge, MA (November 12, 2000)

[7] Kang, Y., Huang, W., Yoo, S., Keen, D., Ge, Z., Lam, V., Pattnaik, P., and Torrellas, J.: FlexRAM: Toward an Advanced Intelligent Memory System. In: Proceedings of International Conference on Computer Design (ICCD), Austin, Texas (October 1999)

[8] Landis, D., Roth, L., Hulina, P., Coraor, L., Deno, S.: Evaluation of Computing in Memory Architectures for Digital Image Processing Applications. In: Proceedings of International Conference on Computer Design, pp. 146–151 (1999)

[9] Oskin, M., Chong, F.T., Sherwood, T.: Active Page: A Computation Model for Intelligent Memory. Computer Architecture. In: Proceedings of the 25th Annual International Symposium on Computer Architecture, pp. 192–203 (1998)

[10] Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Tomas, R., Yelick, K.: A Case for Intelligent DRAM. IEEE Micro, pp. 33-44 (March/April 1997)

[11] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in Fortran 77. Cambridge University Press, Cambridge (1992)

[12] Snip, A. K., Elliott, D.G., Margala, M., Durdle, N.G.: Using Computational RAM for Volume Rendering. In: Proceedings of 13th Annual IEEE International Conference on ASIC/SOC, pp. 253 –257 (2000)
[13] Swanson, S., Michelson, K., Schwerin, A., Oskin, M.: WaveScalar. MICRO-36 (December 2003)
[14] Veenstra, J., Fowler, R.: MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In: Proceedings of MAS-COTS 1994, pp. 201–207 (January 1994)
[15] Wang, K.Y.: Precise Compile-Time Performance Prediction for Superscalar-Based Computers. In: Proceedings of ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation, pp. 73–84 (1994)

Consensus-Driven Distributable Thread Scheduling in Networked Embedded Systems

Jonathan S. Anderson[1], Binoy Ravindran[1], and E. Douglas Jensen[2]

[1] Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg Virginia, 24061, USA
{andersoj,binoy}@vt.edu

[2] The MITRE Corporation
Bedford, Massachusetts, 01730, USA
jensen@mitre.org

Abstract. We demonstrate a consensus utility accrual scheduling algorithm for distributable threads with run-time uncertainties in execution time, arrival models, and node crash failures. The DUA-CLA algorithm's message complexity ($O(fn)$), lower time complexity bound ($O(D+fd+nk)$), and failure-free execution time ($O(D+nk)$) are established, where D is the worst-case communication delay, d is the failure detection bound, n is the number of nodes, and f is the number of failures. The "lazy-abort" property is shown — abortion of currently-infeasible tasks is deferred until timely task completion is impossible. DUA-CLA also exhibits "schedule-safety" — threads proposed as feasible for execution by a node which fails during the decision process will not cause an otherwise-feasible thread to be excluded. These properties mark improvements over earlier strategies in common- and worst-case performance. Quantitative results obtained from our Distributed Real-Time Java implementation validate properties of the algorithm.

1 Introduction

1.1 Dynamic Distributed Real-Time Systems

Distributed real-time systems such as those found in industrial automation, net-centric warfare (NCW), and military surveillance must support for timely, end-to-end activities. Timeliness includes application-specific acceptability of end-to-end time constraint satisfaction, and of the predictability of that satisfaction. These activities may include computational, sensor, and actuator steps which levy a causal ordering of operations, contingent on interactions with physical systems. Such end-to-end tasks may be represented in a concrete distributed system as: chains of (a) nested remote method invocations; (b) publish, receive steps in a publish-subscribe framework; (c) event occurrence and event handlers.

Dynamic Systems. The class of distributed real-time systems under consideration here, typified by NCW applications [1], is characterized by dynamically uncertain execution properties due to transient and persistent local overloads, uncertain task arrival patterns and rates, uncertain communication delays,

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 247–260, 2007.

node failures, and variable resource demands. However, while local activities in these systems may have timeliness requirements with sub-second magnitudes, end-to-end tasks commonly have considerably larger magnitudes of milliseconds to multiple minutes. Despite larger timeliness magnitudes, these activities are mission-critical and require the strongest assurances possible under the circumstances.

End-to-End Context. In order to make resource allocation decisions in keeping with end-to-end activity requirements, some representation of these parameters and the current state of the end-to-end activity must be provided. The *distributable thread* abstraction provides such an extensible model for reasoning about end-to-end activity behavior. Distributable threads (hereafter, simply *threads*) appeared first in the Alpha OS [2] and were adopted in Mach 3.0 [3] and Mk7.3 [4]. Recently, this abstraction has served as the basis for RT-CORBA 2.0 [5] and Sun's emerging Distributed Real-Time Specification for Java (DRTSJ) [6], where threads form the primary programming abstraction for concurrent, distributed activities.

Time Constraints for Overloaded Systems. In underloaded systems it is sufficient to provide an assessment of the *urgency* of an activity, typically in the form of a *deadline*. For these scenarios, known-optimal algorithms (e.g., EDF [7]) exist to meet all deadlines (given some restrictions.) When a system is overloaded, resource managers must decide *which* subset of activities to complete, and with *what degree of timeliness*.

This requires the system to be aware of the relative *importances* activities. We consider the *time/utility function* (TUF) timeliness model [8], in which the utility of completing an activity is given as a function of its completion time. This paper is confined to downward step-shaped TUFs, wherein an activity's utility is a constant U_i when the task is completed before a deadline t; no utility is gained for tasks after their deadline.

Utility Accrual Scheduling. When time constraints are expressed as TUFs, resource allocation decisions may be expressed in terms of *utility accrual* (UA) criteria. A common UA criteria is *maximize summed utility*, in which resource allocation decisions are made such that the total summed utility accrued is maximized. Several such UA scheduling and sequencing heuristics have been investigated (e.g., [9,10]). Such algorithms for activities described by downward-step TUFs equate to EDF during underload conditions, achieving optimum schedules. During overloads, these algorithms favor higher-utility activities over those with lower-utility. Resulting "best-effort" adaptive behavior exhibits graceful degradation as load increases, shedding low utility work irrespective of its urgency.

1.2 Contributions and Related Work

The central contributions of this paper are: (a) the Distributed Utility Accrual - Consensus-based Lazy Abort (DUA-CLA) scheduling algorithm; (b) bounds on DUA-CLA's timeliness, message efficiency, and optimality behavior in a variety

of conditions; (*c*) implementation of DUA-CLA in the DRTSJ middleware; and (*d*) experimental results illustrating the validity of the theoretical results.

This paper presents significant progress on work published by the authors in [11], expanding the theoretical performance envelope in two ways: First, the "lazy-abort" property is introduced (Theorem 6), relaxing conservative task-abortion behavior present in earlier work, while maintaining asymptotic execution times and performance assurances. Second, DUA-CLA is shown (see Theorem 7) to be "schedule-safe" in the presence of failures during distributed rescheduling. This property mitigates pessimism in feasibility assessments due to failures which results in unnecessary task rejection during partial failure.

The DUA-CLA algorithm represents a unique approach to distributable thread scheduling in two respects. First, it unifies scheduling with a fault-tolerance strategy. Previous work on distributable thread scheduling [12, 13] addresses only the scheduling problem, with fault tolerance dealt with by separate *thread integrity protocols* [12,13,14]. While this provides admirable separation of concerns, scheduling and integrity operations become tightly intertwined in distributed systems where failures are prevalent.

Second, DUA-CLA takes a *collaborative* approach to the scheduling problem, rather than requiring nodes independently to schedule tasks without knowledge of other nodes' states. Global scheduling approaches wherein a single, centralized scheduler makes all scheduling decisions have been proposed and implemented. DUA-CLA takes a *via media*, improving independent node scheduler decision-making with partial knowledge of global system state.

Little work has been done on collaborative distributed scheduling for real-time systems. The RT-CORBA 2.0 specification [5] envisions such an approach, enumerating it as the third of its four "cases" for distributed scheduling. Poledna, et. al. consider a consensus-based sequencing approach for operations on replicas to ensure consistency [15]. In a similar vein, Gammar and Kammoun present a consensus algorithm for ensuring database properties such as serializability and consistency in real-time transactional systems [16]. None of these directly address the question of end-to-end causal activity scheduling.

2 The DUA-CLA Algorithm

2.1 Models

Distributable Thread Abstraction. Threads execute in local and remote objects by location-independent invocations and returns. A thread begins its execution by invoking an object operation. The object and the operation are specified when the thread is created. The portion of a thread executing an object operation is called a *thread segment*. Thus, a thread can be viewed as being composed of a concatenation of thread segments.

A thread's initial segment is called its *root* and its most recent segment is called its *head*. The head of a thread is the only segment that is active. A thread can also be viewed as a sequence of *sections*, where a section consists of all

contiguous thread segments on a node. Further details of the thread model can be found in [6,5,13]. Execution time estimates (possibly inaccurate) of the sections of a thread areknown when the thread arrives.The application is thus comprised of a set of threads, denoted $\mathbf{T} = \{T_1, T_2, T_3, \ldots\}$, with sections $[S_1^i, S_2^i, \ldots, S_k^i]$.

Timeliness Model. We specify the time constraint of each thread using a TUF. A thread T_i's unit downward step TUF is denoted as $U_i(t)$, which has a initial time I_i, which is the earliest time for which the TUF is defined, a termination time X_i, which, for a downward step TUF, is its discontinuity point, and a constant utility U_t. $U_i(t) > 0, \forall t \in [I_i, X_i]$ and $U_i(t) = 0, \forall t \notin [I_i, X_i], \forall i$.

System and Failure Models. Our system and failure models follow that of [17]. We consider a system model where a set of processing *nodes* are denoted by the totally-ordered set $\Pi = \{1, 2, \ldots, n\}$. We consider a single hop network model (e.g., a LAN), with nodes interconnected through a hub or a switch. The system is assumed to be (partially) synchronous in that there exists an upper bound D on the message delivery latency. A reliable message transmission protocol is assumed; thus messages are not lost or duplicated. Node clocks are assumed to be perfectly synchronized, for simplicity in presentation, though DUA-CLA can be extended to clocks that are nearly synchronized with bounded drift rates. As many as f_{max} nodes may crash arbitrarily. The actual number of node crashes is denoted as $f \leq f_{max}$. Nodes that do not crash are called *correct*.

Each node is assumed to be equipped with a perfect failure detector [18] that provides a list of nodes deemed to have crashed. If a node q belongs to such a list of node p, then node p is said to *suspect* node q. The failure detection time [19] $d \leq D$ is bounded. Similar to [17], for simplicity in presentation, we assume that D is a multiple of d. Failure detectors are assumed to be (a) *accurate*—i.e., a node suspects node q only if q has previously crashed; and (b) *timely*—i.e., if node q crashes at time t, then correct nodes permanently suspect q within $t + d$.

2.2 Rationale and Design

Our primary scheduling objective is to maximize total utility accrued by all the threads Further, the algorithm must provide assurances on the satisfaction of thread termination times in the presence of (up to f_{max}) crash failures. Moreover, the algorithm must exhibit the UA best-effort property described in Section 1.

Definition 1 (Current and Future Head Nodes). *The current head node of a thread T_i is the node where T_i is currently executing (i.e., where T_i's head is currently located). The future head nodes of a thread T_i are those nodes where T_i will make remote invocations in the future.*

The crash of a node p affects other nodes in the system in three possible ways: (a) p may be the current head node of one or more threads; (b) p may be the future head node of one or more threads; and (c) p may be the current and future head node of one or more threads.

If p is only the current head node of one or more threads, then all its future head nodes are immediately affected when p crashes, since they can now release

allocated processor time. This implies that when a node p crashes, a system-wide decision must be made regarding which subset of threads are eligible for execution in the system—referred to as an *execution-eligible thread set*. This decision must be made in the presence of failures since nodes may crash while that decision is being made. We formulate this problem as a *consensus* problem [20] with the following properties: (a) If a correct node decides an eligible thread set $\mathcal{T}$, then some correct node proposed $\mathcal{T}$;[1] (b) Nodes (correct or not) do not decide different execution-eligible sets (*uniform agreement*); (c) Every correct node eventually decides (i.e., termination).

How can a node propose a set of threads which are eligible for execution? The task model is dynamic and future scheduling events cannot be considered at a scheduling event.[2] Thus, the execution-eligible thread set must be constructed exploiting the current system knowledge. Since the primary scheduling objective is to maximize the total thread accrued utility, a reasonable heuristic for determining the execution-eligible thread set is a "greedy" strategy: Favor "high return" threads over low return ones, and complete as many of them as possible before thread termination times.

The potential utility that can be accrued by executing a thread section on a node defines a measure of that section's "return on investment." We measure this using a metric called the *Potential Utility Density* (or PUD). On a node, a thread section's PUD measures the utility that can be accrued per unit time by immediately executing the section on the node.

Thus, each node iteratively examines thread sections in its local ready queue for potential inclusion in a feasible (local) schedule in order of decreasing section PUDs. For each section, the algorithm examines whether that section can be completed early enough, allowing successive sections of the thread to also be completed early enough, to allow the entire thread to meet its termination time. We call this property the *feasibility* of a section. Infeasible sections are not included in the working schedule. This approach requires a decomposition of the thread's deadline, which is computed at arrival time using the following conservative approach: The section termination times of a thread T_i with k sections are given by:

$$S_j^i.tt = \begin{cases} T_i.tt & j = k \\ S_{j+1}^i.tt - S_{j+1}^i.ex - D & 1 \leq j \leq k-1 \end{cases} \quad (1)$$

where $S_j^i.tt$ denotes section S_j^i's termination time, $T_i.tt$ denotes T_i's termination time, and $S_j^i.ex$ denotes the estimated execution time of section S_j^i.

Thus, the local schedule constructed by a node p is an ordered list of a subset of sections in p's ready queue that can be feasibly completed, and will likely result in high local accrued utility (due to the greedy nature of the PUD heuristic). The set of threads, say T_p, of these sections included in p's schedule is proposed

[1] This property is stronger than the conventional *Uniform Validity* property, and therefore requires additional constraints.

[2] A "scheduling event" is any event that invokes the scheduling algorithm.

by p as those that are eligible for system-wide execution, from p's standpoint. However, not all threads in T_p may be eligible for system-wide execution, because the current and/or future head nodes of some of those threads may crash. Consequently, the set of threads that are eligible for system-wide execution is that subset of threads with no absent sections from their respective current and/or future head node schedules.

2.3 Algorithm Description

The DUA-CLA algorithm that we present is derived from Aguilera *et. al*'s time-optimal, early-deciding, uniform consensus algorithm [17]. A pseudo-code description of DUA-CLA on each node i is shown in Algorithm 1.

Algorithm 1. DUA-CLA: Code for each node i

```
1  input: σ_r^i; output: σ_i; // σ_r^i: unordered ready queue of node i's sections; σ_i:
       schedule
2  Initialization: Σ_i = ∅; ω_i = ∅; max_i = 0;
3  σ_i = ConstructLocalSchedule(σ_r^i);
4  send(σ_i, i) to all;
5  upon receive (σ_j, j) until 2D do // After time 2D, consensus begins
6     Σ_i = Σ_i ∪ σ_j;
7  ω_i = DetermineSystemWideFeasibleThreadSet(Σ_i);
8  upon receive (ω_j, j) do
9     if j > max_i then max_i = j; ω_i = ω_j;
10 at time (i − 1)d do
11    if suspect j for any j : 1 ≤ j ≤ i − 1 then
12       ω_i = UpdateFeasibleThreadSet(Σ_i);
13       send(ω_i, i) to all;
14 at time (j − 1)d + D for every j : 1 ≤ j ≤ n do
15    if trust j then decide ω_i;
16 UpdateSectionSet(ω_i, σ_r^i);
17 σ_i = ConstructLocalSchedule(σ_r^i);
18 return σ_i;
```

The algorithm is invoked at a node i at the scheduling events including 1) creation of a thread at node i and 2) inclusion of a node k into node i's suspect list by i's failure detector.

When invoked, a node i first constructs a local schedule (`ConstructLocalSchedule()`), sending this schedule (σ_i, i) to all nodes. Recipients respond immediately by constructing local section schedules and sending them to all nodes. When node i receives a schedule (σ_j, j), it includes that schedule into a schedule set Σ_i. Thus, after $2D$ time units, all nodes have a schedule set containing all schedules received.

A node i then determines its consensus decision, computed from Σ_i as the subset of threads with no sections absent from node schedules in Σ_i. Node i uses a variable ω_i to maintain its consensus decision.

The algorithm divides time in rounds of duration d, where the ith round corresponds to the time interval $[(i-1)d, id)$. At the beginning of round i, node i checks whether it suspects *any* of the nodes with smaller node ID. If so, it computes a new ω_i using `UpdateFeasibleThreadSet()` (see Algorithm 2),

sending (ω_i, i) to all nodes. Note that the messages sent in a round could be received in a higher round since $D > d$.

Algorithm 2. `UpdateFeasibleThreadSet`

```
1: input: ω_i; output: ω'_i;      // ω'_i: feasible section set with sections on failed
     nodes removed
2: initialize:  ω'_i = ω_i
3: for each section S^t_j ∈ ω_i do
4:    if  suspect j then ω'_i = ω'_i \ S^t_j;
5:    return σ'_i;
```

Each node i maintains a variable max_i that contains the largest node ID from which it has received a consensus proposal. When a node i receives a proposed execution-eligible thread set (ω_j, j) that is sent from another node j with an ID that is larger than max_i (i.e., $j > max_i$), node i updates its consensus decision to thread set ω_j and max_i to j. At times $(j-1)d + D$ for $j = 1, \ldots, n$, node i is guaranteed to have received potential consensus proposals from node j. At these times, i checks whether j has crashed; if not, i arrives at its consensus decision on the thread set ω_i.

Node i then updates its ready queue σ_r^i by removing those sections whose threads are absent in the consensus decision ω_i. The updated ready queue is used to construct a new local schedule σ_i, the head section of which is subsequently dispatched for execution.

2.4 Constructing Section Schedules

We now describe the algorithm `ConstructLocalSchedule()` and its auxiliary functions. Since this algorithm is not a distributed algorithm *per se*, we drop the suffix i from notations σ_r^i (input unordered list) and σ_i (output schedule), and refer to them as σ_r and σ, respectively. Sections are referred to as S_i, for $i = 1, 2, \ldots$

Algorithm 3 describes the local section scheduling algorithm. When invoked at time t_{cur}, the algorithm first checks the feasibility of the sections. First, if the earliest conceivable execution (the current time) of segment will still miss the termination time, the algorithm aborts the section. If the earliest predicted completion time of a section is later than its termination time, it is removed from this round's consideration. The sections considered for insertion into σ in order of decreasing PUD, which is maintained in order of non-decreasing section termination times. After inserting a section S_i, the schedule σ is tested for feasibility.[3] If σ becomes infeasible, S_i is removed. After examining all sections, the ordered list σ is returned.

Algorithm 3 includes those sections likely to result in high total utility (due to the greedy nature of the PUD heuristic). Further, since the invariant of schedule

[3] A schedule σ is feasible if the predicted completion time of each section $S_i \in \sigma$ does not exceed S_i's termination time. For explicit pseudo-code for a linear-time implementation, see Algorithm 3 in [11].

Algorithm 3. `ConstructLocalSchedule()`

```
1:  input: σ_r; output: σ;
2:  Initialization: t := t_cur, σ := ∅;
3:  for each section S_i ∈ σ_r do
4:      if current_time + S_i.ex > S_i.tt then
            abort(S_i)
5:      if S^i_{j-1}.tt + D + S^i_j.ex > S^i_j.tt then
6:          σ_r = σ_r \ S_i;
        else
7:          S_i.PUD = U_i (t + S_i.ex) / S_i.ex;
8:  σ_tmp := sortByPUD(σ_r);
9:  for each section S_i ∈ σ_tmp from head to tail do
10:     if S_i.PUD > 0 then
11:         Insert(S_i, σ, S_i.tt);
12:         if Feasible(σ)=false then
13:             Remove(S_i, σ, S_i.tt);
14:     else break;
15: return σ;
```

feasibility is preserved throughout the examination of sections, the output schedule is always a feasible schedule. During underloads, schedule σ will always be feasible in (Algorithm 3), the algorithm will never reject a section, and will produce a schedule which is the same as that produced by EDF (where deadlines are equal to section termination times). This schedule will meet all section termination times during underloads. During overloads, one or more low-PUD sections will not be included. These rejected sections are less likely to contribute a total utility larger than that contributed by accepted sections. The asymptotic complexity of Algorithm 3 is dominated by the nested loop with calls `Feasible()`, resulting in a cost of $O(k^2)$.

3 Algorithm Properties

We now establish DUA-CLA's timeliness and execution time properties in both absence and presence of failures. We first describe DUA-CLA's timeliness property under crash-free runs. The proof of this and some future results are elided for space, but may be found in the full version of the paper.[4] [17]

Theorem 1. *If all nodes are underloaded and no nodes crash (i.e., $f_{max} = 0$), DUA-CLA meets all thread termination times, yielding optimum total utility.*

Theorem 2. *DUA-CLA achieves (uniform) consensus (i.e., uniform validity, uniform agreement, termination) on the system-wide execution-eligible thread set in the presence of up to f_{max} failures.*

Theorem 3. *DUA-CLA's time complexity is $O(D + fd + nk)$ and message complexity is $O(fn)$.*

[4] Full paper available at: `http://www.real-time.ece.vt.edu/euc07.pdf`

Theorem 4. *If $n-f$ nodes (i.e., correct nodes) are underloaded, then DUA-CLA meets the termination times of all threads in its execution-eligible thread set.*

To establish the algorithm's best-effort property (Section 1), we define NBI:

Definition 2. *Consider a distributable thread scheduling algorithm $\mathcal{A}$. Let a thread T_i be created at a node at a time t with the following properties: (a) T_i and all threads in $\mathcal{A}$'s execution-eligible thread set at time t are not feasible (system-wide) at t, but T_i is feasible just by itself; and (b) T_i has the highest PUD among all threads in $\mathcal{A}$'s execution-eligible thread set at time t. Now, $\mathcal{A}$'s non-best-effort time interval, denoted $NBI_{\mathcal{A}}$, is defined as the duration of time that T_i will have to wait after t, before it is included in $\mathcal{A}$'s execution-eligible thread set. Thus, T_i is assumed to be feasible at $t + NBI_{\mathcal{A}}$.*

We now describe the NBI of DUA-CLA and other distributable thread scheduling UA algorithms including DASA [9], LBESA [10], and AUA [12] under crash-free runs.

Theorem 5. *Under crash-free runs (i.e., $f_{max} = 0$), the worst-case NBI of DUA-CLA is $3D + \delta$, DASA's and LBESA's is δ, and that of AUA is $+\infty$.*

In order to further characterize the algorithm's best-effort behavior in the presence of failures, we introduce definitions for *Lazy-Abort* behavior and *Schedule Safety*:

Definition 3. *A collaborative distributable thread scheduling algorithm is said to Lazy-Abort if it delays abortion of a segment until it would be infeasible if it were the only thread in the system.*

Theorem 6. *DUA-CLA demonstrates Lazy-Abort behavior. Sections are only aborted in `ConstructLocalSchedule()` (Algorithm 3), and then only when the segment would exceed its deadline if it were executed immediately. If this is the case, the Lazy-Abort condition is met. Consequently, transient perceived overloads which resolve through node failures or pessimistic execution time evaluations do not cause overly-aggressive abortion of future threads.*

Definition 4. *A consensus-based distributable thread scheduling algorithm is said to exhibit Schedule Safety if it never allows the presence of a remote segment S_f in the global feasible set to render infeasible a local segment S_l if the node hosting S_f is known to have failed during consensus.*

Theorem 7. *DUA-CLA demonstrates schedule-safety despite failures during the distributed scheduling event. The algorithm evaluates feasibility of local segments on node i based on the section set updated in the call to `UpdateSectionSet()` in Algorithm 1. If any nodes j with $1 < j < i$ is suspected by i, then i removes all segments on j from ω_i. (Furthermore, at time $D + fd$, all nodes will receive this reduced proposed section set.) Therefore, no locally feasible thread segments will be rendered infeasible by erroneous inclusion of segments from j.*

4 Implementation Experience

A major objective this work was to bridge the gap between theoretical consideration and the practicalities of implementation. In particular, we constructed experiments to uncover time complexity constants implicit in Theorem 3. Single-node task sets are compared to distributed sets in the presence of underloads as well as overloads, in failure-free as well as the $f = f_{max}$ case. We explore algorithm overhead by comparing well-known single-node scheduling disciplines to a degenerate case of our collaborative approach.

DUA-CLA was implemented on the DRTSJ reference implementation (DRTSJ-RI), consisting of Apogee's Aphelion-based DRTSJ-compliant JVM, executing on Pentium-class machines with Ubuntu Edgy Linux (kernel 2.6.17 with real-time extensions). Nodes were connected via 10Mbps Ethernet through a Linux-based dynamic switch, configured with the `netem` network emulation module [21] to introduce controlled communication delay. Failure detector traffic, experimental control traffic, and normal communication traffic were allocated to priority bands configured to simulate communication delays consonant with the system model described in Section 2.1, resulting in particular in the relationship $D \gg d$ between common communication latency and failure detection latency.

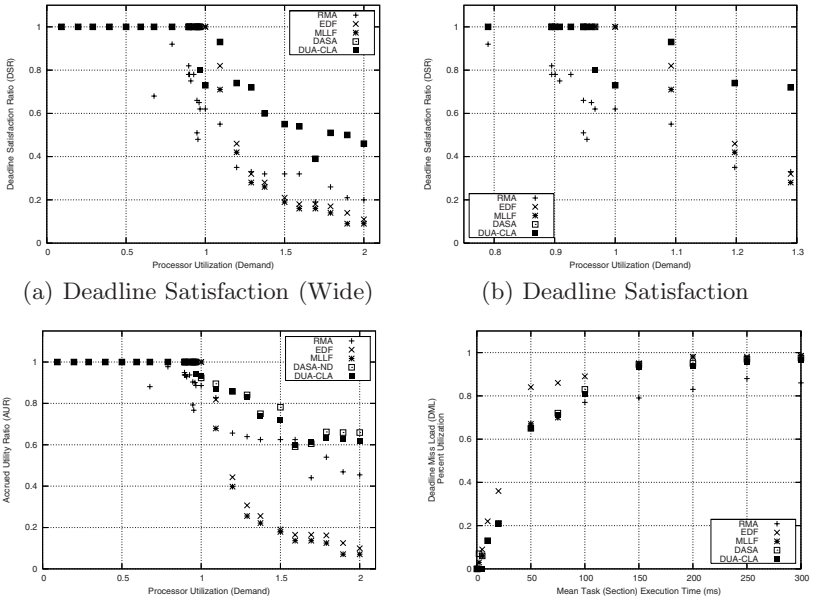
A heartbeat fast-failure detector was implemented as a small, pure RTSJ application [22] with the highest execution eligibility on the node, and not preemptible by the garbage collector. A heartbeat period of 1ms and evaluation period of 3ms were chosen to match the magnitudes of task execution times. Extensive measurements of latency d and application message delay D were made across a range of CPU and network utilization, with no failure detection latency greater than 2.98 milliseconds; therefore we use $d = 2.98$ms for the following DUA-CLA experiments. Similarly, we measured a worst-case message delay $D \approx 69.87$ms. Both latencies are stable across a range of CPU utilization, closely approximating a perfect fast failure detector.

With this detector, the DUA-CLA algorithm was implemented on top of the DRTSJ Metascheduler, a pluggable scheduling framework enabling user-space scheduling implementations in DRTSJ [6], and previously on QNX [23].

Our experiments took place in the testbed described above, with one DRTSJ-compliant JVM instantiated on each node. Node clocks were synchronized using NTP [24]. The dynamic switch was configured to insert normally-distributed communication delay in application communication traffic. All application communication was via UDP, with reliable messaging provided by the application.

Local Scheduler Performance. In order to establish a baseline for assessing the performance of our scheduler implementation, we compared DUA-CLA to a variety of other scheduling algorithms. In these experiments, each submitted thread consisted of a single segment to be executed on the local node. Since no remote segments appeared in the ready set, no remote scheduling event was triggered and DUA-CLA is functionally equivalent to Clark's DASA algorithm.

Figures 1(a) and 1(b) illustrate deadline satisfaction performance of some UA and non-UA scheduling policies. We use Deadline Satisfaction Ratio (DSR), the



(a) Deadline Satisfaction (Wide)

(b) Deadline Satisfaction

(c) Local Accrued Utility

(d) Local Deadline Miss Load

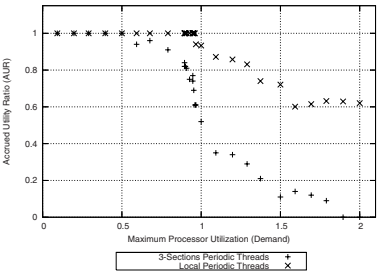
Fig. 1. Local Scheduler Performance: Deadline Satisfaction, AUR, and DML

ratio of jobs which satisfy their deadline to the total number of jobs released. In the case of this non-distributed experiment, a job is exactly equivalent to a thread segment. A collection of 5 periodic threads were created with relatively-prime periods and random phase offsets. Mean execution time for each job release was varied producing processor demands ranging from 0 to 200%.

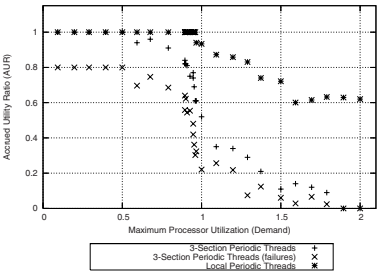
The schedulers presented here include Rate Monotonic Analysis (RMA), Earliest Deadline First, Modified Least-Laxity First (MLLF), Dependent Activity Scheduling Algorithm (DASA), and DUA-CLA. Of these, only DASA and DUA-CLA are utility accrual algorithms. Each algorithm was implemented in the Metascheduler, and each was run with an identical task set for each utilization.

Figure 1(b) provides a detailed look at the "deadline-miss load" region from Figure 1(a), the utilization range at which the scheduling policies begin missing activity deadlines. Theoretically, each of the schedulers shown (with the exception of RMA) should obtain 100% DSR up to 100% load. However, due to middleware overhead activities miss deadlines at lower CPU utilizations. Understanding this overhead as we consider more complex scheduling policies is critical to engineering systems which appropriately trade off scheduling "intelligence" against the additional overhead incurred by more complex policies.

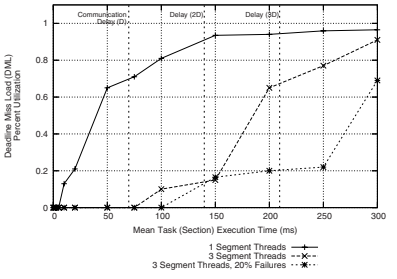
Figure 1(c) captures scheduler performance measured against the UA metric Accrued Utility Ratio (AUR). AUR is the ratio of the accrued utility (sum of the U_i for all completed jobs) to the utility available. Since we have chosen unit-downward step TUFs for these experiments, the AUR and DSR are similar



(a) AUR for 3-Segment Threads



(b) AUR with $f = f_{max} = 0.2n$



(c) Deadline Miss Load for DUA-CLA

Fig. 2. Distributed Scheduler Performance

metrics, with AUR appearing as a weighted form of the DSR, with weights U_i. The reader will note that, while Figures 1(a) and 1(b) indicate that the non-UA policies like RMA sometimes outperform DASA and DUA-CLA in the DSR metric during overloads, Figure 1(c) shows us that this is because RMA is dropping the "wrong" tasks, while the UA policies favor high-utility tasks. It is precisely this behavior we wish to explore in the distributed case, in particular understanding the additional overhead incurred.

Finally, Figure 1(d) characterizes scheduler overhead for each policy by measuring Deadline Miss Load (DML). For each data point, a task execution time (the plot's x-axis) was fixed for every job during a single run. Periods of each periodic task were varied, measuring the resulting utilization and deadline satisfaction. The DML (the y-axis) is that greatest utilization for which the scheduling policy was able to meet all deadlines. The theoretical (zero-overhead) behavior during underload for each policy is a DML of 1.0: these policies should never miss a deadline until the CPU is saturated.

Distributed Scheduler Performance. Our final set of experiments sought to establish the concrete behavior of the DUA-CLA algorithm for qualitative comparison to local scheduling approaches, for validation of the theoretical results above, and to investigate execution timescales for which consensus thread scheduling is appropriate. In each trial, each three-segment periodic thread originates on a common origin node with a short segment, makes an invocation onto one of many server nodes to execute a second segment, then returns to the origin

node to complete in a final segment. We fix the periods, and vary the execution times to produce the utilizations in Figure 2.

In Figure 2(a), we compare the AUR of a collection of one-segment (local) threads to a collection of three-segment threads. As can be seen from the plot, the penalty incurred by collaboration is significant, but the scheduling policy continues to accrue utility through 1.8 fractional utilization. Furthermore, Theorem 1 is borne out by the underloaded portion of Figure 2(a), modulo scheduling overhead. This overhead is explored in detail in the discussion of Figure 2(c).

The behavior of DUA-CLA in the presence of failures is shown in Figure 2(b), wherein we fail f_{max} nodes. Again, the performance of the scheduler suffers, but as shown in Theorem 4, our implementation meets the termination times for all threads remaining on correct nodes.

Finally, we investigate overhead incurred by DUA-CLA across a selection of mean task execution times. Figure 2(c) demonstrates the expected penalty paid in terms of DML for conducting collaborative scheduling. It is clear that the DML for tasks with execution times less than $3D$ suffers because this is the minimal communication delay required to accept a thread's segments for execution.

5 Conclusions and Future Work

The preliminary investigation of consensus-driven collaborative scheduling approach described in this work may be extended in a variety of ways. In particular, algorithmic support for shared resources, deadlock detection and resolution, and quantitative assurances during overload represent worthwhile theoretical questions. Furthermore, improved implementations investigating real-world behavior under failures and with non-trivial abort handling are suggested by the results presented here. An exhaustive look at the practical message complexity would enable broad-based analysis of algorithm design and implementation trade-offs between time complexity and overload schedule quality.

References

1. CCRP: Network centric warfare, www.dodccrp.org/ncwPages/ncwPage.html
2. Northcutt, J.D., Clark, R.K.: The Alpha operating system: Programming model. Archons Project Tech. Report 88021, Dept. of Computer Science, Carnegie Mellon, Pittsburgh, PA (February 1988)
3. Ford, B., Lepreau, J.: Evolving Mach 3.0 to a migrating thread model. In: USENIX Technical Conference, pp. 97–114 (1994)
4. Open Group: MK7.3a Release Notes. Open Group Research Institute, Cambridge, Mass (October 1998)
5. OMG: Real-time CORBA 2.0: Dynamic scheduling. Technical report, Object Management Group (September 2001)
6. Anderson, J., Jensen, E.D.: The distributed real-time specification for Java: Status report. In: JTRES (2006)
7. Chetto, H., Chetto, M.: Some results of the earliest deadline scheduling algorithm. IEEE Transactions on Software Engineering 15(10), 466–473 (1989)

8. Jensen, E.D., et al.: A time-driven scheduling model for real-time systems. In: RTSS, pp. 112–122 (December 1985)
9. Clark, R.K.: Scheduling Dependent Real-Time Activities. PhD thesis, CMU (1990)
10. Locke, C.D.: Best-Effort Decision Making for Real-Time Scheduling. PhD thesis, CMU CMU-CS-86-134 (1986)
11. Ravindran, B., Anderson, J.S., Jensen, E.D.: On distributed real-time scheduling in networked embedded systems in the presence of crash failures. In: Proceedings of SEUS 2007 (May 2007)
12. Curley, E., Anderson, J.S., et al.: Recovering from distributable thread failures with assured timeliness in real-time distributed systems. In: SRDS, pp. 267–276 (2006)
13. Northcutt, J.D.: Mechanisms for Reliable Distributed Real-Time Operating Systems — The Alpha Kernel. Academic Press, London (1987)
14. Goldberg, J., Greenberg, I., et al.: Adaptive fault-resistant systems. Technical report, SRI Int'l (January 1995), http://www.csl.sri.com/papers/sri-csl-95-02/
15. Poledna, S., Burns, A., Wellings, A., Barrett, P.: Replica determinism and flexible scheduling in hard real-time dependable systems. IEEE ToC (2) (February 2000)
16. Gammar, S.M., Kamoun, F.: A comparison of scheduling algorithms for real time distributed transactional systems. In: Proc. of 6th IEEE CS Workshop on Future Trends of Distributed Computing Systems, pp. 257–261 (October 1997)
17. Aguilera, M.K., Lann, G.L., Toueg, S.: On the impact of fast failure detectors on real-time fault-tolerant systems. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 354–370. Springer, Heidelberg (2002)
18. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. JACM 43(2), 225–267 (1996)
19. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE ToC 51(5), 561–580 (2002)
20. Lynch, N.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
21. NetEm: Netem Wiki, http://linux-net.osdl.org/index.php/Netem
22. JSR-1 Expert Group: Real-time specification for Java, http://rtsj.org
23. Li, P., Ravindran, B., et al.: A formally verified application-level framework for real-time scheduling on POSIX real-time operating systems. IEEE Trans. Software Engineering 30(9), 613–629 (2004)
24. Mills, D.L.: Improved algorithms for synchronizing computer network clocks. IEEE/ACM TON 3, 245–254 (1995)

Novel Radio Resource Management Scheme with Low Complexity for Multiple Antenna Wireless Network System

Jian Xu, Rong Ran, DongKu Kim, and Jong-Soo Seo

Department of Electrical and Electronic Engineering, Yonsei University, Shinchon Dong, Seodaemoon-Ku, Seoul, 120-749, Korea
jianxu@yonsei.ac.kr

Abstract. Multiple-input multiple-output (MIMO) antennas can be combined with orthogonal frequency division multiplexing (OFDM) to increase system throughput by spatial multiplexing. That is the requirement of high speed future wireless networks such as WLANs and WMANs. This paper investigates the radio resource management problem of MIMO-OFDM based wireless network systems in order to maximize the total system throughput subject to the total power and proportional rate constraints of each user. A low-complexity algorithm that separates subcarrier allocation and power allocation is proposed. Simulation results show that the proposed resource allocation algorithm can improve the throughput and also it can make the throughput be distributed more fairly among users than some other schemes.

Keywords: WLANs, Radio Resource Management, Multiple Antenna, OFDM.

1 Introduction

Implementation of high-data-rate wireless local area network (WLAN) has been a major focus of research in recent years. Multiple-input multiple-output (MIMO) schemes [1] and [2] and orthogonal frequency division multiplexing (OFDM) [3] can be combined to operate at the high-throughput (HT) mode, or the diversity mode, or the combination of both in fading environments [4]. Such systems could achieve high spectral efficiency and/or a large coverage area that are critical for future-generation wireless local area networks.

Common open-loop linear detection schemes include the zero-forcing (ZF) and minimum mean-square error (MMSE) schemes [5] and [6]. A large condition number (i.e., the maximum-to-minimum-singular-value ratio, MMSVR) of the channel state information (CSI) matrix implies a high noise enhancement. Thus, MMSVR could be a convenient and effective metric to characterize the performance of different MIMO configurations. The importance and effectiveness of the eigenvalue distribution on MIMO system capacity and the overall system performance have been well recognized [7]–[10]. The eigenvalue analysis for MIMO-OFDM systems can be used to reduce the overall system complexity [11] and [12]. In this paper, the MMSVR is used for the subcarrier allocation, in this way the allocation complexity can be reduced and also the detection performance in the receiver side could be improved.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 261–270, 2007.

In recent years, however, many dynamic subcarrier, power allocation algorithms for single input single output (SISO) OFDM systems have been developed to find the solution of maximizing system throughput or minimizing the overall transmit power [13]-[15]. These suboptimal algorithms have good performances, but cannot be applied to the MIMO-OFDM system. Few researches [16]-[17] have been done for the dynamic subcarrier, power and bit allocation in the MIMO-OFDM system.

In this paper, we investigate the subcarrier and power allocation problems for MIMO-OFDM based wireless network system. We concentrate more on throughput fairness among the users. Our objective is to maximize the total throughput of the system subject to the total power and proportional rate constraints of each user. By dealing subcarrier and power allocation issues separately, we can simplify the resource allocation problem. We proposed a subcarrier allocation algorithm by dividing the users into groups and the MMSVR is treated as an important criterion to pick up the subcarriers for each user.

This paper is organized as follows. Section 2 introduces the MIMO-OFDM system model and presents the optimization objective function. In Section 3, the proposed radio resource management algorithm is described. Simulation results are illustrated in Section 4 and conclusions are drawn in Section 5.

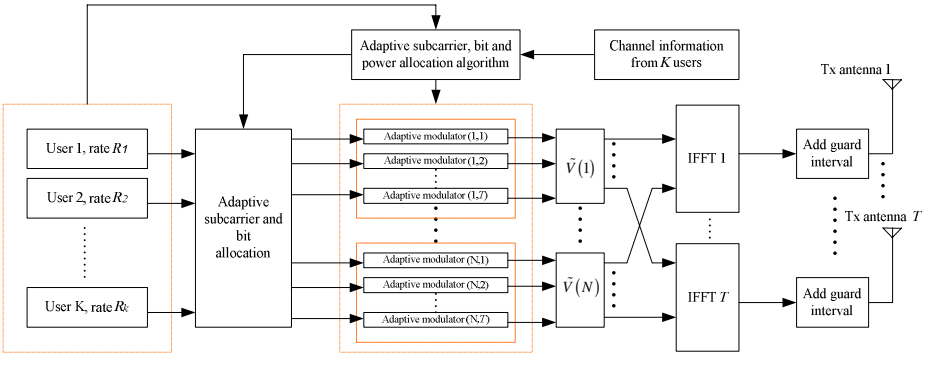


Fig. 1. Block diagram of MIMO-OFDMA system in the downlink

2 System Model and Problem Formulation

2.1 System Model

The block diagram of adaptive MIMO OFDMA system is shown in Fig.1. In this paper, it is assumed that in the base station the channel state information of each couple of transmit and receiver antennas is sent to the subcarrier and power algorithm block through the feedback channels. The resource allocation information is forwarded to the MIMO-OFDM transmitter. The transmitter then selects the allocated number of bits from different users to form OFDMA symbols and transmits via all the transmit antennas. The spatial multiplexing mode of MIMO is considered. The resource allocation scheme is updated as soon as the channel information is collected and also the subcarrier and bit allocation information is sent to each user for detecting.

2.2 Problem Formulation

Throughout this paper, let the number of transmit antennas be T and the number of receiver antennas be R for all users. Denote the number of users as K and the number of subcarriers as N. Assume that the base station has total transmit power constraint Q. The objective is to maximize the total system throughput and consider the fairness between the users with the total power constraint. We use the equally weighted sum throughput as the objective function. We also include a set of nonlinear constraints so that we can control the throughput ratios among users.

We formulate the following system throughput optimization problem to determine the subcarrier allocation and power distribution:

$$\max C = \frac{W}{N}\sum_{k=1}^{K}\sum_{n=1}^{N}\rho_{k,n}\left(\sum_{i=1}^{M_{kn}}\log\left(1+\frac{\lambda^{(i)}{}_{kn}q_{k,n}}{N_0}\right)\right) \tag{1}$$

subject to:

$$\sum_{k=1}^{K}\sum_{n=1}^{N}q_{k,n} \le Q.$$

$$q_{k,n} \ge 0 \; for\; all\; k,n$$

$$\rho_{k,n} = \{0,1\}\; for\; all\; k,n$$

$$\sum_{k=1}^{K}\rho_{k,n} = 1 for\; all\; n$$

$$R_1 : R_2 : ... : R_K = \gamma_1 : \gamma_2 : ... : \gamma_K$$

where K is the total number of users; N is the total number of subcarriers; Q is the total available power; W is the system bandwidth; $q_{k,n}$ is the power allocated for user k in the subcarrier n; M_{kn} is the rank of H_{kn} which denotes the channel gain matrix $(R\times T)$ on subcarrier n for user k and $\{\lambda_{kn}^{(i)}\}_{i=1:M_{kn}}$ are the eigenvalues of $H_{kn}H_{kn}^{\dagger}$; $\rho_{k,n}$ can only be the value of 1 or 0 indicating whether subcarrier n is used by user k or not. N_0 is the noise power in the frequency band of one subcarrier.

The throughput for user k, denoted as R_k, is defined as

$$R_k = \frac{W}{N}\sum_{n=1}^{N}\rho_{k,n}\left(\sum_{i=1}^{M_{kn}}\log\left(1+\frac{\lambda^{(i)}{}_{kn}q_{k,n}}{N_0}\right)\right) \tag{2}$$

and $\{\gamma_i\}_{i=1}^{K}$ is a set of predetermined values which are used to ensure proportional fairness among users.

3 Proposed Radio Resource Management Scheme

Ideally, subcarriers and power should be allocated jointly to achieve the optimal solution in (1). However, this poses a prohibitive computational burden at the access point or base station in order to reach the optimal allocation. Furthermore, the access point has to rapidly compute the optimal subcarrier and power allocation as the wireless channel changes. Hence, low-complexity suboptimal algorithms are preferred for cost-effective and delay-sensitive implementations. Separating the subcarrier and power allocation is a way to reduce the complexity, because the number of variables in the objective function is almost reduced by half.

Before we describe the proposed suboptimal resource allocation algorithm, we firstly show the mathematical expression of MMSVR. Let $\eta_{k,n} = \sigma_{k,n}^{\max} / \sigma_{k,n}^{\min}$ denotes the maximum to minimum singular value ratio for user k on subcarrier n. $\sigma_{k,n}^{\max}$ and $\sigma_{k,n}^{\min}$ are the maximum singular value and the minimum singular value of H_{kn}, respectively.

A large $\eta_{k,n}$ value could arise either because $\sigma_{k,n}^{\min}$ is small or because $\sigma_{k,n}^{\max}$ is large. From simulation results, it is found that the latter is unlikely, thus $\eta_{k,n}$ is a good indicator of noise enhancement, and if $\eta_{k,n} \gg 1$, we can conclude that the channel is ill-conditioned for the *n*th sub-carrier. In the following step, this point will be adopt as the important rule to allocate the subcarrier, which can avoid using the ill-conditioned subcarrier so that the noise enhancement could be reduced.

The steps of the proposed suboptimal algorithm are as follows:

Step 1. Assign the subcarriers to each user in a way that maximizes the overall throughput while maintaining rough proportionality;

Step 2. Assign the total power Q to allocated subcarriers using the multi-dimension waterfilling algorithm for the bad channel gain user group and the equal power allocation for the good channel gain user group.

3.1 Subcarrier Allocation by Avoiding Using Ill-Conditioned Subcarrier

This step allocates the per user assignment of subcarriers N_k, which is the number of subcarrier for user k and is determined by the average channel gain of each user, in a way that maximizes the overall throughput while maintaining rough proportionality. In this subcarrier allocation algorithm, equal power distribution is assumed across all subcarriers, and we define Ω_k as the set of subcarriers assigned to user k. The proposed algorithm is described below.

a.) Initialization

1) sort the users by average channel gains, suppose we get $\overline{H_1} \le \overline{H_2} \le ... \le \overline{H_m} \le ... \le \overline{H_K}$ without loss of generality
2) divide the users into two groups:

bad channel gain group: $user\_b = \{1, 2, ..., m\}$

good channel gain group: $user\_g = \{m+1, m+2, ..., K\}$

3) set $R_k = 0, \Omega_k = \varnothing$ for $k = 1, 2, ..., K$ and $A = \{1, 2, ..., N\}$

b.) For $k = 1$ to m

1) find n satisfying $\eta_{k,n} \le \eta_{k,j} for\ j \in A$, (MMSVR: $\eta_{k,n} = \sigma_{k,n}^{\max} / \sigma_{k,n}^{\min}$)

2) let $\Omega_k = \Omega_k \cup n,\ \mathrm{N}_k = \mathrm{N}_k - 1, A = A - \{n\}$ and update R_k according to (2)

c.) While $|A| > N - \sum_{i=1}^{m} N_i$

1) $user\_b = \{1, 2, ..., m\}$

find k satisfying $R_k / \gamma_k \le R_i / \gamma_i\ for\ all\ i, 1 \le i \le m$

2) for the found k, find n satisfying

find n satisfying $\eta_{k,n} \le \eta_{k,j} for\ j \in A$, (MMSVR: $\eta_{k,n} = \sigma_{k,n}^{\max} / \sigma_{k,n}^{\min}$)

3) for the found k and n,

if $\mathrm{N}_k > 0$

let $\Omega_k = \Omega_k \cup n,\ A = A - \{n\}$

$N_k = N_k - 1$ and update R_k according to (2)

else

$user\_b = user\_b - \{k\}$

d.) redo step b.) and c.) for the good channel gain group, i.e., for user index from $k = m+1$ to K

In the step a.) of the algorithm, the users are divided into the channel gain bad group $user\_b$ and the good group $user\_g$ according to the average channel gain. And then all the variables are initialized. R_k keeps tracks of the throughput for each user and A is the set of yet unallocated subcarriers.

The step b.) firstly assigns to each user of group $user\_b$ the unallocated subcarrier that has the minimum MMSVR for that user. Note that an inherent advantage is obtained by the bad channel gain group of users that are able to choose their best subcarrier earlier than the other group.

The step c.) proceeds to assign subcarriers to each user of group $user\_b$ according to the greedy policy that the user which needs a subcarrier most in each iteration gets to choose the best subcarrier that has minimum MMSVR for it. The need of a user is determined by the user who has the least throughput divided by its proportional constant. Once the user gets his assignment of N_k subcarriers, he can no longer be assigned with any more subcarriers. $|A|$ here denotes the cardinality of the set A.

The step d.) will assign the remaining subcarriers to the good channel gain group. We firstly change the condition "For $k=1$ to m " into "For $k=m+1$ to K " and redo the second step b.). And then we change the condition $|A| > N - \sum_{i=1}^{m} N_i$ into $A \neq \varnothing$ and change the selection range $(1 \le i \le m)$ into $(m+1 \le i \le K)$ so that we can redo the third step c.). Finally all of the subcarriers will be assigned.

By dividing the users into two groups, the users of group $user\_b$ get the inherent advantage to choose their best subcarriers firstly. Thus the power allocated to them in the next step will be deceased significantly compared with other algorithms without giving a priority to bad user group in terms of subcarrier allocation. Accordingly the power allocated to group $user\_g$ will be increased because of the fixed total power in the system. In this way the system throughput can be maximized for the tradeoff between the amount of allocated power and the number and quality of assigned subcarriers for users. Furthermore the proportional fairness will also be roughly guaranteed by step c.).

3.2 Power Allocation among Users and Subcarriers

The subcarrier algorithm in step 1 is for each user to use the subcarriers with low MMSVR as much as possible. However, this is not optimum because equal power distribution in all subcarriers is assumed. In this step, we propose a low complexity and efficient power allocation algorithm based on the user grouping criteria in step 1.

For the bad user group, the power is allocated among the assigned subcarriers for each user by using the multi-dimension water-filling method. In this way, the power efficiency can be improved a lot because of the gain of waterfilling algorithm when SNR is lower. The multi-dimension water-filling method is to find the optimal power allocation as follows.

The power distribution over subcarriers is

$$q_n^* = \max(0, q_n)$$

where q_n means the power for subcarrier n and it is the root of the following equation,

$$\sum_{i=1}^{M_{k_n n}} \frac{\lambda_{k_n n}^{(i)}}{\lambda_{k_n n}^{(i)} q_n + N_0} + \alpha = 0, \quad n = 1, 2, ..., N \tag{3}$$

where k_n is the allocated user index on subcarrier n; α is the water-filling level which satisfies $\sum_{n=1}^{N'} q_n^* = Q'$ where Q' and N' are the total power for bad user group and the number of subcarriers for bad user group, respectively.

For the good user group, equal power allocation among the assigned subcarriers is used because there is only a little bit difference between equal power allocation and power allocation using water-filling algorithm when the SNR is high. In this way, almost the same system throughput could be achieved and furthermore the complexity could be reduced significantly.

Once the power allocation for each subcarrier is decided, all of the transmitter antennas use the corresponding power for that subcarrier. Finally, the goal of maximizing the total throughput while maintaining relatively proportional fairness will be achieved after this proposed efficient and low complex power allocation scheme.

4 Simulation Results and Analysis

In this section, simulation results are presented to demonstrate the performance of the proposed algorithm. In the simulations, the wireless channel between a couple of transmit antenna and receiver antenna is modeled as a frequency selective channel consisting of six independent Rayleigh multipaths. Each multipath component is modeled by Clarke's flat fading model. The number of users is 4, and we assume that the average channel gain for user1 is 10 dB higher than user2, user3 and user4. The number of antennas is T=R=2 and each couple of transmit antenna and receiver antenna is independent to the other couples. The total transmit power is 1 W. The total bandwidth B is 1 MHz, which is divided into 64 subcarriers.

In Fig. 2, we show the system throughput of the proposed algorithm in a four-user MIMO-OFDM system vs. different fairness constraints, that is, $\gamma_1 = 2^m$, $\gamma_2 = \gamma_3 = \gamma_4 = 1$ and m is from $[0,1,2,3,4,5]$. Thus we have 6 sets of fairness constrains. Fig. 2 also shows the system throughput achieved by the method in [18] which is system throughput maximization scheme, and the system throughput achieved by a static FDMA system, in which each user is allocated with an equal number of subcarrier and it is not changing with the channel variation. The system throughputs in Fig.2 are system throughputs averaged over 5000 channel realizations. It can be seen that the two adaptive resource allocation schemes can achieve a significant throughput gain over the static FDMA.

We also notice that the system throughput maximization method in [18] achieves the maximum system throughput, because all the resources are allocated to the users with the best channel gains. The system throughput achieved by the proposed algorithm varies as the rate constraint changes. As more priority is allocated to user 1, i.e., as the index m increases, higher total system throughput is achieved. This is reasonable since user 1 has higher average channel gain and hence can more efficiently utilize the resources. And we can see that the throughput distribution of the method in [18] and static FDMA cannot be changed by varying the gamma set values, because there is no fairness guarantee mechanism in these systems.

Fig. 3 shows the normalized throughput distribution among users for gamma-set index m=3 where $\gamma_1 = 8$ and $\gamma_2 = \gamma_3 = \gamma_4 = 1$. The normalized throughput for user k is given by $R_k \Big/ \sum_{i=1}^{4} R_i$. The throughput distributions of four users MIMO OFDM downlink system are shown using the proposed resource allocation algorithm, the throughput maximization scheme in [18] and static FDMA scheme, respectively. It can be seen that the total throughput maximization method in [18] achieves the largest total throughput and user 1 gets most of the subcarriers and occupies most of the total throughput. The throughput for user 2, user 3 and user 4 is very little. Static FDMA tends to allocate similar throughput to each user, since all users get the same number of subcarriers to transmit. However, with the proposed resource allocation algorithm, the throughput is well distributed, very close to the defined ideal rate constraints, among users.

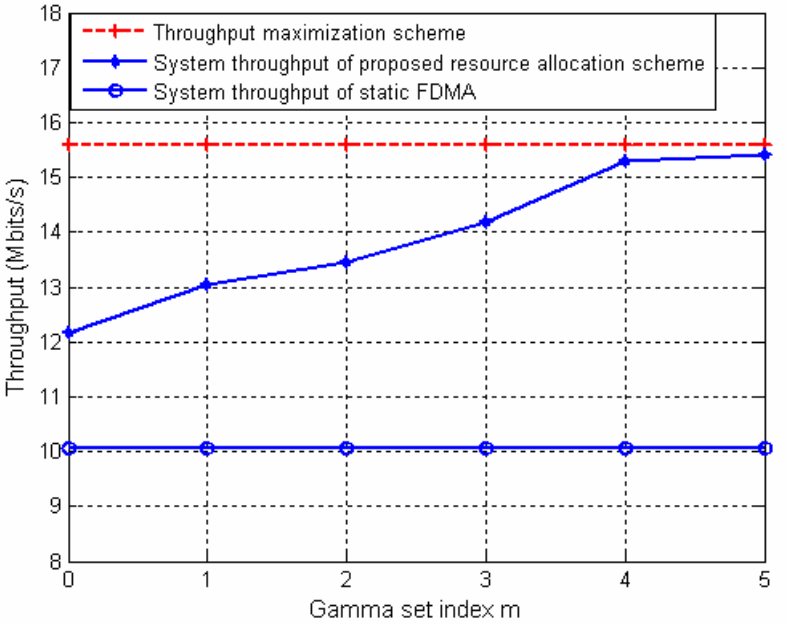


Fig. 2. System throughput of 4 users MIMO-OFDM systems vs. various gamma sets

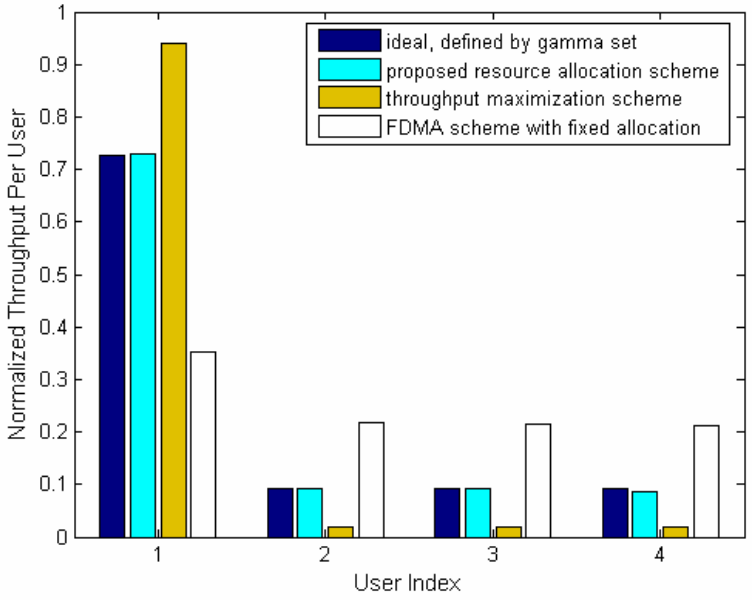


Fig. 3. Normalized throughput ratios distribution among users for the 4 users multiple antenna systems with $\gamma_1 = 8$ and $\gamma_2 = \gamma_3 = \gamma_4 = 1$

5 Conclusions

This paper presents a new method to solve the subcarrier and power allocation problem for the MIMO-OFDMA system. Allocations of subcarrier and power are carried out separately. For the subcarrier allocation, it assigns the subcarriers for each user by dividing the users into groups and also by avoiding using the large MMSVR subcarriers in the second step. The proposed power allocation scheme adopts different algorithms to different user groups in order to achieve high throughput with reduced complexity. Simulation results show that the proposed method can achieve significant throughput gain over the static FDMA. Using the proposed method the system throughput is distributed more fairly among users.

Acknowledgments. This work was supported by Samsung Electronics under the project on 4G wireless communication systems and Yonsei University Institute of TMS Information Technology, a Brain Korea 21 program, Korea.

References

1. Pualraj, A., Gore, D., Nabar, R., Bolcskei, H.: An overview of MIMO communications - a key to gigabit wireless. Proc. IEEE 92, 198–218 (2004)
2. Goldsmith, A., Jafar, S., Jindal, N., Vishwanath, S.: Capacity limits of MIMO channels. IEEE J. Select. Areas Commun. 21, 684–702 (2003)
3. Stuber, G., Barry, J., McLaughlin, S., Li, Y., Ingram, M., Pratt, T.: Broadband MIMO-OFDM wireless communications. Proc. IEEE 92, 271–294 (2004)
4. Van Zelst, A., Schenk, T.: Implementation of a MIMO OFDM-based wireless LAN system. IEEE Trans. Signal Proc. 52, 483–494 (2004)
5. Bjerke, B., Proakis, J.: Multiple-antenna diversity techniques for transmission over fading channels. Proc. IEEE Wireless communication and networking conference 3, 1038–1042 (1999)
6. Gesbert, D., Shafi, M., Shiu, D., Smith, P., Naguib, A.: From theory to practice: an overview of MIMO spacetime coded wireless systems. IEEE J. Select. Areas Commun. 21, 281–302 (2003)
7. Telatar, I.E.: Capacity of multi-antenna Gaussian channels. European Trans. Telecomm. Related Technol. 10, 585–595 (1999)
8. Chiani, M., Win, M.Z., Zanella, A.: The distribution of eigenvalues of a Wishart matrix with correlation and application to MIMO capacity. Proc. of IEEE Globecom conference 4, 1802–1805 (2003)
9. Martin, C., Ottersten, B.: Asymptotic eigenvalue distribution and capacity for MIMO channels under correlated fading. IEEE Trans. Wireless Commun. 3, 1350–1359 (2004)
10. Malik, R.K.: The pseudo-Wishart distribution and its application to MIMO systems. IEEE Trans. Inform. Theory 49, 2761–2769 (2003)
11. Huang, D., Letaief, K.B.: Pre-DFT processing using eigen-analysis for coded OFDM with multiple receive antennas. IEEE Trans. Commun. 52, 2019–2027 (2004)
12. Huang, D., Letaief, K.B.: Symbol based space diversity for coded OFDM systems. IEEE Trans. Wireless Commun. 3, 117–127 (2004)

13. Wong, C.Y., Cheng, R.S., Letaief, K.B., Murch, R.D.: Multiuser OFDM with adaptive subcarrier, bit, and power allocation. IEEE J. Select. Areas Commun. 17, 1747–1758 (1999)
14. Kivanc, D., Li, G., Liu, H.: Computationally efficient bandwidth allocation and power control for OFDMA. IEEE Trans. Wireless Commun. 2, 1150–1158 (2003)
15. Shen, Z., Andrews, J.C., Evans, B.L.: Adaptive resource allocation in multiuser OFDM systems with proportional rate constraints. IEEE Trans. Wireless Commun. 4, 2726–2737 (2005)
16. Pan, Y.H., Letaief, K.B., Cao, Z.G.: Dynamic spatial subchannel allocation with adaptive beamforming for MIMO/OFDM systems. IEEE Trans. Wireless Commun. 3, 2097–2107 (2004)
17. Rey, F., Lamarca, M., Vazquez, G.: Robust power allocation algorithm for MIMO OFDM system with imperfect CSI. IEEE Trans. Signal Proc. 53, 1070–1085 (2005)
18. Li G., Liu, H.: Capacity analysis on downlink MIMO OFDMA system. Submitted to IEEE Trans. Wireless Commun. (2004)
19. Li, G., Liu, H.: On the optimality of OFDM in multiuser multicarrier MIMO systems. Proc.IEEE Vehicular Technology Conference 4, 2107–2111 (2005)

Modelling Protocols for Multiagent Interaction by F-logic

Hong Feng Lai

Department of Business Management, National United University
1, LeinDa road, Miaoli city, 360, Taiwan
walden.lai@msa.hinet.net

Abstract. This paper proposes a method to represent agent Unified Modelling Language (AUML) in logic form using F-logic that provides deductive capability and uniform knowledge integration base. The AUML is used to differentiate relevant interaction more precisely at the analysis phase of developing a multiagent system. However, the AUML lacks for foundation and logic semantics. Thus we aim at constructing sufficient formality to facilitate formal analysis and to explore the behaviour and message route of the AUML. The AUML is transformed into F-logic language first by transformation rules. Secondly, a logic interpretation of this agent structure is presented. The transformation processes and results are illustrated using an example of E-commerce system. Finally, the significance of this approach is discussed and summarized.

Keywords: AUML, F-logic, message route, multiagent system, interaction protocols.

1 Introduction

As heterogeneous mobile devices continue to grow, how to integrate various types of information and knowledge is one of the most important issues in information technology. Since web technology has great potential to develop a collaborative environment, several strategies about information integration have been explored, e.g. centralized and open distributed strategies. However, centralized strategy has been shown not to be scalable. The open distributed strategy is growing and becoming difficult to manage. Beyond these methods, agent-based system with mobility is becoming a noticeable approach [1]. An agent is a computational process that implements the autonomous (actions without inputs), and communicating functionality of an application [2]. The agent-based system provides a convenient method to mobile users. Applying agent-based technologies, the services across web could be designed to be reactive, proactive, autonomous, and social.

In multiagent systems (MAS), agent interaction is ruled by interaction protocols. The agent Unified Modelling Language (AUML) is an extension of the UML, which proposes the standards for expressing the interactions of MAS. The interaction protocol (IP) diagram of AUML can help the designers to differentiate roles and messages between related agents more detailed. However, the IP diagrams do not guarantee the compliance of autonomous and heterogeneous agents to requirements [3].

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 271–282, 2007.

A logical specification describes system requirements formally. Through formal semantics, it supports deductive capabilities that make specifications executable [4]. Mathematical foundations have been studied in several previous papers [5-8]. For instance, first-order logic is exploited to establish a deductive foundation for entity relationship model [5]. In [6], applying the Larch based logic [9] to represent the logical semantics of OMT. In [7], a scheme for integrating object-oriented and logic programming paradigms is proposed. In [8], dynamic master logic diagrams are used to represent time-dependent behavior and knowledge of a dynamic system.

Since the AUML lacks for foundation and logic semantics [10], in this study we aim at constructing sufficient formality to allow formal analysis and to verify the properties of the AUML. To embed deductive capability in the AUML, we transform the AUML into a logical specification language, F-logic, which is proposed by Kifer et al. [11].

How to transform the AUML into formal specifications is investigated in this study. The transformation framework between the AUML and F-logic is displayed in Fig. 1. The deductive AUML consists of three components: message space (a set of messages in F-logic form), role space (a set of roles in F-logic form), and the deductive rules for determining the message routes between message and role objects.

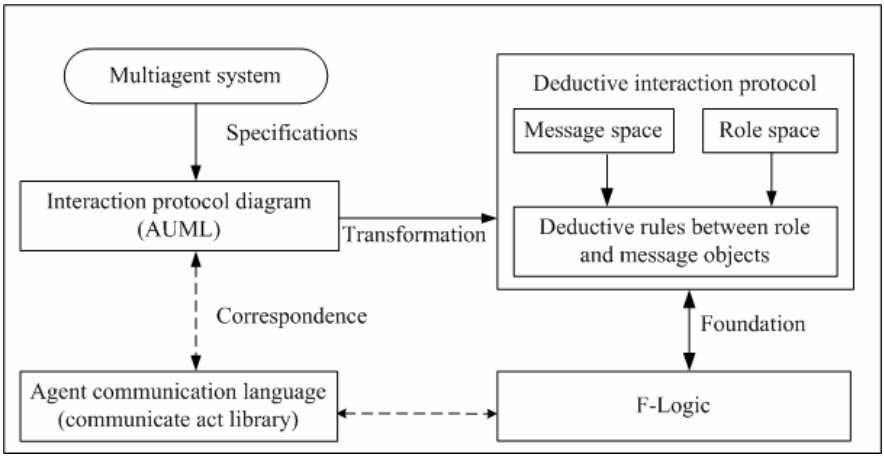


Fig. 1. The AUML/F-logic transformation framework

The structure of the paper is as follows. Section 2 gives an overview of F-logic. . Section 3 describes interaction protocol using AUML. The transformation rules and results between AUML and F-logic are presented in Section 4. Section 5 illustrates the related work. Section 6 concludes the paper.

2 F-Logic

In this section, we make a short summary of F-logic including its vocabulary and syntax. A comprehensive discussion of F-logic can be found in [11].

F-logic is a language with well-defined semantics that extends predicate logic and provides a sound and complete resolution-based proof procedure. This language is powerful in expressing object-oriented features. F-logic enhances the modeling capability of first order logic by syntactic enrichment, while it preserves its model-theoretic semantics by means of semantic structure. Elements are described by identification terms (id-terms), which consist of variables, functions, or constants, similar to terms in first order logic language.

The syntax of F-logic language, $\mathcal{L}$, consists of alphabetic symbols and the syntactic rules required to construct the well-formed formulas. The alphabetic symbols of F-logic language can be expressed in terms of its constituent parts:

a set of function symbols (object constructors), $\mathbb{C}$;
a set υ of variables;
a set $\wp$ of predicate symbols;
auxiliary symbols, such as: (,), [,], $\Rightarrow$, $\rightarrow$, =>>, $\twoheadrightarrow$, etc.;
and usual logical connectives and quantifiers, such as: $\wedge$ (and), $\vee$ (or), $\leftarrow$ (implication), $\neg$ (not), $\forall$ (universal), and $\exists$ (existential).

An id-term consists of constructor f (a member of $\mathbb{C}$) and object variable (a member of υ), similar to terms in first-order logic. For instance, f (X, g (a, Y)) is an id-term, where f and g are object constructors, 'a' is a constant, and X and Y are object variables. A ground F-term is a term not containing variables (variable-free). A ground F-term is denoted by a symbol that begins with a lower-case, while a symbol that begins with a capital letter denotes an id-term that may be non-ground.

An F-logic term (F-term) is defined as one of the following statements, which denote objects, classes, methods and predicates:

(1) An is_a assertion F-term A:b means that object A is a member of class b, or a::b denotes class a is a sub-class of class b. The is_a assertion enables attribute inheritance and subset relationship.

(2) A complex F-term is expressed as O[semicolon-separated list of method expressions], where O signifies an object or a class, and an method M expression can be either a scalar data expression: $M@A_1,\ldots,A_i \rightarrow R$, where $A_1,\ldots,A_i$, R is an id-term, or a set-valued data expression ($k \geq 0$): $M@A_1,\ldots,A_i \twoheadrightarrow <<S_1,\ldots,S_k>>$, and scalar signature expression $M@AT_1,\ldots,AT_i \Rightarrow (RT_1,\ldots,RT_k)$ or set-valued signature expression $M@AT_1,\ldots,AT_i=>>(RT_1,\ldots,RT_k)$, where, $A_1,\ldots, A_i$, $AT_1,\ldots,AT_i$ are arguments of method M, $R,S_1,\ldots,S_k$ denotes the output of the method M, $(RT_1,\ldots,RT_k)$ represents the types of the result of the method M. While a method does not need arguments, "@" will be omitted.

The implementation and application of F-logic can be found in several studies. FLORID [12] is a deductive engine for F-logic. In [13] FLORID with path expression is used to extract, restructure and manage the semi-structured web data. In [14, 15], they propose an operational knowledge specification language KARL, which contains two sublanguage Logic-KARL (F-logic) and Procedure-KARL. For specifying knowledge at conceptual and operational level, the domain layer and inference layer are expressed in Logical-KARL, while the task layer is represented by Procedure-KARL.

Thus, the KARL specification of intermediate representation can bridge the gab between an informal and an implementation of knowledge-based systems. In [16], a practical deductive object-oriented database system FLORA is provided, which integrates F-logic, Hilog and Transition logic, using complier optimization techniques to achieve its performance.

In this study we apply FLORID to express interaction protocols of multiagent system (MAS), i.e. to infer the roles and messages based on the deductive engine of FLORID.

3 The AUML

In this section, we make a brief introduction of AUML. A comprehensive discussion of AUML can be found in [10, 11].

3.1 Introduction to the AUML

The agent UML (AUML) is an extension of the UML, which proposes the standards for expressing the interactions of MAS. AUML applies graphical specification technique to describe interaction between agents. These approaches are partly based on the agent communication language (ACL) of the Foundation for Intelligent Physical Agents (FIPA) [2] using a subset of its communicative act library (CAL) of FIPA as messages.

3.2 Notations the Agent UML

An IP diagram indicates interactions between agents and roles along a timeline. Agents are assigned to roles. A role is a specification of the action that an object should fill. An object can switch roles at different times. Roles can be inserted or removed during the lifetime of agents.

Messages between agent roles are shown as arrows (Fig. 2) signifying an asynchronous communication. A diamond expresses a querying_if point that can result in zero or more communications. The line branch (no diamonds) indicates that all messages are sent concurrently. The empty diamond indicates that zero or more messages may be sent. A crossed diamond shows that exactly one message may be sent.

The message route (sequence of message) in Fig. 2 begins with a customer (initiator) which issues a request (cfp_P_order) to a manufacturer (participants). The manufacturer can reply proposing a price for satisfying the request (propose_proposal), or refusing (refuse_P_order). The customer must accept (accept_proposal) or reject (reject_proposal) the received proposals. After having received the cfp_P_order, the manufacturer must response to the customer by a given deadline, and informs the customer of propose or refuse_P_order. Analogously, the customer must response to the manufacturer by a given deadline. As the manufacturer receives the message of accept_proposal, the manufacturer must check “if stock was sufficient” (query_if_S_sufficient) whether the inventory level is true to satisfy the requirement of the customer.

The AUML diagrams are useful to analyze interactions between agents. Additionally, the agent UML can be taken as object interaction diagrams from the dynamic model viewpoint.

However, there are some limits in AUML [17]. These limits bring about extending or transforming the AUML to other model, e.g. cluttered AUML tends towards misinterpreting; unable to combine roles and cardinalities; indeterminable at decision points; hard to debug redundancy; unable to trace the history.

4 Transformation Rules and Deductive Rules

4.1 Introduction to the E-Commerce Example of a Multiagent System

In the E-commerce system**, t**he member agent customer and manufacturer in Fig. 2 invoke buyer agent and seller agent respectively. Analogously, the member agent vendor and manufacturer in Fig. 2 invoke seller agent and buyer agent respectively. The manufacturer switches roles at different times, i.e. facing customer as a seller and facing vendor as a buyer.

The message exchange in E-commerce can be modeled by mobile agent technology. A buyer agent could do purchasing for a customer, including making orders, negotiating, haggling, and potentially even paying.

A buyer agent can pass the customer's preferences to the host. If a potential match was met, the buyer agent could reply to the customer, or potentially finish the transaction delegated by the customer.

A seller agent must check (query_if_S_sufficient) whether the inventory level is true to satisfy the requirement of the customer, and negotiate price with buyer agent..

From implementation viewpoint, mobile agents are programs dispatching from one computer and transporting to a remote computer. As messages passing to a remote computer, the programs present their authorization and get access to local services and data. The remote computer may act as a broker by putting agents together with common interests and goals, and supporting a platform at which agents can coordinate.

4.2 Transformation Rules of the AUML/F-Logic

To create the message route of AUML IP diagrams involves a process of model transformation. Model transformation is a mapping from a source model to a target model using a set of transformation rules [4]. There is a natural correspondence between the AUML and F-logic. Based on the composite elements and notations in the AUML, the transformation rules from the agent UML into F-logic specifications are listed below.

The following two transformation rules express how to define AUML objects in F-logic form.

Trans_rule1. Each member_agent can be expressed as frame fields, and each role can be defined by F-logic as follows.

member_agent[has_role=>>role; name=>string; type=>string].
role[name=>string; type=>string; use=>string].

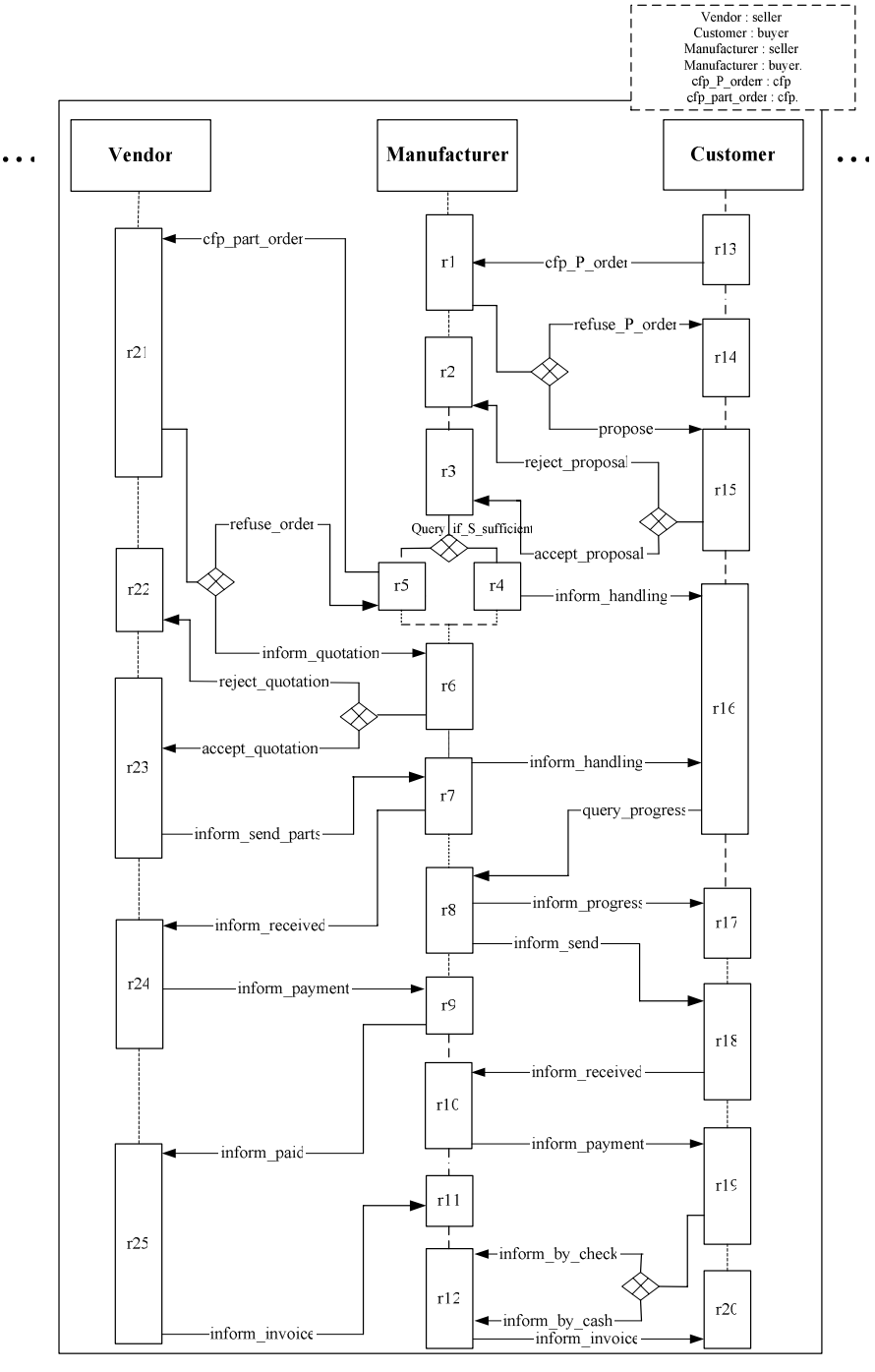


Fig. 2. The interaction protocol in E-commerce

Trans_rule2. Each message can be described via its sender, receiver, name, and type. The types of message includes: resouce_agent_role, delegation_agent_role, wrapping_agent_role, coordination_agent_role, and discovery_agent_role. The message and message_route can be defined by F-logic as follows:

message[sender=>role; receiver=>role; name=>string; type=>string].
message_route[sender=>role; receiver=>role;
add@(message)=>message_route;
add@(message_route)=>message_route].

The following two transformation rules express message and agent_role hierarchical relationships in F-logic form respectively.

Trans_rule3. For each message and its subclass, the relationship can be represented by 'is_a assertion'. This property can be denoted as follows:

refuse :: messgae.
reject :: messgae.
accept :: messgae.
inform :: messgae.
propose :: messgae.
query :: messgae.

Trans_rule4. For each agent_role and its subclass, the relationship can be represented by 'is_a assertion'. This property can be denoted as follows:

agent_role :: role.
resouce_agent_role :: agent_role.
delegation_agent_role :: agent_role.
wrapping_agent_role :: agent_role.
coordination_agent_role :: agent_role.
discovery_agent_role :: agent_role.

The resource_agent_role is used to manages local resources. The delegation_agent_role is used to invoke agent service. The wrapping_agent_role is used to transfer the coordination. The report_agent_role is used to support summarizing and reporting service. The discovery_agent_role is used to discover available external services.

Trans_rule5. For each vertical bar in AUML corresponds to a agent_role in F-logoic as follows:

agent.role : agent_roleType.

The following three transformation rules express how to transform the asynchronous message → of AUML in F-logic form.

Trans_rule6. For each message arrow of AUML can be expressed in F-logoic as follows.

message_name:message[sender→agent1.role_i; receiver→agent2.role_j; type→ message_type].

Trans_rule7. For each message with diamond arrow can be expressed in F-logoic as follows.

xor_message_name:message[sender→agent1.role_i; receiver→agent2.role_j; type→ message_type].

Trans_rule8. For each message with line branch (no diamonds) and empty diamond of AUML can be expressed in F-logoic as follows.
message_name:message[sender→agent1.role_i; receiver→agent2.role_j; type→ message_type].

4.3 The Deductive Rules of the AUML/F-Logic

Based on the transformation rules, the AUML of the E-commerce example (see Fig. 2) can be transformed into F-logic form. The deductive AUML will be expressed in terms of agent role space (a set of agent roles in F-logic), message space (a set of messages in F-logic), structural assertions, and deductive rules.

The deductive rules express the relation between roles, message, and message_route in AUML IP diagrams.

Deductive_rule1. These rules express how to add messages to message_routes as follows.

P.E:message_route[sender→X; receiver→Z] :-
P:message_route[sender→X; receiver→Y], E:message[sender→Y; receiver→Z].

Deductive_rule2. This rule expresses how to concatenate message_routes to a new message_route as follows.

P1[(P2.E)→P3] :- P1.P2[E→P3], E:message.

Deductive_rule3. These rules express how to detect loop routes and eliminate loop routes in a message_route as follows.

P:loop:- P:message_route[sender→P.receiver].
P.C = P :- P.(C:loop)[].

4.4 The Query of the Logic-Based E-Commerce System

After implementing the deductive AUML, the logic-based E-commerce system consist of a set of agent roles, a set of messages, a set of structural assertions, and some deductive rules about these elements. Various types of queries can be evaluated and answered by FLORID. For example, the query "?- M::message" state that "are there any sub-class of message".

% Answer to query : ?- M::messgae.
M/refuse
M/messgae
M/reject
M/accept
M/inform

M/propose
M/query
M/cfp

The query "?- R::agent_role" state that "are there any sub-class of agent_role". The answers are as follows:

% Answer to query : ?- R::agent_role.
R/agent_role
R/resouce_agent_role
R/delegation_agent_role
R/wrapping_agent_role
R/report_agent_role
R/discovery_agent_role

The message route is the sequence of messages passing to and fro on the IP diagrams. These behaviour properties can be also described by reachability tree in Petri net; or by message sequence chart in MSC [18]. However, a logic-based IP can provide more information, e.g. finding the message route between any two agent roles. For example, the query stated as "?- P:message_route[sender -> ven1.r21; receiver -> ven1.r25]" means that what the message route is between agent role ven1.r21 and agent role ven1.r25. The answers are as follows:

% Answer to query : ?- P:message_route[sender -> ven1.r21; receiver -> ven1.r25].
P/ven1.r21.xor_propose_quotation.(man1.r6.xor_accept_quotation).(ven1.r23.inform_sreceiver_parts).(man1.r7.inform_received).(ven1.r24.inform_payment).(man1.r9.inform_paid)
P/ven1.r21.xor_refuse_order.(man1.r5.cfp_part_order).(ven1.r21.xor_propose_quotation).(man1.r6.xor_accept_quotation).(ven1.r23.inform_sreceiver_parts).(man1.r7.inform_received).(ven1.r24.inform_payment).(man1.r9.inform_paid)
% 2 output(s) printed

5 Related Work

To express and coordinate the activities of multiagent systems, two types of approaches have been proposed: graphical and predicate approaches. The graphical approach using diagrammatic notation for intuitive understanding includes: agent UML approach [10], statechart approach [17], message sequence chart approach [18], Petri net approach [19, 20]. The textual approach using rules and declarations for consistency checking includes OMG IDL (Interface Description Language) [1] and logic-based approach [3, 17, 21].

Statecharts is a visual specification language for specifying discrete event system. It extends finite state machines that was proposed by [22] and could be described as: Statecharts = finite state machine + depth + orthogonality + broadcast communication. Statecharts has many good properties that can be applied in object-oriented information system. However, there are still some restrictions in MAS application domain. Many extensions had been proposed for improving their descriptive ability. In [23] they apply

propositional dynamic logic (PDL) to extend the description capability of statecharts for presenting interaction protocol.

The message sequence chart (MSC) diagrams are used to express basic protocols and scenarios in telecommunication systems [18]. An MSC diagram consists of a set of instances, which indicate that events may occur during the execution period. The types of events may be the creation and stopping of an instance, the trigger of a local service, the setting or resetting of a timer, the timeout, the sending or receiving of a message, etc. The main difference between MSC and MAS is that MSC using an axe represents a process of an instance, while MAS using isolated vertical bars signifies multi agent roles.

Petri net [24] is the most frequently used tool for modeling systems. Petri nets and Statecharts have equivalent representative capabilities because they are both state-based models [25]. Applying PN to AUML modelling, the message is taken as a place; the xor-message is expressed by a conflicting place; and the agent role is represented by a transition [20]. The limits of IP in PN include: hard to read, limits in model transformation, the problems of scalability and reusability [23].

The textual approaches using rules and declarations define the interaction of agents. From object-oriented viewpoint, the MAS could be taken as a set of interacting objects. To express the static and dynamical specifications of agents and roles in the agent-enhanced mobile virtual communities [1], they apply OMG IDL to differentiate agents and roles using interface specification sketch including require interfaces, provide interface, behaviour interface, and policy interface. This method resembles requirement decomposition while it lacks of deductive capability and can not check the consistency of specifications.

To verify the compliance of agents' behaviour to protocols, in [3] a logic-based formalism Social Integrity Constraints using Java-Prolog-CHR (Constraint Handling Rules) is proposed. An example of FIPA Contract-Net protocol is specified and verified by this approach. To guarantee the global properties of procedurally constructed MASs, in [21] a generalized linear temporal logic (GLTL) based agent system is constructed. The workflow properties (similar to the message route in this paer) are represented as temporal logic formulas and consequently can be verified by model checker.

The above related work illustrates the various types of approaches for presenting interaction protocol. In our approach, AUML/F-logic can be taken as a schema transformation that transforms conceptual level to operational level. Also, it plays a role of mediator for integrating objects in heterogeneous systems [26].

6 Conclusion

With the growing complexity of multiagent interaction in web-based applications, the requirement of tools and techniques for representing mobile agent is growing in the same way. This paper proposed a method to produce F-logic specifications for AUML that extends its expressive power, and provides syntax, semantics, and inference rules.

This approach was illustrated using an example of electronic commerce systems, which expressed how the various features of F-logic could be applied in AUML. These logical specifications provided more reasoning power. Additionally, the formal specification language is easily adapted to the system requirements.

The future work will explore the logic specifications of dynamical model of electronic commerce systems, i.e. in a working environment such that new rules can be inserted into the system. The transformation should reflect the corresponding logic specifications and support the transformation rules.

Acknowledgment. Financial support for this work was provided by the National Science Council Taiwan, under the contract NSC94-2416-H-239-003.

References

1. Loke, S.W., Rakotonirainy, A., Zaslavsky, A.: An enterprise viewpoint of wireless virtual communities and the associated uses of software agents. In: Rahman, S.M. (ed.) Internet Commerce and Software Agents: Cases, Technologies and Opportunities, pp. 265–287. Idea Group Publishing, Hersey, PA, USA (2001)
2. FIPA.: FIPA Agent Management Specification. Foundation for Intelligent Physical Agents(2002), //www.fipa.org
3. Alberti, M., Daolio, D., Torroni, P., Gavanelli, M., Lamma, E., Mello, P.: Specification and verification of agent interaction protocols in a logic-based system. In: Proceedings of the 2004 ACM symposium on Applied computing, pp. 72–78 (2004)
4. Mineau, G.W., Missaoui, R., Godinx, R.: Conceptual modeling for data and knowledge management. Data & Knowledge Engineering 33, 137–168 (2000)
5. Battista, G.D., Lenzerini, M.: Deductive entity relationship modeling. IEEE Transactions on Knowledge and Data Engineering 5, 439–450 (1993)
6. Bourdeau, R.H., Chen, B.H.C.: A formal semantics for object model diagrams. IEEE Transactions on Software Engineering 21, 799–821 (1995)
7. Lee, J.H.M., Pun, P.K.C.: Frame logic integration: A multi paradigm design methodology and a programming language. Computer Languages 23, 25–42 (1997)
8. Hu, Y.-S., Modarres, M.: Time-dependent system knowledge representation based on dynamic master logic diagrams. Control Engineering Practice 4, 89–98 (1996)
9. Guttag, J.V., Horning, J.J.: Larch: Languages and tools for formal specification. Springer, Heidelberg (1993)
10. Bauer, B., Muller, J.P., Odell, J.: Agent UML: A formalism for specifying multiagent interaction. In: Cuabcarubu, P., Wooldridge, M. (eds.) Agent-Oriented Software Engineering, pp. 91–103. Springer, Heidelberg (2001)
11. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the Association for Computing Machinery 42, 741–843 (1995)
12. FLORID Homepage (2006), http://dbis.informatik.uni-freiburg.de/
13. Ludäscher, B., Himmeröder, R., Lausen, G., W.M., Schlepphorst, C.: Managing semistructured data with florid: a deductive object-oriented perspective. Information systems 23, 589–613 (1998)
14. Fensel, D.: Graphical and formal knowledge specification with KARL. In: Proceedings of the the International Conference on Expert Systems for Development, pp. 198–203 (1994)
15. Fensel, D., Angele, J., Studer, R.: The Knowledge acquisition and representation language, KARL. IEEE Transactions on Knowledge and Data Engineering 10, 527–550 (1998)
16. Yang, G., Kifer, M.: FLORA: Implementing an efficient DOOD system using a tabling logic engine. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 1078–1093. Springer, Heidelberg (2000)

17. Paurobally, S., Chachkov, S., Jennings, N.R.: Developing agent interaction protocols using graphical and logical methodologies. In: Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, pp. 149–168. Springer, Heidelberg (2004)
18. Rudolph, E., Grabowski, J., Graubmann, P.: Tutorial on message sequence charts (MSC). In: Proceedings of the FORTE/PSTV 1996 Conference (1996)
19. Ling, S., Loke, S.W.: Advanced Petri Nets for modelling mobile agent enabled interorganizational workflows. In: Proceedings of the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 245–252 (2002)
20. Ling, S., Loke, S.W.: Engineering Multiagent Systems Based on Interaction Protocols: A Compositional Petri Net Approach. In: Camp, O. (ed.) Enterprise Information Systems V, pp. 279–285. Kluwer Academic, Netherlands (2004)
21. Pokorny, L.R., Ramakrishnan, C.R.: Modeling and verification of distributed autonomous agents using logic programming. In: Leite, J.A., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 148–165. Springer, Heidelberg (2005)
22. Harel, D.: Statecharts: a visual formalism for complex systems. Science Computer Program 8, 231–274 (1987)
23. Paurobally, S., Cunningham, R., Jennings, N.R.: Developing agent interaction protocols using graphical and logical methodologies. In: Workshop on Programming MAS, AAMAS (2003)
24. Peterson, J.L.: Petri-Net Theory and Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)
25. Bucci, G., Campanai, M., Nesi, P.: Tools for Specifying Real-Time Systems. Real-Time Systems 8, 117–172 (1995)
26. Wiederhold, G.: Mediators in the architecture of future information systems. Computer 25, 38–49 (1992)

Adding Adaptability to Mailbox-Based Mobile IP

Liang Zhang[1], Beihong Jin[1], and Jiannong Cao[2]

[1] Institute of Software, Chinese Academy of Sciences,
Hai Dian, Beijing, PRC
{zhangliang1216,jbh}@otcaix.iscas.ac.cn
[2] The Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong
csjcao@comp.polyu.edu.hk

Abstract. Mobile IP is one of the dominating protocols that provide mobility support in the Internet. However, even with some proposed optimization techniques, there is still space for improving the performance. In this paper, we present a mailbox-based scheme to further improve the performance. In this scheme, each mobile node migrating to a foreign network is associated with a mailbox. A sender sends packets to the receiver's mailbox, which will in turn forward them to the receiver. During each handoff, a mobile node can decide whether to move its mailbox and report the handoff to the home agent, or simply to report the handoff to the mailbox. In this way, both the workload on the home agent and the registration delay can be reduced. Also, the scheme is adaptive. By applying the dynamic programming to compute the optimal mailbox's migration policy, the scheme can make the total cost minimized.

Keywords: mobile computing, mailbox, mobile IP, dynamic programming.

1 Introduction

The growth of wireless communication technologies and the advancement of laptop and notebook computers induce a tremendous demand for mobile and nomadic computing. Researchers have investigated Internet Protocol (IP) for mobile internetworking, leading to the development of a proposed standard for IP mobility support called Mobile IP [1].

However, Mobile IP suffers from the well-known triangle routing problem where packages have to first take the detour to the home network before being forwarded to the foreign network where the mobile node is currently residing. To deal with this problem, Mobile IP route optimization [1] is proposed. Any node that communicates with a mobile node maintains a binding cache. When the home agent intercepts a packet for the mobile node outside the home network, it will send a binding update message to the sender, informing it of the mobile node's current care-of address. The sender then updates its binding cache and tunnels any ensuing packets for the mobile node directly to its care-of address. An extension to the registration process called smooth handoff [1] enables a foreign agent to also make use of binding update to reduce the packet loss during a handoff. The mobile node may request the new

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 283–293, 2007.

foreign agent to send to the previous foreign agent a binding update message called "Previous Foreign Agent Notification", which will enable the previous foreign agent to re-tunnel any packet for the mobile node to the new care-of address. Although Mobile IP and route optimization provide general mechanisms for mobility support in the Internet, there are still several performance problems that need to be addressed [6].

In this paper, we present a mailbox-based scheme to alleviate the performance problems stated above. Each mobile node migrating to a foreign network is associated with a *mailbox*. A sender sends packets to the receiver's mailbox, which will in turn forward them to the receiver. During each handoff, a mobile node can decide whether to move its mailbox and report the handoff to the home agent, or simply to report the handoff to the mailbox. In this way, both the workload on the home agent and the registration delay can be reduced. Since the mailbox is located somewhere in the network closer to the receiver than the sender, the retransmission cost for the lost packets can also be reduced. By separating the mailbox from its owner, we can achieve adaptive location management that enables the dynamic tradeoff between the packet delivery cost and the location registration cost to minimize the total cost.

The rest of the paper is organized as follows. Section 2 presents our mailbox-based scheme. Section 3 proposes an adaptive algorithm to optimize the performance of our scheme. Section 4 evaluates the performance. The final section concludes the paper.

2 A Mailbox-Based Scheme

In this paper, both home agents and foreign agents are referred to as *mobility agents*. Each mobile node is associated with a mailbox, which is a data structure residing at a mobility agent. As shown in Fig. 1, if a sender wants to send a packet to a mobile node, it will simply send the packet to the receiver's mailbox. Later, the receiver receives the packet from its mailbox.

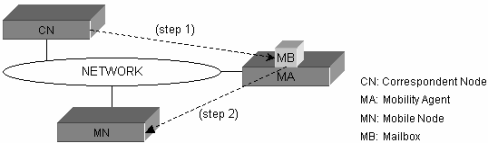


Fig. 1. Mailbox-based scheme

Initially, the mailbox is residing in the same network as its owner. The mobile node realizes that it has entered a new foreign network whenever it receives an "Agent Advertisement" message [1] from a new foreign agent. Immediately, it sends a registration message to the old foreign agent where its mailbox resides. The old foreign agent then decides whether or not to move the mailbox to the new foreign agent with the consideration of two primary factors: the distance to the new foreign agent and the communication traffic of the mobile node. If the mobile node is expected to receive many packets while the distance is long, it will be costly to forward all these packets to the new address and better to move the mailbox closer to the mobile node so as to

achieve a more optimal route. On the other hand, if the mobile node seldom receives packets or the distance is quite short, it is economical to leave the mailbox at where it is to reduce the registration overhead. Therefore, how to decide the mailbox's migration pattern adaptively according to the two factors can affect greatly the performance of our scheme. We would like to postpone the discussion about this question in the next section. As a summary, we differentiate two types of handoff in our scheme, i.e., handoff without mailbox and handoff with mailbox, and we name them *local handoff* and *home handoff*, respectively.

Besides mailbox, another new data structure called *address table* is defined in each mobility agent. Each entry in an address table has six attributes: 1) the home address of the mobile node, 2) the mailbox's address, 3) a valid tag, 4) a pointer to the mailbox, 5) the care-of address of the mobile node, and 6) a time-to-live (TTL) timer. The valid tag is used to indicate whether the mailbox is under migration or not, which also implies whether the mailbox's address is outdated or not. The TTL timer is used to timeout trash entries in the address table. The scheme also defines operations for two processes, *Migrating* and *Packet-forwarding*.

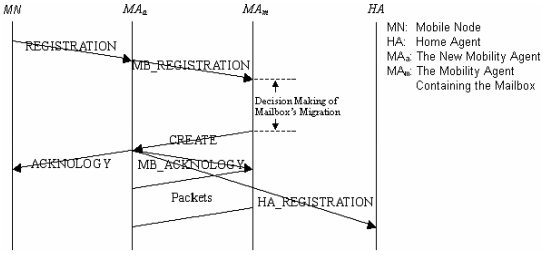


Fig. 2. Migrating

Upon receiving the advertisement from a new mobility agent MA_a, the mobile node *MN* determines that it has roamed to a new foreign network. It then initiates the registration process as shown in Fig. 2. It first uses gratuitous ARP [2] to update the ARP caches of the nodes in the foreign network so that they will associate *MN*'s link layer address with its home address. After that, it sends a "REGISTRATION" message to MA_a, within which the key information contained is the address of the mobility agent MA_m where the mailbox *MB* is currently residing.

Upon receiving the "REGISTRATION" message, MA_a extracts the address of MA_m from the message, and sends a "MB_REGISTRATION" message to MA_m.

Upon receiving the "MB_REGISTRATION" message, MA_m makes a decision whether or not to move *MB* to MA_a. In case that *MB* does not migrate, MA_m simply updates the care-of address of *MN* to MA_a. Otherwise, it will:

- set the valid tag of the corresponding entry in the address table to *false*, and
- send a "CREATE" message to MA_a, requesting for a new mailbox *MB'* for *MN*.

Upon receiving the "CREATE" message, MA_a creates *MB'* and adds an entry to its address table to record this newly created mailbox. It also sends three messages:

- an "ACKNOLOGY" message to *MN*, informing it of the new address of *MB'*,
- an "MB_ACKNOLOGY" message to MA_m, telling it the creation of *MB'*, and

- an "HA_REGISTRATION" message to the home agent *HA*, registering the new address of the mailbox.

  After receiving the "MB_ACKNOLOGY" message, MA_m will:
- update the address of the mailbox in the address table to that of *MB'*,
- set the valid tag to *true*,
- activate the TTL timer,
- stream every packet buffered in *MB* to *MB'*,
- inform the new address of the mailbox to the senders of the buffered packets in *MB* by sending "UPDATE" messages, and
- after all the buffered packets have been streamed out, deconstruct the mailbox and setting *null* to these two attributes in the address table – the pointer to the mailbox and the care-of address of the mobile node.

After receiving the "HA_REGISTRATION" message, *HA* updates the address of the mailbox in its address table.

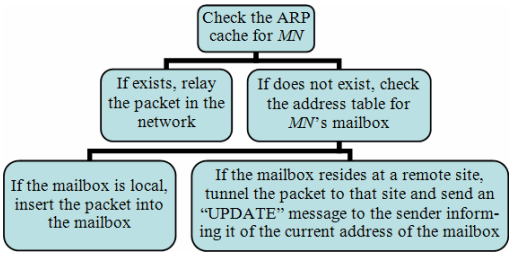


Fig. 3. Packet-forwarding

If a correspondent node *CN* wants to send a packet to *MN*, it will first check its binding cache to see whether the address of *MN*'s mailbox has been cached locally or not. If so, it will tunnel the packet to the cached address. Otherwise, it will send the packet with regular IP routing to *MN*'s home address. Once the packet arrives in the home network, *HA* will intercept the packet as it acts as a proxy ARP server for *MN*.

When a mobility agent receives a packet destined to *MN*, it will perform actions according to Fig. 3. For a packet in *MB* to be forwarded to *MN*, MA_m first checks the valid tag. If it is *false*, i.e., the mailbox is migrating to a new foreign agent, MA_m will suspend the packet forwarding. It will rely on the migrating process to stream the packet to the new location of the mailbox later. Otherwise, the valid tag is *true* and MA_m will directly tunnel the packet to the care-of address of *MN*.

3 An Adaptive Algorithm

In this section, we will first present the system model for a mobile network and the walk model for a mobile node, which are adopted in many existing studies such as [3] and [4]. The model assumes that the coverage area of a mobile network is partitioned into cells. A cell is defined as the coverage area of a mobility agent that can exchange packets with mobile nodes directly. One mobility agent serves only one cell and cells

do not overlap with each other. A movement occurs when a mobile node moves from the residing cell to one of its neighboring cells. The distance between any two cells in the network is measured by the minimum number of cell boundary crossings for a mobile node to travel from one cell to another. If we assume that a mobility agent is a router in a cell that can communicate directly through wired lines with other mobility agents in the neighboring cells, the distance between two mobility agents can be defined as the distance between their cells.

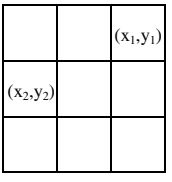


Fig. 4. System model

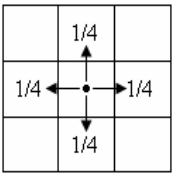


Fig. 5. Walk model

We consider a grid configuration for a mobile network, which is composed of equal-sized, rectangular and non-overlapping cells. With this configuration as shown in Fig. 4, each cell has four neighbors and the distance between any two cells with the coordinates (x_1, y_1) and (x_2, y_2) is $|x_2-x_1|+|y_2-y_1|$. We also consider a commonly used walk model – the *random walk model* for a mobile node. In this model, a mobile node moves to one of its four neighbors with equal probability of 1/4 as shown in Fig. 5.

Based on the model, we will develop an adaptive algorithm that can dynamically adjust the mailbox's migration pattern according to the two factors in order to optimize the performance. The performance metric is the total communication cost over the experimental period which includes both the location registration cost and the packet forwarding cost. The communication cost is defined as the multiplication of the number of messages sent, the message size and the traveling distance.

Table 1. Definition of parameters

Parameter	Definition
$f_m(t)$	the (negative exponential) probability distribution function of the packet's inter-arrival time
λ	the mean packet arrival rate, i.e., $f_m(t) = \lambda e^{-\lambda t}$
$f_r(t)$	the (negative exponential) probability distribution function of the mobile node's residence time at a cell
μ	the mean residence time at a cell, i.e., $f_r(t) = \mu e^{-\mu t}$
η	the expected number of packets to be received at a cell, which is known as the packet-to-mobility ratio λ/μ; **it is actually the second primary factor that may affect the mailbox's migration**
d_{Cs}	the proportionality constant between the signaling transmission cost and the transmission distance
d_{Cp}	the proportionality constant between the packet delivery cost and the transmission distance
d_{Ts}	the proportionality constant between the signaling transmission time and the transmission distance
w	the proportionality constant between the transmission cost (the transmission time) of the wireless link and that of the wired link
M	the number of correspondent nodes

In order to optimize the performance, we choose to use the dynamic programming since it can help to make a sequence of inter-related choices to optimize the system performance. Readers are referred to [5] for the detailed description and procedure

about the dynamic programming. Before applying the dynamic programming, let us first precisely define our performance optimization problem. The experimental period starts with the mobile node and its mailbox collocating at an initial foreign agent, and ends after the mobile node has performed *N* migrations. The total communication cost over this period can be expressed as follows.

$$\mathrm{Cost}_{\mathrm{total}} = \sum_{i=1}^{N} \left(\mathrm{Cost}_{\mathrm{signaling}}(\mathrm{i}) + \mathrm{Cost}_{\mathrm{packet}}(\mathrm{i})\right) \tag{1}$$

where $Cost_{signaling}(i)$ and $Cost_{packet}(i)$ means the location registration cost during the mobile node's *i*th migration and the packet forwarding cost during the mobile node's residing at the new cell after its *i*th migration, respectively. Our performance optimization problem is to minimize (1).

Fig. 6 depicts the network scenario about the mobile node's *i*th migration in our scheme. The *Current FA* is the foreign agent after the mobile node's *i*th migration from the *Previous FA*. Therefore, the two foreign agents must be adjacent to each other, i.e., $d_5 = d \pm 1$ where ***d* is actually the first primary factor that may affect the mailbox's migration** and the $\pm$ sign depends on whether the mobile node migrates away from or close to the mailbox. In case of home handoff, the *Current FA* will become the new residing place for the mailbox. Here d_2, d_4 and d_7 are the average distance to all the correspondent nodes.

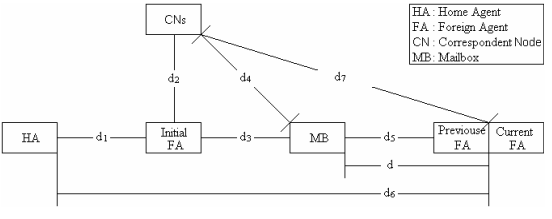


Fig. 6. Network scenario in our scheme

Depending on whether the *i*th migration is a home handoff or a local handoff, $Cost_{signaling}(i)$ can be expressed as follows. Readers are referred to Fig. 2 for details.

$$\mathrm{Cost}_{\mathrm{signaling}}(\mathrm{i}) = \begin{cases} (2\mathrm{w} + 3\mathrm{d} + \mathrm{d}_6)\mathrm{d}_{\mathrm{Cs}} + \mathrm{Cost}_{\mathrm{update}}(\mathrm{i}), & \text{a home handoff} \\ (\mathrm{w} + \mathrm{d})\mathrm{d}_{\mathrm{Cs}}, & \text{a local handoff} \end{cases} \tag{2}$$

where $Cost_{update}(i)$ means the signaling cost for the binding updates to the correspondent nodes during the mobile node's *i*th migration and can be expressed as follows.

$$\mathrm{Cost}_{\mathrm{update}}(\mathrm{i}) = \mathrm{M} \times \mathrm{d}_4 \times \mathrm{d}_{\mathrm{Cs}} \tag{3}$$

Now come to discuss the expression of $Cost_{packet}(i)$. Normally, the packet forwarding cost, depending on the handoff type, can be expressed as follows.

$$\mathrm{Cost}_{\mathrm{normal}}(\mathrm{i}) = \begin{cases} (\mathrm{d}_7 + \mathrm{w})\mathrm{d}_{\mathrm{C_p}}, & \text{a home handoff} \\ (\mathrm{d}_4 + \mathrm{d} + \mathrm{w})\mathrm{d}_{\mathrm{C_p}}, & \text{a local handoff} \end{cases} \tag{4}$$

However, as packet loss may occur during the mobile node's migration, we have to also consider the packet retransmission cost for those lost packets. In our scheme, the lost packets are retransmitted from the nearby mailbox instead of the possibly far away senders. Therefore, the cost for successfully delivering any lost packet to the mobile node can be expressed as follows. Readers are referred to Fig. 6 for details.

$$\mathrm{Cost}_{\mathrm{retransmission}}(\mathrm{i}) = (\mathrm{d}_4 + \mathrm{d}_5 + \mathrm{w} + \mathrm{d} + \mathrm{w})\mathrm{d}_{\mathrm{Cp}} \tag{5}$$

During the period when the mobile node resides at the new cell after its *i*th migration, it is expected to receive η packets, within which a portion are retransmitted packets since they were delivered based on the outdated location information. As we know, the mailbox learns the new address of the mobile node only when it receives the "MB_REGISTRATION" message from the new foreign agent MA_a. Therefore, the period of the outdated location information can be expressed as follows.

$$\mathrm{T}_{\mathrm{retransmission}}(\mathrm{i}) = (\mathrm{w} + \mathrm{d})\mathrm{d}_{\mathrm{Ts}} \tag{6}$$

As the mean packet arrival rate is λ, the mean lost packets is $\lambda \times T_{retransmission}(i)$. The mean packets that do not require retransmission are therefore $\eta - \lambda \times T_{retransmission}(i)$. $Cost_{packet}(i)$ can now be expressed as follows.

$$\begin{aligned}\mathrm{Cost}_{\mathrm{packet}}(\mathrm{i}) = {} & (\eta - \lambda \times \mathrm{T}_{\mathrm{retransmission}}(\mathrm{i})) \times \mathrm{Cost}_{\mathrm{normal}}(\mathrm{i}) + \\ & \lambda \times \mathrm{T}_{\mathrm{retransmission}}(\mathrm{i}) \times \mathrm{Cost}_{\mathrm{retransmission}}(\mathrm{i})\end{aligned} \tag{7}$$

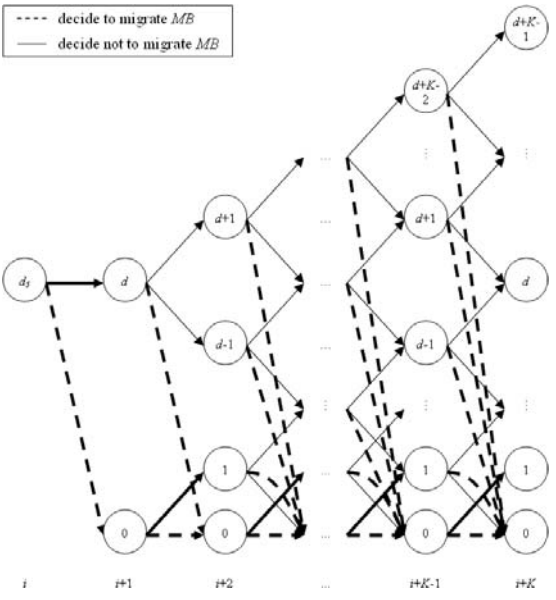


Fig. 7. Model for our cost optimization problem

Now let us start to formulate our cost optimization problem with the dynamic programming. Fig. 7 graphically depicts the model where the dynamic programming is utilized to optimize the communication cost for the next *K* migrations of the mobile

node. The circle represents the state, which means the distance between the mobile node and its mailbox before the current migration; and the arrow represents the action, which can be either a home handoff or a local handoff. Initially, the distance between the mobile node and its mailbox is d_5 according to Fig. 6. The mobile node now starts its ith migration. If this migration results in a home handoff, the system will go to state 0 since the mobile node and its mailbox are about to collocate. Otherwise, the system goes to state d according to Fig. 6. Similarly, during the mobile node's $(i+1)$th migration, if the system state is 0, either the system will remain at state 0 should a home handoff takes place, or the system will go to state 1 should a local handoff takes place; if the system state is d, not mentioning the home handoff, the system's next state will be either $d+1$ or $d-1$ depending on whether the mobile node moves away from or close to its mailbox. We use a thick line to represent an action with 100% certainty and a thin line to represent an action with possibility. With all the necessary items for the dynamic programming defined, the backward induction algorithm can be applied to derive the optimal decision policy for the mailbox's migration. Readers are referred to [5] for details.

4 Performance Evaluation

In this section, we will evaluate the performance of our scheme with the adaptive algorithm. First, let us conduct the performance modeling for the benchmark scheme – Mobile IP route optimization with the smooth handoff extension. Fig. 8 depicts the network scenario about the mobile node's migration in this scheme. Here d_8 is the average distance from the *Previous FA* to all the correspondent nodes.

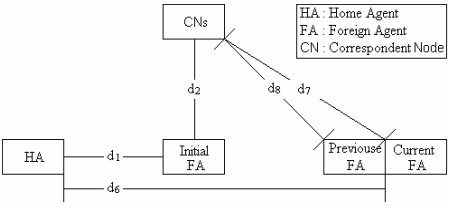


Fig. 8. Network scenario in the benchmark scheme

During each mobile node's migration, different types of signaling messages will be issued to the following entities: the home agent, the *Previous FA* and all the correspondent nodes. Therefore, $Cost_{signaling}(i)$ can be expressed as follows.

$$\mathrm{Cost}_{\mathrm{signaling}}(\mathrm{i}) = (\mathrm{w} + \mathrm{d}_6 + 1)\mathrm{d}_{\mathrm{Cs}} + \mathrm{Cost}_{\mathrm{update}}(\mathrm{i}) \tag{8}$$

$$\mathrm{Cost}_{\mathrm{update}}(\mathrm{i}) = \mathrm{M} \times \mathrm{d}_8 \times \mathrm{d}_{\mathrm{Cs}} \tag{9}$$

To derive $Cost_{packet}(i)$, we have to separately deal with the normal packet delivery and the packet retransmission. Below list the expressions of the cost and the time period of the normal packet delivery, and those of the packet retransmission. The

packet retransmission period is after the mobile node's migration to the new cell but before the "Previous Foreign Agent Notification" message arrives at the *Previous FA*.

$$\mathrm{Cost}_{\mathrm{normal}}(\mathrm{i}) = (\mathrm{d}_7 + \mathrm{w})\mathrm{d}_{\mathrm{Cp}} \tag{10}$$

$$\mathrm{T}_{\mathrm{normal}}(\mathrm{i}) = 1/\mu - \mathrm{T}_{\mathrm{retransmission}}(\mathrm{i}) \tag{11}$$

$$\mathrm{Cost}_{\mathrm{retransmission}}(\mathrm{i}) = (\mathrm{d}_8 + \mathrm{w} + \mathrm{d}_7 + \mathrm{w})\mathrm{d}_{\mathrm{Cp}} \tag{12}$$

$$\mathrm{T}_{\mathrm{retransmission}}(\mathrm{i}) = (\mathrm{w}+1)\mathrm{d}_{\mathrm{Ts}} \tag{13}$$

Therefore, $Cost_{packet}(i)$ can be expressed as follows.

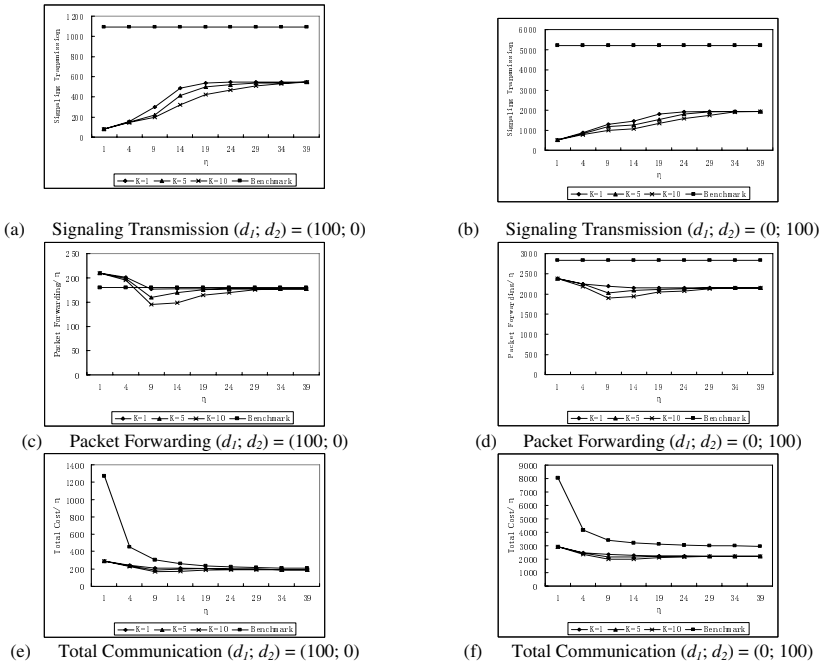
$$\begin{aligned}\mathrm{Cost}_{\mathrm{packet}}(\mathrm{i}) = \; & \lambda \times \mathrm{T}_{\mathrm{normal}}(\mathrm{i}) \times \mathrm{Cost}_{\mathrm{normal}}(\mathrm{i}) + \\ & \lambda \times \mathrm{T}_{\mathrm{retransmission}}(\mathrm{i}) \times \mathrm{Cost}_{\mathrm{retransmission}}(\mathrm{i})\end{aligned} \tag{14}$$

As mentioned, our experiments start with the mobile node and its mailbox collocating at an initial foreign agent, where the distance to the home agent and the average distance to the corresponding nodes are d_1 and d_2, respectively. We selectively choose two (d_1; d_2) pairs: (100; 0) and (0; 100), where the first pair visualizes the scenario when the mobile node is far away from the home agent but close to its correspondent nodes, and the second pair shows the scenario in a reverse condition.

The experimental period lasts for 36 mobile node's migrations; the signaling transmission cost per hop d_{Cs} is set to a normalized value 1; the packet delivery cost per hop d_{Cp} is twice as high as d_{Cs}; d_{Ts}, which can be understood as the signaling processing time on a router, is set to 0.05 sec; we assume the transmission cost (the transmission delay) in the wireless environment is twice as high (long) as that in the wired environment; and there are five active correspondent nodes.

Other parameters that affect the performance metric are d_4, d_6, d_7, d_8, η and K. As the mobile node can move in any direction, the average value for d_6 should be the same as d_1 and the average values for d_4, d_7 and d_8 should be the same as d_2. For simplicity, we just replace d_6 with d_1, and d_4, d_7 and d_8 with d_2 in the equations. All these distance values may be obtained from the routing table if it uses link state routing protocols such as OSPF, and the packet-to-mobility ratio η is easy to obtain since the mailbox acts as a relay and buffer station of the mobile node. We also select three Ks: 1, 5 and 10, where K=1 only considers the immediate benefit and K=10 takes more future influence into the consideration.

Fig. 9 lists the experimental results. For all six diagrams, we use η as the x-axis, and exam separately the signaling transmission cost, the packet forwarding cost and the total communication cost under different experimental settings. We observe that the signaling transmission cost increases as η rises. This is because as more packets are received during each migration, it is more likely that the mailbox also moves, i.e., a home handoff occurs. For a home handoff, more signaling messages for the location registration are needed than a local handoff. However, the packet forwarding cost drops since after the mailbox's migration, packets are routed in a direct path (sender → receiver (mailbox)), instead of the triangle routing (sender → mailbox → receiver), between the sender and the receiver.



(a) Signaling Transmission (d_1; d_2) = (100; 0)
(b) Signaling Transmission (d_1; d_2) = (0; 100)
(c) Packet Forwarding (d_1; d_2) = (100; 0)
(d) Packet Forwarding (d_1; d_2) = (0; 100)
(e) Total Communication (d_1; d_2) = (100; 0)
(f) Total Communication (d_1; d_2) = (0; 100)

Fig. 9. Experimental result

We are pleased to discover that both the signaling transmission cost and the packet forwarding cost of our scheme are smaller than those of the smooth handoff scheme. For the signaling transmission cost, it is easy to understand the advantage of our scheme since our scheme normally does not require the home agent be notified of every mobile node's migration while the smooth handoff scheme does. For the packet forwarding cost, although the smooth handoff scheme may spend less for the normal packet transmission, it would require significant cost for retransmitting lost packets, especially when the distance to the senders d_2 is long. This is because in our scheme, the retransmission starts from the nearby mailbox while in the smooth handoff scheme, the retransmission starts from the possibly distant senders. Finally, the total cost of our scheme is smaller than that of the smooth handoff scheme.

Besides, a smaller K normally performs worse than a bigger K, which implies that the dynamic programming configured with a smaller K does not produce the optimal mailbox's migration policy. This is because a smaller K only considers the myopic benefit without thinking about the future effect. Generally speaking, the bigger K is, the more future effect is considered, the more optimal migration policy of the mailbox can be derived, and the less total communication cost is required.

5 Conclusion

In this paper, we propose a mailbox-based scheme with an adaptive algorithm to improve the performance of Mobile IP in the following aspects: reduced workload on

the home agent, fast handoff, reduced packet loss, high throughput, reduced retransmission cost and delay, per-user-based adaptive location management, and dynamic tradeoff between the packet delivery cost and the location registration cost. The experiments conducted show a very sound result that demonstrates the benefits of using the mailbox, especially when the home agent is far away from the current location of the mobile node.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 60673123 and the National Hi-Tech Research and Development 863 Program of China under Grant No. 2006AA01Z231.

References

1. Perkins, C.: IP Mobility Support for IPv4, RFC 3220 (January 2002)
2. Postel, J.: Multi-LAN Address Resolution, RFC 925 (October 1984)
3. Wang, Y., Chen, W., Ho, J.: Performance Analysis of Mobile IP Extended with Routing Agents. In: Proceedings of European IASTED International Conference on Parallel and Distributed Systems (July 1998)
4. Akyildiz, F., Ho, J., Lin, Y.: Movement-Based Location Update and Selective Paging for PCS Networks. The IEEE/ACM Transactions on Networking 4(4), 629–638 (1996)
5. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
6. Zhang, L., Cao, J., Das, S.K.: A Mailbox-based Scheme for Improving Mobile IP Performance. In: Proceedings of Mobile and Wireless Networks (May 2003)

Palpability Support Demonstrated

Jeppe Brønsted[1], Erik Grönvall[2], and David Fors[3]

[1] Dept. of Computer Science, University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark
jb@daimi.au.dk
[2] Communication Science Department, University of Siena
Via Roma 56, 53100 Siena, Italy
[3] Dept. of Computer Science, Lund University
Ole Römers väg 3, 223 63 Lund, Sweden

Abstract. In ubiquitous computing, as more and more devices are embedded into the environment, there is a risk that the user loses the understanding of the system. In normal use this is not always a problem, but when breakdowns occur it is crucial that the user understands the system to be able to handle the situation. The concept of palpable computing, introduced by the PalCom project, denotes systems which support such understandability. In PalCom, a set of prototype scenarios provide input for an open software architecture and a conceptual framework for palpable computing. One of these prototype scenarios is based on the Active Surfaces concept in which therapists rehabilitate physically and mentally impaired children by means of an activity that stimulates the children both physically and cognitively.

In this paper we demonstrate how palpability can be supported in a prototype of the Active Surfaces. Services on the tiles have been developed using the PalCom service framework that allows them to be combined into PalCom assemblies. The support for palpability is shown by examples of use scenarios from the work of the therapist who can inspect and alter the runtime state of the tiles to change their configuration and cope with breakdown situations. The prototype implementation runs on a standard PC simulating the network layer and a first reference implementation has been made on the target embedded platform.

Keywords: Palpable computing, ubiquitous computing, middleware, services, assemblies, inspection, simulation framework.

1 Introduction

Palpable computing is a new perspective on ubiquitous computing, in which traditional ubiquitous computing challenges such as invisibility and composition are complemented with visibility and deconstruction. The concept has been formed based on the observation that when applied in real settings, ubiquitous computing systems tend to become hard to understand for users. Mark Weiser painted a vision of systems being 'physically invisible' and that these systems

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 294–308, 2007.

also mentally (as maybe physically) could disappear through use. Dr. Weiser describes the concept well as follows; "Whenever people learn something sufficiently well they cease to be aware of it." and "The most profound technologies are those that disappear" [1]. This implies not only that as we learn to master a technology, we move the use (or perception) of it from a foreground to a background cue [2], but also that a technology that has the capacity to allow the user to interact with or through it as a background process is a more thoughtful, intense or reflective technology.

As part of the ubiquitous conceptual framework and closely related to the work regarding foreground and background cues is the notion of 'calm technology' [3]. Calm technology as described by Weiser and Brown regards how technology can move from the centre of our attention out to the periphery, and between those two states as required by the situation at hand. The vision of calm technology is that technology should not overload us with information or require an ongoing 'active' mental activity. Weiser and Brown argues that this can be reached in two ways; 1) Allowing the technology (or information) to move between the centre to the periphery of our attention (and between these two states) and 2) by enhancing our peripheral reach. This is done by allowing more data to enter the periphery cues. As described in their paper, a video conference could be an example of technology that enhance the peripheral reach in respect to an ordinary telephone call where the users cannot use facial and body expressions as part of their communication [3].

In many ways, ubiquitous systems tries to embed the notion of 'ready at hand', meaning that these highly distributed, networked systems and devices should adapt to the current needs of the user or users. In normal use the system should be invisible and not interfere with the present task. However, when a breakdown occurs the user should be able to inspect the system to determine the reason and, if possible, resolve the situation. If the system is composed of multiple devices, it should be possible to replace malfunctioning devices with new ones without having to recompile or restart the system. We do *not* claim that techniques such as self-reconfiguration, error detection and fault tolerance should not be used. We make the observation that such mechanisms will never be perfect and that we therefore, as a supplement, need a way to handle the situations where the mechanisms are imperfect.

Palpable computing is researched in the EU IST project PalCom [4]. The main output of the project is a conceptual framework for palpable computing, an open architecture supporting palpable computing, and a collection of tools to be used in development of palpable computing applications. A part of the work in the project deals with developing palpable computing prototypes using participatory design to provide input to the conceptual framework and the design of the open architecture.

The *Active Surfaces* [5] concept provides support for physical-functional and cognitive rehabilitation in a swimming pool setting. The concept has been developed using participatory design techniques in corporation with therapists and patients. Through analysis of the rehabilitation practice an activity

(i.e. a number of different games) has been developed in which children assemble floating tiles into meaningful configurations. Each of the tiles is a resource constrained embedded system that communicates using only a low bandwidth short-range infrared link. The only output available to the user is a set of light emitting diodes and therefore the game is an example of a ubiquitous computing system where it is essential that the physical and functional characteristics are such that palpability can emerge during use.

For the software on the tiles, support for palpability is achieved by adhering to the PalCom open architecture [6], and by building on the PalCom service framework [7]. Services developed using the framework can be combined into PalCom assemblies, which coordinate the services and provide support for inspection, deconstruction and reconstruction. Through interaction with the assemblies, the therapist can inspect and change the configurations of the tiles. This way, she can adapt the therapeutic activity in the middle of an exercise, and the visibility given by the assemblies helps her cope with unexpected breakdown situations.

The rest of the paper is structured as follows. In the next section we describe the Active Surfaces concept and the physical and functional aspects of the prototype. Section 3 presents the PalCom software architecture and demonstrates how it can be used to support palpability in the implementation of the prototype. In Section 4 scenarios from therapist work are presented, together with an evaluation of how the prototype supports palpable qualities. Section 5 sums up conclusions and presents future work.

2 Active Surfaces

Active Surfaces is a concept developed for rehabilitation practitioners being a support for physical-functional and cognitive rehabilitation treatments in a swimming pool setting. Therapists working in cognitive and physical rehabilitation with disabled patients usually experience their job as challenging and demanding. Every time the therapist starts a treatment she has to define a specific program and ad hoc solutions with the aim of designing a rehabilitation intervention that could adapt to the individual patients' needs. Thus, the work of the therapist is mainly characterised by creativity both in designing engaging activities and suitable tools.

The lack of integration of physical and cognitive rehabilitation represents a constraint for current rehabilitation practice. The cognitive tasks are usually too static and children may lose attention. On the other hand, motor rehabilitation is very demanding at a physical level and is based on repetitive sequences of actions: patients often perceive them as tiring and not engaging. Here the Active Surfaces allow an integration of these two therapeutic goals with the activity. Water as such is an interesting context from the activity perspective; Water creates a safe context where impaired people can move autonomously relying on the added support to the body, something they cannot do elsewhere. Apart from this, water also poses specific and interesting research issues both for the development of digital technologies and for the therapeutic practice.

The work has been driven following a co-evolutionary method [8,9]. The approach integrates participatory design with creative concept design, using different typologies of scenarios for converging ideas into solutions. The concept and project has been developed together with children affected by different impairments undergoing therapeutic activities in the swimming pool, their parents, trainers and therapists at the swimming pool and at the 'Le Scotte' hospital in Siena, Italy. The early phases of the fieldwork have been devoted to understand the activity, to define requirements, and to collect best practices. On this basis, the concept of the Active Surfaces has been developed, capitalising on participatory design activities and creative workshops together with Travelling Architect [10] and Future Lab [11] sessions.

2.1 The Prototype

The prototype consists of a set of floating tiles (figure 1) that can be connected to each other to form a network. The tiles support multiple games by having a simple composable physical appearance and multipurpose programmable hardware. On each of the tiles' four sides magnets are placed to make the tiles "snap" together when they are in close vicinity. On the top of the tile is a replaceable plastic cover also held in place by magnets. The image on the cover depends on the game. On each side of the tiles light emitting diodes (LEDs) provide visual feedback to the user.

Fig. 1. Tiles

Inside each tile an embedded system uses infrared light to communicate with and detect the presence of other tiles. Two tiles can only communicate if they are close to each other. Figure 2 shows an overview of the hardware components in the tiles. The main computational unit is the UNC20 module, which is an ARM7-based embedded system running uClinux [12] at 55MHz with approximately 8MB ram. The UNC20 module communicates with a sideboard using a serial connection. The sideboard is responsible for controlling the infrared communication and the LEDs. The bandwidth of the infrared communication is approximately 600 bits per second.

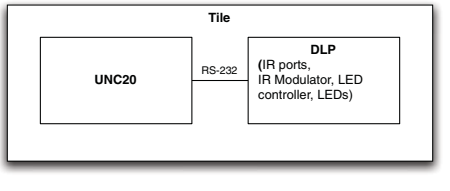


Fig. 2. Hardware

2.2 Games

The tiles support multiple games and in the following we describe a few suggestions. To change the current game the therapist connects the tile to a PDA

running PalCom software. Since the PDA is not suited for a wet environment this should be done prior to the training activity.

A lot of games can be imagined for the Active Surfaces. However, for a game to be appropriate for the tiles it should support both physical and cognitive rehabilitation while at the same time be implementable on the resource-constrained devices. Furthermore, to be able to help a wide range of patients the set of games should be of varying difficulty, both on the physical and on the cognitive level. Finally, the games should be open ended and configurable so that they can be adapted and customised to each rehabilitation session.

In this section we describe three games with different properties with respect to physical and cognitive rehabilitation. The first game, *catch*, is meant to only require simple cognitive effort but challenges the patients reflexes, speed, and coordination. The second game, *scrabble*, has the requirement that the patient should be able to form words out of letters. The last game, *puzzle*, is a traditional puzzle game in which an image is created by assembling the tiles in a specific pattern.

Catch. In the catch game the therapist aligns a set of tiles and gives another tile to the patient (at the bottom in figure 3). When the game is started the point of the game is for the patient to try to catch the light by approaching her tile to the glowing tile within a certain timeframe. If she succeeds another random tile will light up (not hers) and she tries to catch that one. When she eventually fails to catch the light before the time limit her tile will blink how many lights she caught. The game can be adapted to the patient by configuring the length of the timeframe.

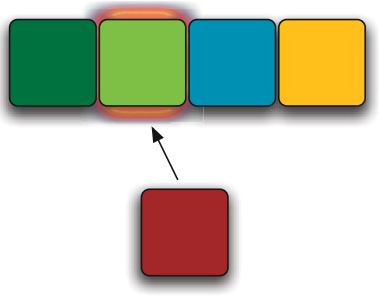


Fig. 3. Catch

Scrabble. In the scrabble game each tile has a letter on the surface. The patient uses the tiles to create words. When a tile is part of a word it lights up on all four sides. Each tile should be aware of what letter it has on the face. The memory requirement for the game depends on the number of tiles and on which letters the tiles have. As an example at least 24 English words can be generated from letters on the tiles in figure 4[1]. Since this number grows exponentially with the number of tiles it is not feasible to store all possible combinations on each tile. Instead, only the valid words for a *particular* tile-letter configuration should be uploaded to the tiles. The letter configuration should therefore be uploaded along with the game before the training activity.

[1] a, ad, at, act, arc, art, cad, car, cat, had, hat, rat, tad, tar, arch, card, cart, char, chat, dart, hard, hart, chard, and chart

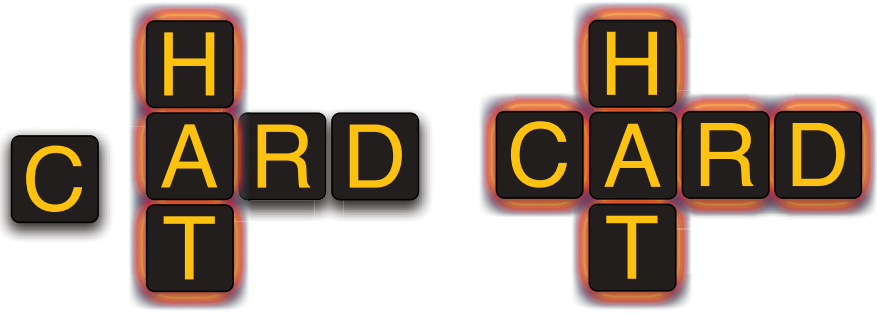


Fig. 4. Scrabble

Puzzle. In the puzzle game the face of each tile is part of a larger image (see figure 5). Initially the tiles are spread in a random pattern after which the patient starts to solve the puzzle. As the game progresses the patient gets continuous feedback from the LEDs. When two tiles are connected correctly the corresponding sides light up (fig. 5a). When all of a tile's neighbours are correct all sides of that tile light up (fig. 5b), and finally when the puzzle is solved the outline of the solution lights up (fig. 5c).

(a) **(b)** **(c)**

Fig. 5. Puzzle

During the session the therapist can change the faces of the tiles to make a new puzzle. To reprogram the tiles a special *assembler tile* is used. The assembler tile has the same physical appearance as the other tiles, but also has a button. To make the tiles remember the new solution they are arranged in the solution pattern and the assembler tile is put next to one of the tiles and the button is pressed. After this, the tiles will remember the new solution and can be scattered randomly again. This way of programming the tiles by showing them the correct solution has some similarities with *programming by example* [13] and *physical programming* [14]. The LED feedback can be configured by the therapist to alter the difficulty level of the game. It is, e.g., easier to solve the puzzle if all the outer edges of the final solution will light up as the game is started.

The different game types described above all have different game rules. These rules defines the base of a game. Apart from them, different behaviour can be configured to support the game rules and the activity. This can for example be different output to the end-user to aid in accomplishing the task, i.e. the game. This configuration can be physical and logical.

3 Implementation

In this section we demonstrate how the PalCom software architecture and runtime system can be used to implement the Active Surfaces prototype in a way that supports palpability. We describe the PalCom runtime system (section 3.1) and a simulation framework that can be used to experiment with the tiles on a standard PC (section 3.2). The top layer of the runtime system is the application layer in which applications are built by composing services (section 3.3) into assemblies (section 3.4). Finally, in section 3.5, the implementation of the puzzle game is described.

3.1 PalCom Runtime System

The PalCom runtime system (see figure 6) consists of four layers: the execution platform, the runtime environment, the middleware management layer, and the application layer. The execution platform consists of hardware and optionally an operating system. Presently multiple hardware platforms are supported including the UNC20 [15] and standard PCs. The runtime environment consists of standard libraries and a runtime engine which can be Sun's Java VM or the Pal-VM [16], which is a compact virtual machine specially designed for embedded systems for ubiquitous computing. If the hardware platform is the UNC20 only the Pal-VM is supported. The middleware management layer consists of managers handling resources, services, assemblies, and contingencies. For further description of the middleware managers we refer to [6]. At the time of writing, the memory footprint of the middleware management layer is too big to fit into the memory of the UNC20 (app. 8MB). Therefore, concurrently with the development of the hardware for the tiles and the optimisation of the middleware management layer, the software for the tiles has been developed to

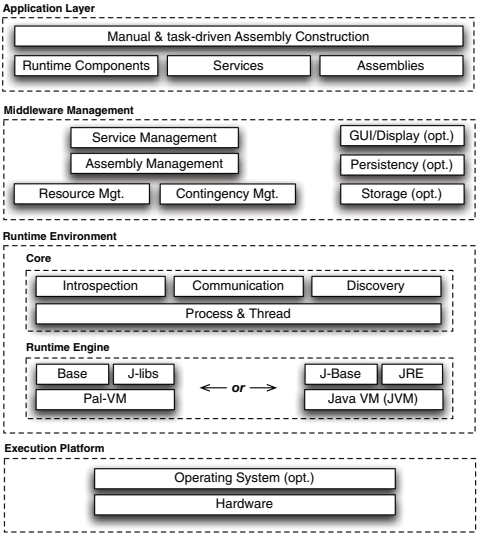


Fig. 6. PalCom layered runtime system

run on a standard PC with simulated infrared communication, on top of Sun's Java VM. When the middleware management layer fits into the memory of the UNC20, the implementation of the prototype should be able to run unaltered on the UNC20.

3.2 Simulation Framework

To ease the development of game logic and software for the tiles a simulation framework (figure 7) has been developed. Having a simulator available makes it possible to develop software and hardware in parallel and high level tools that are not available for the embedded platform can be used for debugging and profiling. Furthermore, testing involving repeated rearrangement of the tiles is much easier done using a mouse in a graphical user interface than physically moving the actual tiles around.

The simulator consists of a model of the swimming pool as a medium for infrared communication and a graphical user interface for manipulating the physical location of the tiles. The user interface is connected with the pool model so that when a tile is moved in the user interface, the pool model is updated accordingly.

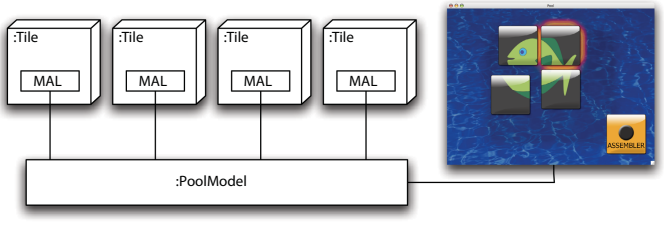


Fig. 7. Simulation framework

The model of the pool is also used in the medium abstraction layer of each of the tiles. When a tile sends a message, the medium abstraction layer of the tile accesses the pool model to determine which tiles the tile is connected to (if any) and delivers the message accordingly. From an application developer's perspective it is transparent whether the simulation framework or the physical hardware is used. The only part of the middleware that interacts with the simulation framework is the media abstraction layer and therefore system behaviour experienced on the simulator is likely to be similar on the embedded platform.

3.3 Services

The software implementing the functionality of the tiles is divided into services using the PalCom service framework [7]. As described in [6] a PalCom service is an entity that 1) contains a number of runtime components, 2) is discoverable, and 3) can be invoked remotely. The interaction with the service is done using

an explicitly defined service interface. A PalCom service has a set of commands, in-going or out-going, that optionally can be grouped. An in-going command specifies a point of entry to the service that is similar to a traditional interface except that invocation of the command is done in an asynchronous manner and that no results are returned. Outgoing commands specify what in-going commands the service invokes. Both in-going and outgoing commands have types specified by MIME [17] strings. We adopt the UML lollipop interface notation for commands - provided interface ("closed lollipop") for in-going commands and required interface ("open lollipop") for outgoing commands.

PalCom services can be *bound* or *unbound*. Bound services are tied to the hardware device on which they run, and typically expose functionality in the hardware for remote use. Unbound services, on the other hand, are not tied to the hardware and can thus be moved to and installed on new devices. We use a UML stereotype `<<unbound>>` for unbound services.

3.4 Assemblies

Services are connected by means of *assemblies*. The assembly is PalCom's mechanism for service coordination, central in the project's approach to support for construction and deconstruction of palpable systems. A system that is constructed using assemblies can be inspected in a service browser [18], making its inner structure visible at a certain level. This gives better understanding of the system, and is particularly important when a system breaks down. Furthermore, the assembly concept targets construction of systems from services that were not originally created for cooperation with each other. By inspecting the interfaces of a set of services, it is possible to construct an assembly that combines them. Service composition has previously been used to implement ubiquitous computing applications [19].

Assemblies are defined by assembly scripts that can be loaded at runtime by interacting with an assembly manager (see figure 6). When an assembly is loaded the assembly manager makes the appropriate connections and governs the flow of service invocations. We use the nesting mechanism in UML for assemblies.

One goal of the implementation of the tiles has been to make it possible to replace the game logic easily without rebooting the devices. This is done using assemblies. A set of basic services encapsulates the basic hardware functionality of the tiles and each game is implemented as one or more unbound services that can be connected to these services. The basic services of the tiles are a `LED` service controlling the LEDs, a `Connectivity` service detecting the presence of neighbour tiles, and a `Touch` service receiving input from the button if one is present (as is the case for the assembler tile in the puzzle game). The combination of the assembly and the unbound services for the game logic can be replaced when switching to another game. At present only the puzzle game has been implemented.

The split in functionality between an assembly and an unbound service is normal for a PalCom system. The assembly captures the coordination logic between services, while the services perform most of the calculations. For adding

behaviour to a set of services without programming a new service it is possible to express some calculation in assembly scripts, but the assembly script language is intended to be much simpler than the general-purpose programming languages normally used for implementing services. Therefore, complex calculations are implemented in unbound services that are incorporated when constructing an assembly. Finding the right level of sophistication in the assembly script language, and how much should be delegated to unbound services is a challenge that is under active research in the project.

3.5 The Puzzle Game

As described in [20] each of the tiles can be in one of three states: *sideHappy*, *localHappy*, or *globalHappy*. The states correspond to the types of feedback given by the tiles in the game. In figure 5, the top-right tile is sideHappy in figure 5a, localHappy in figure 5b, and globalHappy in figure 5c. Three rules determine which state a tile is in:

1. A tile is sideHappy if it has less than four correct sides.[2]
2. It is localHappy if it has four correct sides but at least one of the other tiles are sideHappy. This means that the tile has found its place in the puzzle, but the complete puzzle is not solved.
3. If no tile is sideHappy then all tiles are globalHappy. The puzzle is solved.

As can be seen from these simple rules, the game has a notion of global state, namely, whether there is at least one sideHappy node. This information is used by the tiles to distinguish whether the tile is localHappy or globalHappy. If a tile has less than four correct sides it does not need this information (because of rule 1).

The global state is maintained by handling two situations: The first situation occurs when a tile observes that it has four correct sides instead of three. It then broadcasts (by using the publish-subscribe mechanism of the communication model) a challenge to the other nodes requiring any sideHappy nodes to reply immediately (also with a broadcast). If no responses are received within a certain timeframe it is concluded that there are no sideHappy nodes and that the node therefore instead of being sideHappy should be globalHappy. If a response is received the node should be localHappy. When a localHappy node receives a challenge it treats it as if it was originating from itself - the node sets a timer and waits for responses. It is assumed that the solution of the puzzle is connected and includes all nodes and therefore it cannot be the case that no nodes are sideHappy in a proper subset of all the nodes. Therefore, if there is a sideHappy node there is a path from that node to the node that initiated the challenge.

The second situation, inverse to the first one, occurs when a node observes that it has three correct sides instead of four. The new state of the node is now sideHappy. If the node was globalHappy before the other nodes are unaware that

[2] We define a correct side of a tile to be a side that has a correct neighbour or has no neighbour and should have no neighbour according to the solution.

the node is now sideHappy, and therefore a message is broadcasted specifying so. If the node was localHappy before, it can assume that there is at least one sideHappy node in the graph it is connected to. The above paragraphs describe how the global state is maintained. Alternately, this could be done using the two-phase commit (2PC) protocol. The 2PC protocol uses, however, a lot more communication and since the inter-node communication bandwidth is approximately 600 bits/second the protocol has been deemed inappropriate.

Figure 8 shows an UML deployment diagram outlining the structure of the implementation. In the puzzle game there are two types of tiles - the normal tiles and the assembler tile. The normal tiles communicate with each other and with the assembler tile using IR communication. In the normal tiles the `PuzzleAsm` assembly (listed in figure 9) connects the basic services to the unbound services handling the game logic. The `Puzzle` service receives connectivity events from the `Connectivity` service (line 18–20 in figure 9) and determines the local state of the tile. This information is sent to the `LED` service (line 21–26) and to the `Coord` service (line 27–32) that coordinates the global state using the algorithm specified above. If all tiles are correctly aligned the `Coord` service notifies the `LED` service (line 33–35). The assembler tile contains a `Configure` service with the responsibility of initiating and configuring the game and the `PuzzleConfAsm` assembly to connect it to the basic services.

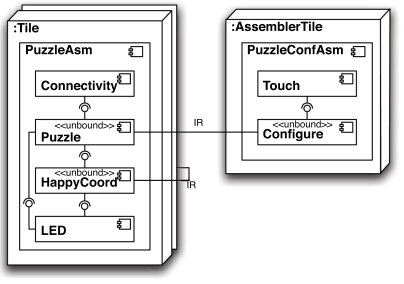


Fig. 8. Services in the puzzle game

4 Evaluation

We argue that a system can be designed to have some palpable behaviour from an activity perspective in the sense that the system is easily perceivable and allows for spontaneous interaction. During normal use it can be very hard for a user to perceive the difference between a system running a palpable framework or not. But if a breakdown occurs, these systems lose all their palpable qualities if the qualities only were implemented 'in the interface'. On the other hand, a system can run the palpable framework, without a user perceiving any palpability in the use of the system. If the system is not designed to communicate its palpability to the users, palpability will not be perceived by the users. But finally, if combined, a much higher level of palpability can be reached within a system. Palpability is not only about internal structure of the software, it is also about communication and interaction.

In Active Surfaces, as in other distributed systems that are characterised by a high level of configurability and limited output capability, the contingencies that might occur over time is of special interest and have to be dealt with. Especially those that can occur in a multi-user environment. Multi-user not only with

```
assembly PuzzleAsm {
  devices {
    device = urn:palcom://Tile_1;
  }
  services {
    Connectivity on device = Connectivity;
    Puzzle on device = Puzzle;
    HappyCoord on device = HappyCoord;
    LED on device = LED;
  }
  connections {
    Connectivity -> this;
    Puzzle -> this;
    HappyCoord -> this;
    LED -> this;
  }
  script {
    when connectivityUpdate from Connectivity {
      send connectivityUpdated(thisevent.param) to Puzzle;
    }
    when localHappy from Puzzle {
      send setled('1 1 1 1') to LED;
    }
    when sideHappy from Puzzle {
       send setled(thisevent.sides) to LED;
    }
    when localHappy from Puzzle {
      send localHappy() to HappyCoord;
    }
    when sideHappy from Puzzle {
       send sideHappy() to HappyCoord;
    }
    when globalHappy from HappyCoord {
      send setled('2 2 2 2') to LED;
    }
  }
}
```

Fig. 9. Puzzle assembly script

respect to the number of users, but also with respect to different kinds of users. The challenge here is to allow these users to be in control [16], a key challenge in ubiquitous computing addressed by Palpable computing. We will try to visualise this point with a simple scenario.

> *One day two therapists work at the swimming pool. During the day different children arrive to start their sessions. One of the therapists uses the assembler tile to program and then configure 5 tiles to take part in the therapeutic activity concerning one of the children. She makes a puzzle game with stable and blinking light feedback to indicate the different states of the system. While she is at lunch, her colleague takes 7 tiles to use with another child. By mistake, she includes one of the tiles configured in the 'first' game. As the first therapist returns after lunch, she tries to continue the activity. Now one of the tiles acts in a strange way, or not at all. As the second therapist now has finished her work, the first therapist cannot consult her to realise that she might have altered the game. As the first therapist perceives the situation, the Active Surfaces worked before lunch, and now they do not.*

In distributed and resource constrained systems, many error situations can occur and normally it can be hard for a user to find the reason behind a

problem or a mismatch. The assembler tile introduced before in this paper, can, besides being used in the therapeutic activity, also be used as an inspection tool. The therapist can utilise the IR communication protocol to inspect the running services within each tile. Through the inspection the therapist can understand what services and assemblies are running, how they are configured and detect whether there are resource problems that have to be solved. The therapist starts to inspect the game service, and realises immediately that the tile configuration has been altered. It was not a system error. The therapist reconfigures the tile and can carry on the activity in the swimming pool.

The Active Surfaces system has been developed together with the users (mainly therapists) of the system. The use of the Participatory Design [21] method including different iterations of mock-ups, prototypes and Wizard of Oz [22] sessions indicate that end-users can perceive and control the Active Surfaces as described above. Further full-scale trials have to be carried out to fully support this claim.

The simple scenario demonstrates the need for inspection and user control. Here the therapist initially perceives the behavioural mismatch as a bug or error in the system. In reality all components behave as they should, but one of the tiles have been reconfigured without the knowledge of the current user. It is an example of an error occurring over time in one of the distributed components, even when the system should have been idle. The user must be provided with tools that allow him to understand or 'look inside' the system to overcome the mismatch. It is important that the user understands that this is not an error; it is a misconfiguration that can be overcome.

5 Conclusions and Future Work

In this paper we have described the implementation of the Active Surfaces prototype, which is used in physical and cognitive rehabilitation work. We have shown in an example scenario how palpable qualities in a system can be valuable when the system behaves in unexpected ways. In those situations, it is beneficial for the user to have a system that is built as a set of services combined into assemblies. Palpable system allows for inspection, so errors and misconfigurations can be located and corrected by end users.

The Active Surfaces prototype has been implemented in a distributed, embedded system, executing in a set of floating tiles, and in a simulation framework running on a standard PC. Experiences gained during the work on the prototype provide input to the on-going development of the PalCom assembly concept. The prototype implementation has helped concretise requirements for supporting more powerful coordination logic, including coordination based on broadcast communication. The structure of the tile games calls for assemblies that span over multiple physical devices, and for decentralised assemblies that do not require particular devices always to be present. When assembly descriptions can

express such behaviour, less coordination logic has to be delegated to unbound services.

Acknowledgements. Thanks to Laura Cardosi, parents and children for their open-minded collaboration and continuous support during fieldwork and participatory design. We also would like to thank our colleagues at our universities, especially Alessandro Pollini, Patrizia Marti, Alessia Rullo, Boris Magnusson, Jacob Frølund, Henrik Gammelmark, and Mie Schou Olsen. The research was part of the European IST project PalCom.

References

1. Weiser, M.: The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. 3(3), 3–11 (1999)
2. Buxton, W.: Integrating the periphery and context: A new model of telematics. In: Proceedings of Graphics Interface, pp. 239–246 (1995)
3. Weiser, M., Brown, J.S.: Designing calm technology. PowerGrid Journal (1996)
4. PalCom - making computing palpable, http://www.ist-palcom.org
5. Grönvall, E., Marti, P., Pollini, A., Rullo, A.: Active surfaces: a novel concept for end-user composition. In: NordiCHI 2006. Proceedings of the 4th Nordic conference on Human-computer interaction, pp. 96–104. ACM Press, New York (2006)
6. PalCom: PalCom External Report no 50: Deliverable 39 (2.2.2): PalCom Open Architecture. Technical report, PalCom Project IST-002057 (2007)
7. Svensson, D.: PalCom Working Note #112: Service framework. Technical report, PalCom Project IST-002057 (2006)
8. Marti, P., Rizzo, A.: Levels of design: from usability to experience. In: Proceedings of HCI International (July 2003)
9. Marti, P., Moderini, C.: Creative design in safety critical systems. In: Proceedings of ECCE 11 (September 2002)
10. Corry, A.V., Hansen, K.M., Svensson, D.: Traveling architects – a new way of herding cats. Quality of Software Architectures, pp. 111–126 (2006)
11. Büscher, M., Kristensen, M., Mogensen, P.: Making the future palpable: Notes from a major incident future laboratory. In: Proceedings of the 4th International Conference on Information Systems for Crisis Response and Management (ISCRAM) (May 2007)
12. uClinux, http://www.uclinux.org/
13. Myers, B.A.: Visual programming, programming by example, and program visualization: a taxonomy. SIGCHI Bull. 17(4), 59–66 (1986)
14. Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., D'Amour, A.: Physical programming: designing tools for children to create physical interactive environments. In: CHI 2002. Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 299–306. ACM Press, New York (2002)
15. UNC20, http://www.unc20.net/
16. Schultz, U.P., Corry, E., Lund, K.V.: Virtual machines for ambient computing: Virtual machines for ambient computing: A palpable computing perspective. In: Black, A.P. (ed.) ECOOP 2005. LNCS, vol. 3586, Springer, Heidelberg (2005)
17. MIME Media Types, http://www.iana.org/assignments/media-types/

18. PalCom: PalCom External Report no 57: Deliverable 43 (2.6.2): End-User Composition: Software support for assemblies. Technical report, PalCom Project IST-002057 (2007)
19. Brønsted, J., Hansen, K.M., Ingstrup, M.: A survey of service composition mechanisms in ubiquitous computing. In: Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI) at Ubicomp (to appear, 2007)
20. Grönvall, E., Pollini, A., Rullo, A., Svensson, D.: Designing game logics for dynamic active surfaces. Presented at MUIA 2006: third international workshop on mobile and ubiquitous information access (2006)
21. Greenbaum, J., Kyng, M.: Design at work: cooperative design of computer systems. Lawrence Erlbaum Associates, Inc, Mahwah (1992)
22. Erdmann, R.L., Neal, A.S.: Laboratory vs. field experimentation in human factors–an evaluation of an experimental self-service airline ticket vendor. Human Factors 13(6), 521–531 (1971)

GPS-Based Location Extraction and Presence Management for Mobile Instant Messenger*

Dexter H. Hu and Cho-Li Wang

Department of Computer Science, The University of Hong Kong,
Pokfulam Road, Hong Kong
{hyhu,clwang}@cs.hku.hk

Abstract. Location is the most essential presence information for mobile users. In this paper, we present an improved time-based clustering technique for extracting significant locations from GPS data stream. This new location extraction mechanism is incorporated with Google Maps for realizing *cooperative place annotation* on *mobile instant messengers* (MIM). To enhance the context-awareness of the MIM system, we further develop an *ontology-based* presence model for inferring the location clues of IM buddies. The GPS-based location extraction algorithm has been implemented on a Smartphone and evaluated using a real-life GPS trace. We show that the proposed clustering algorithm can achieve more accurate results as it considers the time interval of intermittent location revisits. The incorporation of location information with the high-level contexts, such as mobile user's current activity and their social relationship, can achieve more responsive and accurate presence update.

1 Introduction

Instant Messenger (IM), characterized by its instantaneous message delivery and presence awareness, has become an important part of our everyday life. With the advancement of cellular technology (*e.g.*, GPRS), it has become possible to get the instant messenger services through mobile phones. As the mobile IM users could potentially move from place to place, location information becomes the most essential contextual cue among the mobile IM users. Location awareness among the communicating buddies makes collaboration more effective in field-work or on-site business as the communication attempts can be initiated without being "blind" or intrusive.

In recent years, owing to the low-cost and lightweight Global Positioning System (GPS) solutions, more and more mobile devices are equipped with GPS functionality. Geospatial data can be obtained by mobile users in real time. To make the GPS-based location service be truly useful for the mobile IM users, we address three issues in this research. First, the solution must be able to filter out useless GPS location data on the fly, as user's mobility is highly dynamic and

* This work is supported by National Natural Science Foundation of China (NSFC) Grant No. 60533040.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 309–320, 2007.

evolving. Moreover, the GPS-based location service should have an accurate and efficient location model to extract most of the places that users deem important in real life.

Second, after detecting the significant locations, raw location data (longitude, latitude, etc.) should be translated to more symbolic, personally meaningful place annotations. With the location extraction unit built on the mobile IM, all IM users become the location information providers. Users potentially can acquire location information from their buddies through relating new locations to some existing annotations. As thus, mobile users can rapidly access spatial information through such location knowledge sharing. So far this feature is not available in most mobile IM solutions.

Third, most instant messengers support presence management which checks the presence of the user's buddies (*e.g.*, "busy", "off line", "away") and provides a visual indication of each buddy's presence status on the IM client. Location awareness could potentially enable the development of more advanced presence management scheme for MIM. For example, the presence management in MIM should further consider what sort of presence information should be available to whom, and under which circumstances according to the given location information. It also requires to incorporate other context information (*e.g.*, activity status, people's relationship) to infer each buddy's presence status.

In this paper, we present a *mobile instant messenger* (MIM) with three new features: (1) *Improved location extraction algorithm*, an on-line clustering algorithm to extract significant locations more accurately from raw GPS data. We assume there is no location knowledge (*e.g.*, GPS trace) priori the execution of the clustering as the location service is usually needed in a new or partial familiar environment where not every place or path is known. (2) *Cooperative place annotation.* Google Maps is integrated to allow mobile users to share place markers among IM buddies for creating his/her personal map. (3) *Context-aware presence management.* Web Ontology Language (OWL) [3] is used to model buddies' relationship, locations, and activity for automatic presence management. We rely on public Web services like Google Calendar as sources of user contexts.

The rest of this paper is organized as follows. Section 2 explains the *i-Cluster* location extraction algorithm and cooperative place annotation. Section 3 discusses the context-aware presence management in MIM. Section 4 highlights the design of the MIM system. Section 5 reports the implementation details of of MIM and evaluation of its features. Conclusions are discussed in Section 6.

2 Location Extraction and Place Annotation

Identifying significant locations from user's trace is basically a clustering problem [4]. In the past, various location extraction solutions have been proposed based on different sources, such as GPS coordinates [2], GSM cell transition data [5], and Wi-Fi (or Bluetooth) beacons [1]. Intuitively, significant places are usually location visits having a recognizable duration [1] [2]. In some cases, we are also interested in places which may not have a long stay duration, but are revisited

shortly [5] [6]. This type of places include entrance of a parking lot, main gate of a university, junctions of street, etc. There are also situations, where a user's on-going task is disrupted unexpectedly and the user returns to the same place shortly afterwards to finish the task. These are all strong indicators of meaningful locations. We propose an improved time-based clustering algorithm (named *i-Cluster*), which can further extract these types of places.

The original time-based clustering algorithm [1] (called *TBC* afterwards) determines significant places where the user stays longer than a given time threshold t. A new run of clustering is started when distance between the new location and the centroid of the current cluster is larger than a threshold d. In general, it is difficult to tune the two parameters in order to extract all significant places aforementioned in real life.

Our algorithm takes additional consideration to the junction area of user's trace. We introduce a third parameter t_{intv} and use an auxiliary data structure *Tempplaces*. *Tempplaces* keeps track of those visited places with a duration of stay less than t, which are temporarily not qualified as significant places by the TBC algorithm. t_{intv} is a given threshold value that specifies the tolerable time interval of intermittent location revisits. Two temporary clusters in *Tempplaces* will be merged if user moves away from a cluster and returns within t_{intv} time.

The pseudo code of the *i-Cluster* is shown in Algorithm 1. We follow the same definition of parameter d and t as in the TBC algorithm. There are additional variables used in *i-Cluster*. The input to *i-Cluster* is *loc*, which is the new reading of GPS location data. *cl* is the current cluster, which records the centroid coordinate, first timestamp, last timestamp, and the size (number of GPS points) of the cluster. *Places* is used to record the extracted significant places. Function *Distance*() calculates the distance from a given point to the centroid of a cluster. Function *Duration*() measures the time duration of a user staying in the clustered area. To cater for the GPS positioning error and make sure the user is really moving away, *plocs* is used to temporarily keep a small number of pending location data. We report departure of user from current cluster if at least l samples are collected in *plocs*.

We explain the *i-Cluster* algorithm as follows. In line 1-3 (also line 29-30), we add the *loc* to current cluster *cl* if its distance to *cl* is shorter than *d*. *plocs* is cleared whenever *loc* falls within *cl* (line 3, 28). In line 6-7, a significant place is added to *Places* if *cl*'s duration is longer than t. Otherwise, *cl* will be inserted to *Tempplaces* (line 9-25) for potential merge. The merging process scans through *Tempplaces* in reverse time order (line 11), and tries to merge *cl* with a most recent temporary cluster created within t_{intv} time earlier than *cl* (line 13), which satisfies (1) the summation of the duration is no less than t, and (2) the distance between the centroids of the two clusters is no longer than d (line 16). Other temporary clusters beyond the t_{intv} time window are removed (line 22). The time gap (line 13) is the difference of the later cluster's first timestamp and the earlier cluster's last timestamp. After these steps, the algorithm starts a new cluster from *plocs.end* (line 26-28), as the user is moving away. We make sure there are at least l location data received in between two consecutive clusters.

Note that we merge two nearby clusters found in $Tempplaces$ by measuring the distance between the centroids of them. We set the merge distance threshold d (line 16) instead of a small one (*e.g.*, 2 meters), as the centroid of cluster does not timely reflect the current position of the user. User's current position may in fact be very close to the centroid of the cluster to be merged with the current cluster, while the distance between the two centroids is still in a distance of d.

Algorithm 1. *i-Cluster* (*loc*)

```
1: if Distance(cl, loc) < d then
2:    add loc to cl {/*Add the new data to current cluster if it's within distance range*/}
3:    clear plocs
4: else
5:    if plocs.length > l then
6:       if Duration(cl) > t then
7:          add cl to Places {/*A significant place found*/}
8:       else
9:          merged ⇐ false {/*Add the temporary cluster to Tempplaces for potential merge*/}
10:         add cl to the end of Tempplaces
11:         for j = Size(Tempplaces) − 2 to 0 do
12:            tc ⇐ jth cluster in Tempplaces
13:            if (Firsttimestamp(cl) − Lasttimestamp(tc)) < t_intv then
14:               dist ⇐ Distance(tc, cl_centroid)
15:               sum ⇐ Duration(cl) + Duration(tc)
16:               if dist ≤ d and sum ≥ t and merged = false then
17:                  merge cl, tc to a single cluster added to Places
18:                  remove cl, tc from Tempplaces
19:                  merged ⇐ true
20:               end if
21:            else
22:               remove tc from Tempplaces
23:            end if
24:         end for
25:      end if
26:      clear cl
27:      add plocs.end to cl
28:      clear plocs
29:      if Distance(cl, loc) < d then
30:         add loc to cl
31:      else
32:         add loc to plocs
33:      end if
34:   else
35:      add loc to plocs
36:   end if
37: end if
```

The *i-Cluster* algorithm has several merits. First it is space-efficient as we do not keep the GPS data belonged to a cluster. Besides, the memory size of $Tempplaces$ is bounded by the intermittent time value t_{intv} and the average speed v of user, since we only keep clusters within a time window of t_{intv}. In a worst case when the user keeps moving, the expected number of clusters nc in $Tempplaces$ can be estimated as:

$$nc = \frac{t_{intv}}{\frac{2d}{v}} = \frac{t_{intv}v}{2d} \tag{1}$$

In a typical case, where $d = 40$ meters, $t_{intv} = 1200$ seconds (20 minutes), $v = 5$ km/h (the average walking speed of pedestrians), nc is roughly 20. So the space

overhead induced by *i-Cluster* algorithm is not large, and the time complexity of merging clusters $O(nc)$ is thereby tolerable in resource-restricted mobile devices. As the cluster merging step is quite efficient, the performance of *i-Cluster* is as good as the TBC algorithm.

After detecting the significant locations, raw location data (longitude, latitude, etc.) should be converted to meaningful place labels. We implement a place annotation function similar to those used Google Maps [7] and GeoNote [8] on our MIM. Users can either enter their annotation (i.e., "points of interest") manually, then upload to a central map server, or they can select a place label among those created by their buddies whoever visited the same place before. The place annotation selection is usually subjective and it depends on user's focus of interest on the spot. We call this *cooperative place annotation*. More details are discussed in Section 4.

3 Context-Aware Presence Management

The co-awareness of buddy's location context can avoid unnecessary conversations (*e.g.*, "Where're you?", "What're you doing now?"), and achieve more efficient collaboration. For example, we can schedule a meeting at a place where each person is most close to, and when every member is available.

We further consider the *connectedness implications of presence* [9] in terms of MIM user's social networks and community. Given the useful location annotations generated by all users, the presence management subsystem determines what sort of location data be revealed to user's buddies. In our design, user's presence is represented by the triple "Status:Activity@Place". The subsystem will derive customized location indicators according to the activity they are currently involved and the social relationship between users and their buddies (or their roles in a group activity). In other words, a user's physical location is interpreted differently and different place labels could be displayed on his/her buddies' mobile phones.

We use the Web Ontology Language (OWL) [3] to model buddy relationship and domain knowledge involved in IM communications. By inferring on the ontology-based framework, we update buddies' presence in the buddylist accordingly. This method also helps to rank location recommendations while performing the cooperative place annotation. For example, the location recommendations by buddies to participate the coming group activity could be ranked higher.

Figure 1 shows the ontology model used in our MIM. The *presence* ontology is incorporated with *location*, *activity*, and *status* ontologies. The *hasPeoplerelation* property of *user* captures the social relationship between people. Currently, we define three sub-properties to reflect the common buddy relationship, including family, colleagues, and friends. These properties appear as *hasFamilyRelation*, *hasWorkRelation*, and *hasFriendRelation* in the ontology model.

The buddylist on the client is automatically refreshed whenever any salient context changes happen to buddies. The process is divided into three steps:

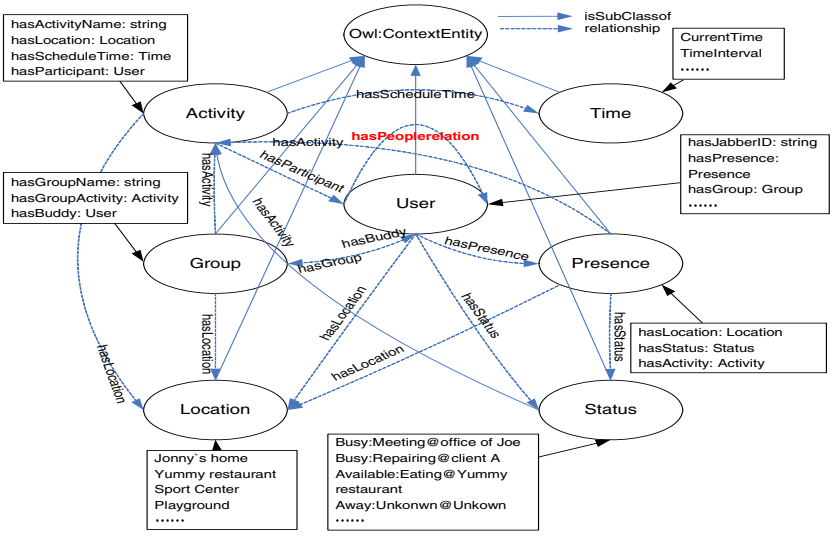


Fig. 1. Diagrammatic view of the MIM ontology model

1. Decide the current activity of the user (if any), which is triggered by location update of *i-Cluster*, starting of a scheduled event's time, etc.
2. Determine user's new presence to buddies based on the rules defined.
3. Generate each buddy's new buddylist pushed to client according to the update priority: (1) the buddylist showing members involved in the current activity, (2) the buddylist recording people nearby user's current location, and (3) buddy relationship.

4 The MIM System Design

The design of MIM is extended from *Smart Instant Messenger* (SIM) system [11], which was developed atop of the Jabber IM platform [13]. Figure 2 shows the MIM system architecture and communications between each component. The MIM client communicates with other buddies via the Jabber Server using the instant messenger protocols *XMPP*. It performs *i-Cluster* for detecting significant places as a background task on a GPS-connected mobile phone. It also handles the map download, rendering, and display of various types of buddylist. These are related to the GUI design.

On the server side, the Jabber server serves as a gateway to mediate the communications between MIM client and other server-side components. There is a *packet listener* in the Jabber server to parse the *XMPP* packet type, and activate the corresponding back-end service. There are four other components in the server side. Among them, the Jena server is the most complicated one. It realizes the context-aware presence management scheme based on the new ontology model. The Jena server employs a *context listener* to monitor each

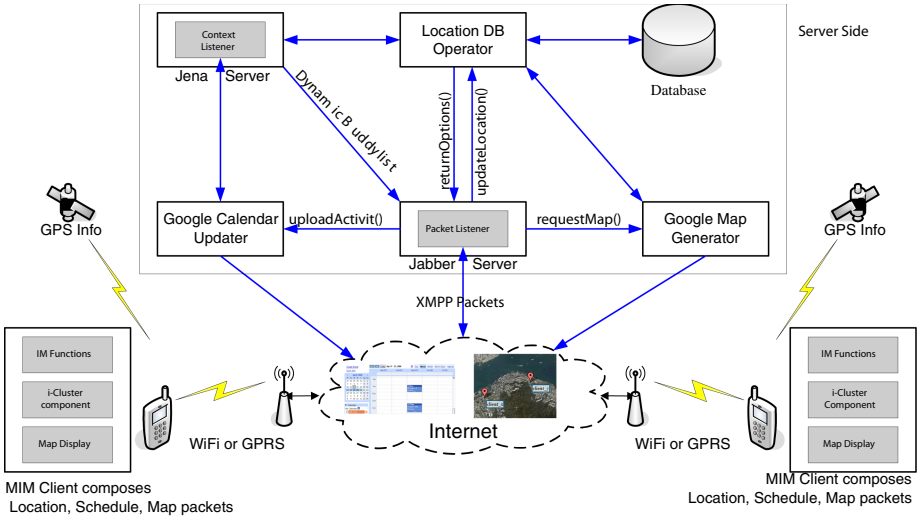


Fig. 2. Overview of the MIM System

buddy's location changes and calendar events. Based on these high-level contexts, Jena server infers the current activities of all users. Upon detecting a location change from a MIM client, Jena server determines the "Status:Activity@Place" triple to be sent to his/her buddies and when to deliver. The decision of location annotation and delivery time is user-dependent as it is based on the relationship between the user and his buddy, and the activity they are currently involved. To facilitate calendar entry creation and event query, the *Google calendar updater* is added.

The main function of the *location database operator* (LDBO) is to perform insertion, deletion, and query operations on the location database. LDBO stores various annotated location information produced by users through *i-Cluster* component in the MIM client. It records user's id registered in Jabber server, place's GPS coordinates and its semantic label, creation time of place data, and *hit number* which is the number of times a place's semantic label ever chosen by others MIM users. LDBO can reply queries with location recommendations that satisfy a given distance criterion or time range. It also provides the initial ranking of the location recommendations according to the hit number. These location recommendations will be further analyzed by the Jena server to derive more accurate recommendations based on other reasoning rules discussed in the previous section. To deal with the map generation, a *Google map generator* is designed to resize the map images and add location markers. It also caches the map images downloaded from Google Maps.

For supporting communication between MIM client and various server-side components, the XMPP protocol of Jabber is extended by defining three new types of *custom* packets [13]: *location*, *google-calendar-event* (*schedule*), *google-map-request* (*map*).

When MIM client detects that a user has stayed at a place for a short period of time, it automatically sends a location request to Jabber server. The request packet containing the latitude and longitude of user's current location is first handled by the Jabber server, which parses the type of the XMPP packet and forwards it to LDBO. LDBO will reply with the requested place's semantic labels filtered by Jena server. In case it is a new location (*i.e.*, no recommendation available), or the user decides to annotate himself (*i.e.*, rejecting all recommendations), the *i-Cluster* is resumed. Once it detects a significant place, it alerts user to enter a semantic location label. The location information is then sent to the Jabber server with a *google-calendar-event* packet containing centroid's GPS coordinates, starting time, ending time, user id, and the added semantic location label.

In case the received XMPP packet is a *map* packet, for requesting location information of other user(s) on the buddylist, the Jabber server will forward the request to the Google Map generator to create maps with location indication of buddies. It can either reply to client with an online map URL or transfer the converted map image with place markers.

5 Implementation and Evaluation

5.1 MIM Client and Map Display

We implemented two versions of MIM client on a Dopod C720W Smartphone running Windows Mobile 5.0 operating system in both C# and Java in the J2ME (MIDP 2.0) platform.

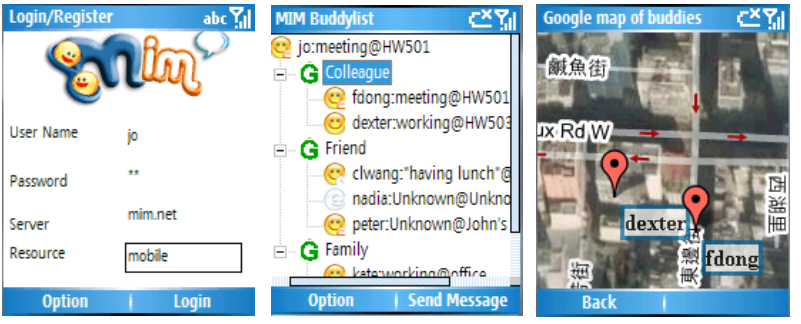
Figure 3 (a) shows the GUI of MIM client produced from Windows Mobile 5.0 Smartphone emulator. Figure 3 (b) shows the presence of Jo's friends, colleagues, families, including status icon, activity and location. Note that this test used the RFID reader to detect users' indoor location. The activity information was retrieved from the Google Calendar. Upon selecting a group/buddy, the user can choose to fetch and view google map image with location makers of the buddy/group members. The image size parameters sent to MIM server is determined by the screen size of the smartphone.

5.2 Evaluation of *i-Cluster* Algorithm

In this experiment, the Smartphone reads GPS data from a Holux GPSlim236 GPS receiver [14] via Bluetooth connection, as the C720W Smartphone does not have a built-in GPS receiver.

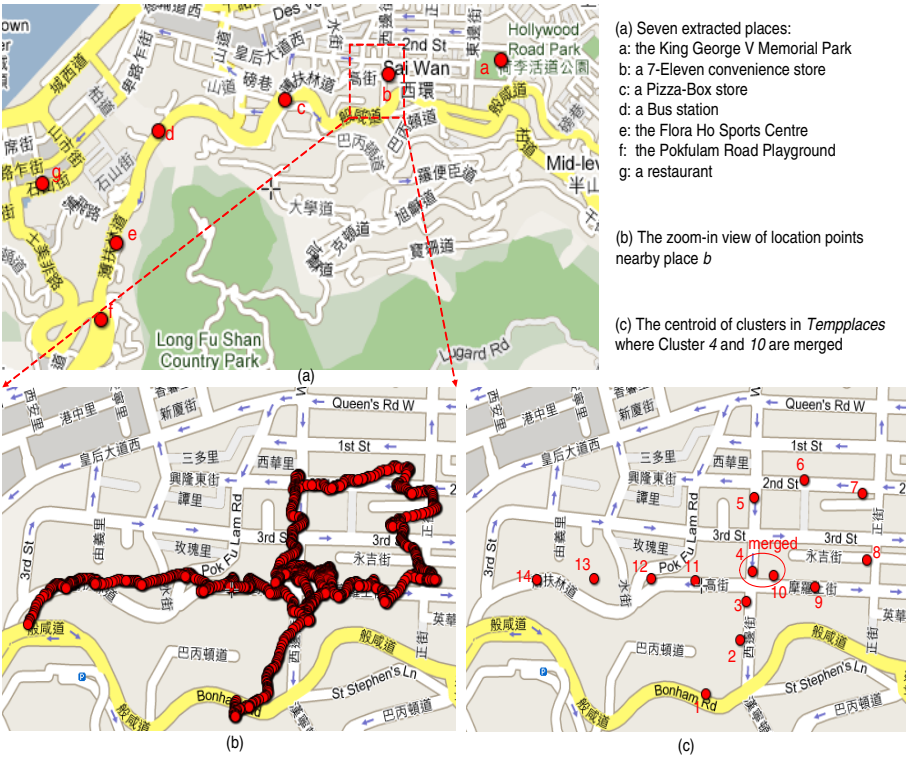
A sample user trace was collected by walking around the Sai Wan area of Hong Kong Island for a total time of 2.6 hours. The GPS receiver reports GPS position reading one per second. A total of 9373 GPS data points are collected. The MIM client only made use of longitude, latitude, and timestamp of the GPS position reading. Figure 4 (a) shows the area the mobile user has visited.

We set the parameters of the *i-Cluster* algorithm as $d = 40$ meters, $t = 300$ seconds, $t_{intv} = 1200$ seconds, and $l = 10$. The values of d and t are determined



(a) MIM Login GUI (b) Buddylist of Jo (c)Map with buddy locations

Fig. 3. MIM Client GUI and Map Display



(a) Seven extracted places:
a: the King George V Memorial Park
b: a 7-Eleven convenience store
c: a Pizza-Box store
d: a Bus station
e: the Flora Ho Sports Centre
f: the Pokfulam Road Playground
g: a restaurant

(b) The zoom-in view of location points nearby place *b*

(c) The centroid of clusters in *Tempplaces* where Cluster *4* and *10* are merged

(a) (b) (c)

Fig. 4. The experiment shown in Google Maps format

according to the knee point in [1]. We set $l = 10$ for keeping the pending locations. The value of t_{intv} is set to 20 minutes as we believe when an important task is disrupted unexpectedly, the user usually will come back to the same place soon. To evaluate the accuracy of the *i-Cluster* algorithm, we disable the

stationary detection function, *i.e.*, the MIM client is not allowed to query location recommendation from MIM server.

Figure 4 (a) shows the seven extracted significant places by *i-Cluster*. They are plotted by the GPS visualizer [15] as *waypoints* in Google Maps format. We found the reported locations are very close to the places we visited by viewing the street map.

We further investigate those clusters detected by *i-Cluster*. Figure 4 (b) shows the GPS data points nearby place *b*. Figure 4 (c) shows the centroids of the corresponding clusters in the *Tempplaces*. They are numbered from 1 to 14 in time order. In the experiment, we stayed at the 7-Eleven convenience store shortly (about 3.2 minutes), left for an ATM machine, then came back again through another block after 6.5 minutes, and stayed another 3.4 minutes at the store. As shown, cluster 4 and cluster 10 are merged into a single cluster as place *b* (the 7-Eleven convenience store in Figure 4 (a)), which would be simply ignored if TBC algorithm is used.

5.3 Presence Reasoning Logic

We use Jena's Rule Java object [12] to define reasoning rules for determining user's new presence. The basic rules are in a human readable form of "*antecedent => consequent*".

Table 1. Basic rules for inferring user's presence

Cases	Antecedents	Consequents
Determine the presence shown to a colleague	hasTime(CurrentTime, "Work time") hasWorkRelation(?x1, ?x2) hasSameActiviy(?x1, ?x2)	hasAvailableStatus(?x1, ?x2) hasSameGroup(?x1,?x2)
Determine the presence shown to family member	hasTime(CurrentTime, "Work time") hasFamiliyRelation(?x1, ?x2)	hasBusyStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationShown(?x1,?x2)
Determine the presence shown to friend	hasTime(CurrentTime, "Work time") hasFriendRelation(?x1, ?x2)	hasAwayStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationHidden(?x1,?x2)
Determine the presence shown to colleague when it's off-duty	hasTime(CurrentTime, "Off-duty") hasWorkRelation(?x1, ?x2)	hasAwayStatus(?x1,?x2) hasActivityHidden(?x1,?x2) hasLocationHidden(?x1,?x2)
Determine the presence shown to friends when it's off-duty	hasTime(CurrentTime, "Off-duty") hasFriendRelation(?x1, ?x2)	hasAvailableStatus(?x1,?x2) hasActivityShown(?x1,?x2) hasLocationShown(?x1,?x2)
Determine the presence shown to family member when it's off-duty	hasTime(CurrentTime, "Off-duty") hasFamilyRelation(?x1, ?x2)	hasAvailableStatus(?x1,?x2) hasActivityShown(?x1,?x2) hasLocationShown(?x1,?x2)

Table 1 shows the rules we used in MIM for automatic update of user's presence. The first three rules infer user's availability to his co-workers during work time, but avoid exposure to friends and family members. The last three show user's presence to his friends and families, whereas hide it from colleagues to avoid disturbance during his spare time.

We evaluated the responsiveness of MIM presence model performed by the Jena server. In this evaluation, the MIM client was connected to a desktop

server via Wi-Fi connection. We found the average processing time for a typical reasoning on context changes is around 2-4 seconds. We will investigate more time-efficient reasoning solutions in the future.

6 Conclusion and Future Work

In this paper, we present a GPS-based location extraction system to support MIM. We designed the *i-Cluster* algorithm to better locate the significant places that would be ignored by previous time-based clustering algorithms. We found the cooperative place annotation scheme can greatly reduce the computing cost of the GPS-based location extraction method and produce more accurate location information. With the wide adoption of low-cost GPS receiver built on mobile devices, the co-awareness of the location information has a good potential to develop more powerful context-aware applications. We modeled IM-related concepts and people's relationship using ontologies to automate the presence inference. We believe this is an emerging trend in the development of IM. In the future, we shall further extend our MIM presence model and study more intelligent and faster reasoning solutions.

Lastly, we found the powerful Web services have made it very convenient to build the proposed MIM functions. We suggest that the location-aware presence management should make use of public Web services like Google Maps and Google Calendar. The "attachment" of user's context information to the public Web with more reliable and stable services is a viable solution to realize "pervasive" context-aware computing, especially for mobile users.

References

1. Kang, J.H., Welbourne, W., Stewart, B., Borriello, G.: Extracting Places from Traces of Locations. In: Proc. WMASH, pp. 110–118. ACM Press, New York (2004)
2. Ashbrook, D., Starner, T.: Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. Personal and Ubiquitous Computing 7(5), 275–286 (2003)
3. OWL Web Ontology Language, http://www.w3.org/TR/owl-features
4. Zha, H., Ding, C., Gu, M., He, X., Simon, H.D.: Spectral Relaxation for K-means Clustering. Neural Information Processing Systems 14, 1057–1064 (2001)
5. Nurmi, P., Koolwaaij, J.: Identifying Meaningful Locations. In: 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services, San Jose, CA (July 17–21, 2006)
6. Schmid, F., Richter, K.F.: Extracting Places from Location Data Streams. In: Zipf, A. (eds.), Workshop Proceedings (UbiGIS), Münster, Germany
7. Google Maps API, http://www.google.com/apis/maps
8. Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E., Bylund, M.: GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) Ubicomp 2001: Ubiquitous Computing. LNCS, vol. 2201, pp. 2–17. Springer, Heidelberg (2001)

9. Rettie, R.: Connectedness, Awareness and Social Presence. In: Proc. PRESENCE 2003, online proceedings
10. Google Calendar Data API, http://code.google.com/apis/calendar/overview.html
11. Law, C.F., Zhang, X., Chan, M.S.M., Wang, C.L.: Smart Instant Messenger in Pervasive Computing Environments. In: The First International Conference on Grid and Pervasive Computing, Taichung City, Taiwan (May 3-5, 2006)
12. Jena: a Semantic Web Framework for Java, http://jena.sourceforge.net
13. Jabber Instant Messenger, http://www.jabber.org
14. GPSlim236 GPS Receiver, http://www.holux-uk.com/Products/gpslim236/index.shtml
15. GPS Visualizer, http://www.gpsvisualizer.com/map

Bilateration: An Attack-Resistant Localization Algorithm of Wireless Sensor Network*

Xin Li[1], Bei Hua[2,**], Yi Shang[3], Yan Guo[4], and LiHua Yue[5]

[1,2,4,5]Department of Computer Science and Technology
University of Science and Technology of China, Hefei 230027, China
[1,2,4]Mobile Computing Laboratory
Suzhou Institute for Advanced Study, Suzhou, Jiangsu 215123, China
{xinxinol,guoyan6}@mail.ustc.edu.cn, {bhua,llyue}@ustc.edu.cn
[3]Department of Computer Science
University of Missouri-Columbia, Columbia, MO 65211, USA
shangy@missouri.edu

Abstract. Most of the state-of-the-art localization algorithms in wireless sensor networks (WSNs) are vulnerable to attacks from malicious or compromised network nodes, whereas the secure localization schemes proposed so far are too complex to be applied to power constrained WSNs. This paper provides a novel secure scheme "Bilateration" which is derived from multilateration but can be calculated more accurately and quickly to resolve the positions of unknown nodes without explicitly distinguishing what kind of location attacks the WSN is facing. This paper also compares Bilateration with three existing multilateration solutions that optimize the location estimation accuracy via LS, LMS and LLMS respectively in a simulated threat environment. The experiment results show that Bilateration gets the best tradeoff among estimation error, filtering ability and computational complexity.

Keywords: localization, WSNs, multilateration, Bilateration, LS.

1 Introduction and Related Work

A WSN may run in a hostile environment without any supervision, where the attackers may easily threat the functionality of position-aware applications by exploiting the vulnerabilities of the localization schemes. There are mainly two types of attacks aiming at the localization process in WSNs [2]. The first type is launched by malicious nodes that are not a part of the network and controlled by an attacker. Typical attacks include modifying distance, jamming communication and creating wormholes [3] [4] in the network. The second type is launched by compromised nodes that are a part of the network and can authenticate

* This work was supported by the National Natural Science Foundation of China under Grant No.60673173 and No.60673111, and the Fund for Foreign Scholars in University Research and Teaching Programs.

** To whom correspondence should be addressed.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 321–332, 2007.

themselves as honest nodes, but are controlled by an attacker. They report false positions and disseminate false topology information. Most of the existing localization algorithms don't have the ability to filter out incorrect information, thus are vulnerable to various location attacks.

Recently, some secure localization schemes have been proposed to resist the attacks launched by compromised or malicious nodes. The most common techniques include location-verification[5], distance-verification[6,7], distance-bounding plus some symmetric key cryptography[8], RSS measurements[10] and "packet leashes"[4]. However these methods always require powerful calculation ability, precise synchronization, fast transmission, or somewhat training, etc, which are not suitable for such tiny, low-cost, power constrained sensor nodes. Alternatively, localization based on least median squares (LMS)[9] has been introduced to improve the resilience and accuracy of localization, which however is not suitable to WSN as well due to the heavy burden of calculation. Then [9] chooses to formulate a linearization of the LS (LLMS) location estimator in order to reduce the computational complexity of LMS at the cost of accuracy.

The main contributions of our work are in two aspects. Firstly, unlike the traditional secure localizations which introduce countermeasures to every possible attack, we propose a novel secure localization mechanism Bilateration, which is efficient in calculation and independent of the type of attacks. Secondly, we compare the performance of Bilateration with three multilateration solutions using LS, LMS, LLMS in the simulated settings, and the results show that our method outperforms the other secure schemes in estimation accuracy, filtering ability and computational complexity.

The remainder of this paper is organized as follows. Section 2 formulates the secure localization problem; section 3 overviews the basic idea of LS, LMS and LLMS; section 4 describes Bilateration algorithm; section 5 compares the performance of the above four algorithms; section 6 concludes the paper.

2 Problem Formulation

We consider a homogeneous network consisting of a set of wireless sensor nodes, including anchors and unknown nodes. Sensor nodes are equipped with radio transceivers, and two nodes can communicate with each other if the distance between them is within node's radio range. Each node can measure the distance to other nodes through TDOA, RSSI or something like DV-HOP with a white Gaussian noise:

$$d_{measured}=d_{real}+noise,\ noise\sim N(0,VD) \tag{1}$$

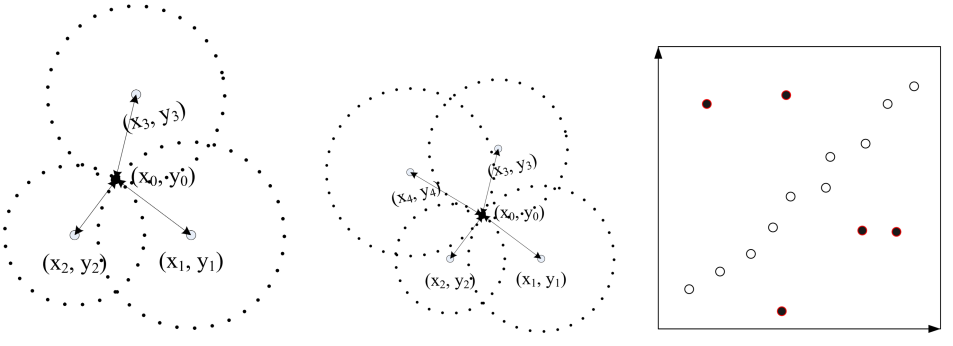
The most remarkable feature of Bilateration is that it only cares about the result of attack, i.e., the received distance and/or reference location might be false, but needs not distinguish who launches the attack or what kind of attack it is. To simply the description, in this paper we use compromised nodes to refer to both malicious nodes and compromised nodes that issue false distance and/or

reference location information in the network. The threat model we consider is as follows: a few anchor nodes have been compromised and purposely disseminate randomly false reference positions; moreover compromised nodes may not be detected by other means.

Suppose an unknown node located at (x_0,y_0) has collected a set of reference positions $\{(x_1,y_1),\cdots,(x_n,y_n)\}$ and measured distances $\{d_1,\cdots,d_n\}$ to these nodes. In an idealistic environment setting without any noise and threat, these positions and distances satisfy the following n equations:[-4mm]

$$\begin{aligned}(x_1-x_0)^2+(y_1-y_0)^2&=d_1^2\\ &\vdots\\ (x_n-x_0)^2+(y_n-y_0)^2&=d_n^2\end{aligned} \tag{2}$$

If $n\geq 3$, (x_0,y_0) can be uniquely determined by solving any 3 of the equations if the selected anchors are not in a line. This method is the classical trilateration algorithm. In a 2D plane, solution of trilateration is the intersection of three circles centered at three anchors(fig.1(a)), and solution of multilateration (2) is the intersection of all n circles (fig.1(b)). However, in a real noisy environment with imprecise reference location and/or distance measurements, n circles do not intersect at one point. Therefore an objective function (3) is used to minimize the error between estimated position and real position.



(a) Trilateration (b) Multilateration (c) Outliers

Fig. 1. Multilateration and Outliers

Generally speaking, in a noisy environment without compromised nodes, multilateration with LS is not a bad choice. However, in an environment with some compromised nodes, LS is not good since the estimated position $(\hat{x_0},\hat{y_0})$ may be adversely "removed" far away from the optimal position by compromised nodes. In order to get rid of the "outliers" (see fig.1(c)) caused by compromised nodes, LMS and linear LMS are applied to replace LS in estimation process.

3 LS, LMS and LLMS

3.1 Least Square

Multilateration with LS is to minimize the difference between the estimated position $(\hat{x_0},\hat{y_0})$ and the real position (x_0,y_0) of a node, see (3).

$$(\hat{x_0},\hat{y_0})=arg\ \min_{(x_0,y_0)}\sum_{i=1}^{n}[\sqrt{(x_i-x_0)^2+(y_i-y_0)^2}-d_i]^2 \tag{3}$$

This method usually involves some kind of iterative searching technique such as gradient descent or Newton method. To avoid local minimum LS must run several times with different initial starting points, which is expensive in terms of computing overhead. Moreover, it is vulnerable in the presence of compromised nodes, e.g., if an unknown node receives a false position sent by a compromised node, the estimated position may deviate significantly from its true value even if the measured distance is accurate. This is because pure LS tries to achieve a global optimality of all samples including outliers.

3.2 Least Median Square

To increase the resilience of multilateration with LS, least median squares (LMS) is proposed in [9]. Instead of minimizing the summation of the residue squares, LMS tries to minimize the median of the residue squares:

$$(\hat{x_0},\hat{y_0})=arg\ \min_{(x_0,y_0)}\ med_i[\sqrt{(x_i-x_0)^2+(y_i-y_0)^2}-d_i]^2 \tag{4}$$

According to [9], the procedure for implementing the robust LMS algorithm is summarized as follows:

1. Set n=4 as the appropriate subset size.
2. Set $M=\begin{cases}20,\ if\ n>6\\ \binom{n}{4},\ otherwise\end{cases}$ as the appropriate total number of subsets.
3. Randomly draw M subsets of size n from the heard anchors $\{(x_0,y_0),\cdots,(x_n,y_n)\}$. Calculate the estimation $(\hat{x_0},\hat{y_0})_j$ using LS for each subset and the corresponding median of residues $\{r_{ij}^2\}$ for every $(\hat{x_0},\hat{y_0})_j$. Here i=1,2,$\cdots$,n is the index for heard anchors, while j=1,2,$\cdots$,M is the index for the subsets.
4. Set $m=arg\ min_j\ med_i\{r_{ij}^2\}$, then $(\hat{x_0},\hat{y_0})_m$is the subset estimation with the least median of residues, and $\{r_{im}\}$ is the corresponding residues.
5. Calculate s_0=1.4826$(1+\frac{5}{n-2})\sqrt{med_i r_{im}^2}$.
6. Assign weight ω_i to each heard positions with equation $\omega_i=\begin{cases}1,\ \left|\frac{r_i}{s_0}\right|\leq\lambda\\ 0,\ otherwise\end{cases}$.
7. Do LS to all heard positions with weights $\{\omega_i\}$ to get the final $(\hat{x_0},\hat{y_0})$.

3.3 Linear LMS

Considering that finding estimation for M subsets requires a lot of computation, [9] transforms nonlinear LS into linear LS, which is a suboptimal but more computationally efficient algorithm.

1. Average all the left parts and right parts of (2):

$$\frac{1}{n}\sum_{i=1}^{n}[(x_i-x_0)^2+(y_i-y_0)^2]=\frac{1}{n}\sum_{i=1}^{n}d_i^2 \tag{5}$$

2. Subtract (5) from each equation in (2), and linearizes to get the following new equations:

$$\begin{aligned}(x_1-\frac{1}{n}\sum_{i=1}^{n}x_i)x_0+(y_1-\frac{1}{n}\sum_{i=1}^{n}y_i)y_0&=\frac{1}{2}(x_1^2+y_1^2-d_1^2-\frac{1}{n}\sum_{i=1}^{n}(x_i^2+y_i^2-d_i^2))\\ &\vdots\\ (x_n-\frac{1}{n}\sum_{i=1}^{n}x_i)x_0+(y_n-\frac{1}{n}\sum_{i=1}^{n}y_i)y_0&=\frac{1}{2}(x_n^2+y_n^2-d_n^2-\frac{1}{n}\sum_{i=1}^{n}(x_i^2+y_i^2-d_i^2))\end{aligned} \tag{6}$$

3. Estimate $(\hat{x_0},\hat{y_0})$ by linear least squares.

Transforming nonlinear LS into linear LS saves much computation, since the solution can be calculated directly from (6) without iterative searching and repeating. Furthermore, the solution of linear LS can be used as the starting point of nonlinear LS to prevent nonlinear LS from getting trapped in a local minimum. In our simulation, we use this starting point to do nonlinear LS.

However, due to the subtraction, the optimal solution of linear equations in (6) is not exactly the same as that of nonlinear LS in (2), which means much accuracy is lost, especially when the number of heard anchors is small. In the experiments of [9], as the number of heard anchors is fixed to 30, linear LS is acceptable as it still performs very well. However, 30 heard anchors per unknown node is not practical in the realistic settings.

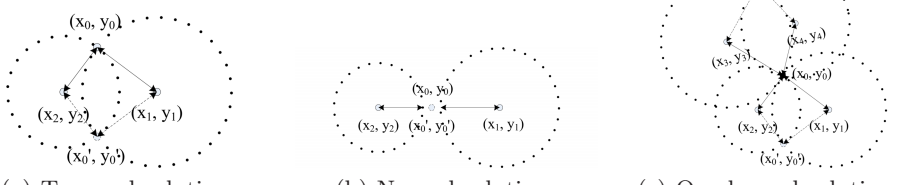
4 Bilateration

Due to inherent limitation, the performance of LMS and LLMS is poor when the number of heard anchors is small, or the percentage of outliers exceeds 50% even if there are still many usable samples. The goal of bilateration is to achieve the same accuracy as LMS and the same computational speed as LLMS, meanwhile its performance is less affected by the number of heard anchors and the percentage of outliers.

In an idealistic environment without measurement noise and attacks, if n=2 we can solve (2) as follows:

$$
\begin{aligned}
x_0 &= \frac{-(mn-ny_1-x_1)}{1+n^2} \pm \frac{\sqrt{2(nx_1+m)y_1-y_1^2-n^2x_1^2-2mnx_1-m^2+(1+n^2)d_1^2}}{1+n^2} \\
y_0 &= m+nx_0 \\
m &= \frac{1}{2}\frac{(x_1^2-x_2^2)+(y_1^2-y_2^2)-(d_1^2-d_2^2)}{y_1-y_2} \\
n &= -\frac{x_1-x_2}{y_1-y_2}
\end{aligned}
\tag{7}
$$

Evaluation of (7) is very fast given the value of $\{(x_1,y_1),(x_2,y_2),d_1,d_2\}$. The real solutions for (x_0,y_0) are called ***candidate positions***, which in a 2D plane are the intersections of two circles (see fig.2(a)); the complex solutions are not considered in this paper (see fig.2(b)). If another two anchors (at least one of the them doesn't belong to $\{(x_1,y_1),(x_2,y_2)\}$) and corresponding distances are selected, another two candidate positions are solved for (x_0,y_0). Among the 4 candidate positions, at least 2 positions overlap each other, and this overlapped point is the correct solution for (x_0,y_0) (see fig. 2(c)). If more anchor positions and distances are available, more overlapped points will occur. In this way, even most of the heard anchors are compromised, this method can correctly locate an unknown node as long as at least three anchors and corresponding distance measurements are accurate.



(a) Two real solutions (b) No real solutions (c) Overlapped solutions

Fig. 2. Bilateration

In real noisy environment, there may be no overlapped points due to distance error. However, there is reason to believe that correct positions should be close to each other if the distance error is bounded. We define:

Correct candidate positions: *a group of candidate positions, in which there is at least one position whose distances to the other members are less than the threshold* δ.

Candidate neighbors: *two candidate positions between which the distance is within* δ.

For an unknown node μ, the procedure for implementing our Bilateration algorithm is summarized as follows:

1. If $n \leq 3$, set μ as un-localized and terminate the algorithm. This situation will not be considered in our performance comparison, because there is no way to distinguish which position is false.

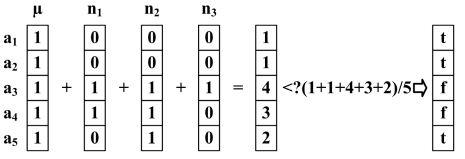


Fig. 3. Who is the compromised node

2. Exhaust all the combinations of two anchors and the corresponding measured distances $\{(a_i,d_i),(a_j,d_j)\}$ to evalute (7), and suppose M candidate positions $\{c_1,\cdots,c_M\}$ have been solved from the $\binom{n}{2}$ combinations.
3. For each candidate position c_i, calculate $\{D_{i1},\cdots,D_{ii-1},D_{ii+1},\cdots,D_{iM}\}$, where D_{ij} is the distance between c_i and c_j, i,j=1,2,$\cdots$,M is the index to candidate positions.
4. For each c_i, find out all the distances shorter than the threshold δ and get $\{D_{ip},\cdots,D_{it}|D_{ip}<\delta\wedge\cdots\wedge D_{it}<\delta,D_{ip},\cdots,D_{it}\in\{D_{i1},\cdots,D_{ii-1},D_{ii+1},\cdots,D_{iM}\}\}$, set $n_i=|D_{ip},\cdots,D_{it}|$. ($|\cdot|$denotes the cardinality of a set).
5. Find out $m=argmax_i\{n_i\}$, suppose $\{D_{mp},\cdots,D_{mt}\}$ are the distances between c_m and its candidate neighbors $\{c_p,\cdots,c_t\}$; find out the corresponding anchors $\{a_l,\cdots,a_q\}\subseteq\{a_1,\cdots,a_n\}$ from which $\{c_m,c_p,\cdots,c_t\}$ are solved; set the weights of $\{a_l,\cdots,a_q\}$ as 1; set the weights of the other heard anchors as -1.
6. Exchange the weight table with its neighbors.
7. Collect all the weight tables from its neighbors; pick out the common heard anchors; add their weights together; set the anchors whose weight is less than the average weight as the compromised nodes. (see fig.3)
8. Delete the candidate positions caused by compromised nodes from $\{c_1,\cdots,c_M\}$; set the average of all the left candidate positions as the final estimated position e_μ.

If the unknown node hears 4 different positions including 1 false position, LMS and LLMS are unable to deal with this situation, whereas our scheme can find out the correct positions if the distance between the correct candidate positions is shorter than δ .

5 Simulation

To evaluate Bilateration, we simulated it and multilateration with LS, LMS, and LLMS (abbr. to LS, LMS and LLMS) on Matlab, and compared them in terms of estimation error, ability of false position filtering, and computational complexity in simulation environment. Estimation error is the average variance between estimated locations and real locations. Ability of false position filtering is the average number of false positions that are used in the location estimation of each unknown node. Each data point represents the average value of 500 trials

with different random seeds. We use ideal LS as a benchmark in the performance comparisons, which can filter out all the compromised anchors before estimation.

In our simulation settings, we have the following definitions and assumptions.

- Anchors and unknown nodes are uniformly distributed in an area of $200\times 200m^2$.
- The coordinates of false positions, x and y, are independent and identically follow normal distribution $N(100,VP)$, where VP varies from 20 to 200m.
- The noise of measured distance obeys normal distribution $N(0,VD)$, where VD varies from 0 to 50m.
- R is the radio range of node, and is fixed to $50m$ in our experiments.
- NA is the average number of heard anchors by each unknown node.
- NU is the average number of heard unknown nodes by each unknown node.
- CP is the percentage of compromised anchors, and varies from 0 to 1.

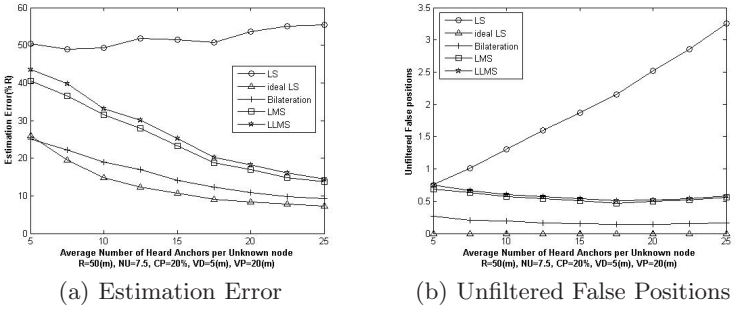
In the following experiments, without specification, the default environment settings are: VP=$20m$, VD=$5m$, NA=7.5, NU=7.5 and CP=0.2. After a lot of experiments with different δ and λ, we find out that the optimal δ for Bilateration is 10, and the optimal λ for both LMS and LLMS is 1.5. We omit the detailed performance comparison due to limitation of space.

5.1 Influence of Average Number of Anchors

In this experiment, we investigate the influence of average number of heard anchors (NA) on the performance of the four localization algorithms.

In fig.4(a), except for LS whose estimation error increases about 5% when NA increases from 5 to 25 due to its lack of filtering ability, the estimation error of other four algorithms (including Ideal LS) decreases. Bilateration has lower estimation error than LMS and LLMS, but their gap shrinks when NA increases.

Fig.4(b) compares the filtering ability of all algorithms. Since LS doesn't filter out outliers and CP is fixed, the false positions used by LS increases with NA. The number of unfiltered false positions used by Bilateration is smaller than that used by LMS and LLMS; that is to say, Bilateration has stronger filtering ability



(a) Estimation Error (b) Unfiltered False Positions

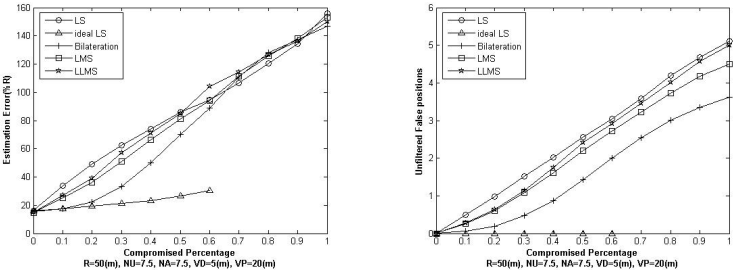
Fig. 4. Influence of Average Number of Anchors

than LMS and LLMS. This explains why Bilateration has lower estimation error than LMS in a hostile environment: the stronger filtering ability compensates for the suboptimal estimation accuracy.

In fig.4(a), the estimation error of Bilateration is close to that of Ideal LS all the time, whereas LMS and LLMS requires many more anchors to get the same accuracy. Since there are only a few anchors in a real wireless sensor network, this result shows that Bilateration is more suitable to real settings.

5.2 Influence of Percentage of Compromised Nodes

In this experiment, we investigate the influence of compromised percentage (CP) on the performance of algorithms. It is interesting to observe that ideal LS terminates when CP reaches 0.6, this is because the number of un-compromised anchors is smaller than 3 for each unknown nodes with NA=7.5. Therefore it is meaningless to discuss the performance for CP larger than 0.6.



(a) Estimation Error (b) Unfiltered False Positions

Fig. 5. Influence of Percentage of Compromised Nodes

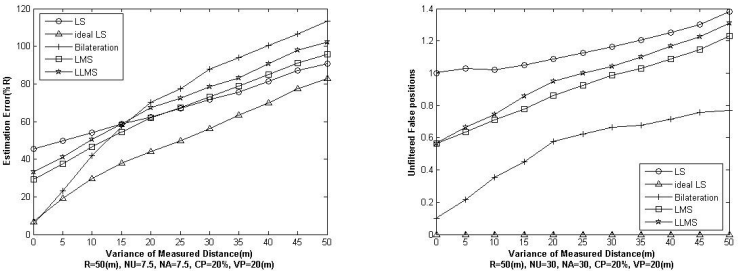
In fig.5(a), the estimation error of Bilateration is lower than that of LS, LMS and LLMS all the time when CP is smaller than 0.6, which shows that Bilateration is less affected by CP. However the four curves tend to approach when CP increases, since there is no difference among them when no right position is available.

In fig.5(b), the number of unfiltered false positions used by Bilateration is smaller than that used by LS, LMS and LLMS, which shows that Bilateration has the strongest filtering ability.

5.3 Influence of Distance Measurement Error

In this experiment, we investigate the influence on distance measurement error on the performance of algorithms.

In fig.6(a), the estimation error of Bilateration increases rapidly as the variance of distance (VD) increases. The estimation error of Bilateration is lower than that of LS, LMS and LLMS when VD is less than 13, then it exceeds them



(a) Estimation Error (b) Unfiltered False Positions

Fig. 6. Influence of Distance Measurement Error

quickly. We observe that the estimation error of Bilateration does not reach 0 even when VD is 0, since δ=10 allows some false positions to participate in the location estimation (fig.6(b)). If δ is set to 0, then Bilateration can filter out all the false positions when VD is 0. LMS and LLMS outperform LS when VD is less than 22 and 15 respectively, and then lost their advantage as well.

In fig.6(b), the number of unfiltered false positions used by all the four algorithms increase as VD increase, since large distance error makes it more difficult to distinguish between correct position and false position. Therefore if the measured distance is not accurate enough, the filtering ability of algorithm has no meaning.

This experiment shows that Bilateration is more suitable to work in an environment with moderate noise that is less than 24% or radio range.

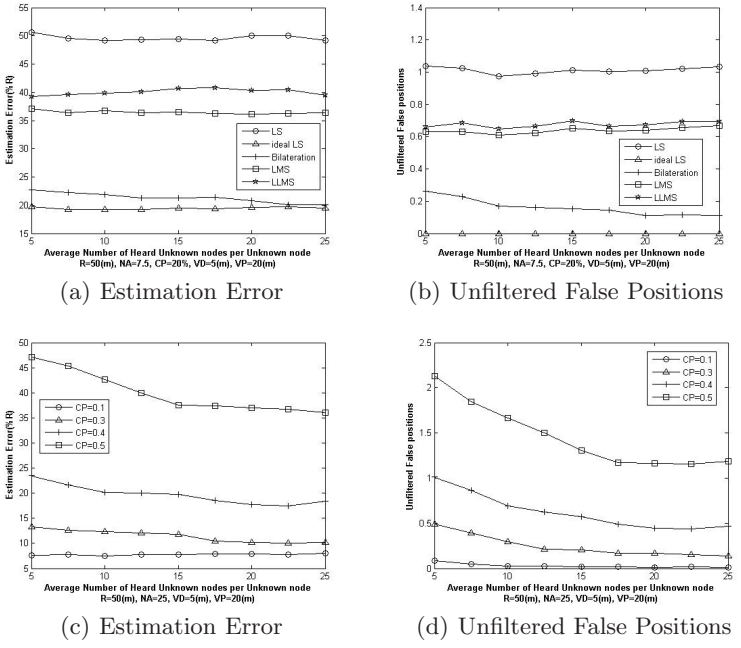
5.4 Tradeoff Between Performance and Communication Complexity

Bilateration is the only algorithm which needs to communicate with heard unknown nodes to identify the compromised nodes. However, the performance of Bilateration with small NA is not sensitive to the average number of unknown nodes. Meanwhile, the performance of the other three algorithms are not sensitive to the average number of unknown nodes (Fig.7(a)(b)). So, in this experiment, we only evaluate Bilateration with NA=25 and different CPs.

Fig.7(c)(d) evaluate Bilateration with different CPs. The estimation error and unfiltered false positions of Bilateration with bigger CP decreases more rapidly than those with smaller CP as NU increases. So the strategy of exchanging weight tables is more useful for Bilateration with big NA and CP. In other words, if CP or NA is not big we can abandon this strategy to save energy.

5.5 Computation Complexity Analysis

Since Bilateration, LMS and LLMS primarily differ in the means of estimation, we only analyze the amount of computation involved in estimation. Suppose that an unknown node μ hears n anchors.



(a) Estimation Error (b) Unfiltered False Positions

(c) Estimation Error (d) Unfiltered False Positions

Fig. 7. Influence of Average Number of Unknown nodes

In LMS, μ needs to do LS estimation $\binom{n}{4}+1$ times, when $n \leq 6$ or 21 times when $n>6$. In each round of LS estimation except for the last round, four anchor positions are involved in the estimation calculation, and in the last round all the unfiltered positions are involved. In order to avoid local minimum, solution of linear LS is used as a start point to search the global optimality, which adds another $\binom{n}{4}+1$ or 21 times of linear LS calculation. It is possible to use LS without preliminary linear LS, however, LS may need to search more times for optimal solution from different start point, and moreover the solution may be trapped into local minimum.

In LLMS, μ needs to do linear LS $\binom{n}{4}+1$ times when $n \leq 6$ or 21 times when $n>6$. The amount of computation involved in linear LS is much less than that involved in LS.

In Bilateration, μ needs to evaluate (7) $\binom{n}{2}$ times to find out all the candidate positions, and then perform $4\binom{n}{2}^2$ times of distance calculation between each candidate position to every other candidate position. All the computation only involves simple algebraic calculation, so Bilateraton runs much faster than LMS and comparable with LLMS, which was verified by our experiments as well.

6 Conclusion and Future Work

In this paper we propose Bilateration, an attack-resistant localization algorithm that tries to find a set of close-by positions from all candidate positions and use

the average of these close-by positions as the estimated position. Bilateration is resilient to all kinds of position and distance cheating attacks in the sense that it only cares about the result of attacks rather than the process of attacks; and is more close to the real world.

In the threat model of this paper, we assume that compromised nodes do not cooperate and disseminate randomly false positions. If some or all of the compromised nodes cooperate to give false but specious positions, e.g., each compromised node reports a position that is a fixed displacement to its real position, then detecting and filtering these nodes is difficult. In the future, we will evaluate Bilateration and other three localization algorithms under this kind of attack.

References

1. Lymberopoulos, D., Lindsey, Q.: An Empirical Analysis of Radio Signal Strength Variability in IEEE 802.15.4 Networks using Monopole Antennas, ENALAB Technical Report 050501, Yale University (2006)
2. Capkun, S., Hubaux, J.P.: Secure positioning in sensor networks, Technical report (May 2004)
3. Xu, W., Wood, T., Trappe, W., Zhang, Y.: Channel surfing and spatial retreats: defenses against wireless denial of service. In: Proceedings of the 2004 ACM workshop on Wireless security, pp. 80–89 (2004)
4. Hu, Y.C., Perrig, A., Johnson, D.: Packet leashes: a defense against wormhole attacks in wireless networks. In: Proceedings of IEEE Infocom, pp. 1976–1986 (2003)
5. Brands, S., Chaum, D.: Distance-bounding protocols. Theory and Application of Cryptographic Techniques, 344–359 (1993)
6. Sastry, N., Shankar, U., Wagner, D.: Secure Verification of Location Claims. In: Proceedings of WiSe (2003)
7. Waters, B., Felten, E.: Proving the Location of Tamper-Resistant Devices. Technical report, Princeton University
8. Čapkun, S., Buttyǎn, L., Hubaux, J.-P.: SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks. In: Proceedings of SASN (2003)
9. Li, Z., Trappe, W., Zhang, Y., Badri Nath.: Robust statistical methods for securing wireless localization in sensor networks. In: Proceedings of IPSN (2005)
10. Kusy, B., et al.: Node-density independent localization. In: IPSN (2006)
11. Zhou, G., et al.: Impact of Radio Irregularity on Wireless Sensor Networks. In: Proceedings of 2nd MobiSys (2004)

ID-Based Key Agreement with Anonymity for Ad Hoc Networks

Hung-Yu Chien

Department of Information Management, National Chi Nan University, Taiwan, R.O.C.
redfish6@ms45.hinet.net

Abstract. Security support is a must for ad hoc networks. However, existing key agreement schemes for ad hoc networks ignore the issue of entity anonymity. Without anonymity, the adversary can easily identify and track specific entities in the communications. Not only entities' movement information is valuable to the adversary but also the adversary can launch heavy attacks on those important nodes, based on the information. This paper proposes an ID-based n-party ($n \geq 2$) key agreement scheme that preserves entity anonymity from outsiders. The scheme is efficient and very suitable for the structure-free mobile ad hoc networks. The security of the schemes is proved in a modified Bellare-Rogaway model.

Keywords: ad hoc networks, bilinear pairing, identity-based cryptosystem, key agreement, anonymity.

1 Introduction

Ad hoc networks that support self-configurable and autonomous communications are regarded as ideal technologies for creating instant communication networks for civilian and military applications. Depending on the applications and the environments, different ad hoc networks may require different degree of support infrastructure. Asokan and Ginzboorg [13] classified three types of support infrastructures for ad hoc networks. The first type is the routing infrastructure in the form of fixed routers and stable links. The second type is the server infrastructure of on-line servers that provide various services such as name service, directory services, certificate look-up services, etc. The third type is the organizational and administrative support such as registration of users, issuing of certificates, and cross-certification agreements between different user domains. Regarding ad hoc networks, some other features are worth further discussions. First, ad hoc networks are dynamic. It means that nodes in an ad hoc network will move dynamically and some nodes might own poor connectivity with neighbors (or might own rich connectivity for only a short time). So the algorithms designed for ad hoc networks should take these features into account. Secondly, the locations and movements of specific nodes could be valuable information to the adversary. For example, in military applications or some commercial applications, some nodes might play important (or even vital) roles in the communications. One example is the commander in military operations or in

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 333–345, 2007.

crisis management operations, and another example is the server in a business meeting outside. Therefore, exposure of the identities and the locations of nodes could endanger the whole system.

Regarding the security requirements of ad hoc networks, secure key agreement schemes and efficient group key management are two of the most important mechanisms to build a secure network [15]. However, existing key agreement schemes or key management schemes like [10, 12-14, 16-18] for ad hoc networks all ignore the anonymity issue, and many of them assume the on-line certificate servers to support Public Key Infra-structure (PKI) service. Even though it is feasible to support on-line PKI services via distributed mechanism [16-17], the cost to pay is very high which limits their applications, when we consider the dynamic property, the poor connectivity property and the possible resource limitation on these mobile nodes.

Conventionally, the certificate-based public key infrastructure requires an entity to access and verify certificates before using the public keys. It is costly. To get rid of the weaknesses of certificate-based public key infrastructure, Shamir [2] first proposed the first IDentity-based (ID-based) cryptosystem, where an entity's identification is taken as its public key, and, therefore, there is no requirement to securely maintain and verify the public key before using it. An essential requirement of ID-based schemes is that entities' identifications are well known. Fortunately, many ad hoc applications meet this requirement. For example, in military, campus, emergency operations and commercial environments, some kind of identification mechanisms (like social security number, e-mail address, IP address, or codes) have been widely used to uniquely identify the entities. These features make the ID-based cryptosystems very suitable for many ad hoc networks.

In this paper, we focus on the issues of key agreement schemes with anonymity for ad hoc networks; and the issue of secure routing and secure channels for multi-hop link are beyond the scope of this paper; of course, the mechanism proposed in this paper can be used as building blocks for secure routing and secure channel over multi-hop links. We also assume that it is not easy to compromise the entities; therefore, those compromise-prone devices like sensors and RFIDs are excluded in this work. We propose ID-based key agreement schemes with anonymity for ad hoc networks with single Key Generator Center (KGC), and the possible extension for multiple KGCs [21] or for compromise-prone devices is our future work. The benefits of the schemes include: (1) there is no requirement of on-line server support; (2) the schemes preserve anonymity so that an outsider cannot identify or track the communicating parties; (3) they are efficient and adaptive so that they meet the poor connectivity property and the resource-limited property. Regarding the security, we consider the indistinguishability of the session key and the anonymity of the communicating parties in our modified Bellare-Rogaway model that considers the anonymity property and the tripartite case. The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 introduces some definitions useful to understand the design of the schemes. Section 4 presents our two-party key agreement, tripartite key agreement and group key agreement with anonymity. Finally, our conclusions are drawn in Section 5.

Related Works. Related works includes the key agreement schemes for ad hoc networks, the key management schemes for ad hoc networks and the pairing-based key agreement schemes. They are briefly discussed as follows.

One interesting key agreement scheme for ad hoc networks is Asokan-Ginzboorg' *location*-based key agreement scheme [13], where the people physically present in some place know and trust one another physically, but they do not have any a priori means of digitally identifying and authenticating one another, such as shared keys, public key certificate, or on-line trusted servers. Their scheme is so called "location-based key agreement", because only the people locating at the same room (or place) who can see each other can set up a shared password physically and establish the secure communication accordingly.

Contrary to the above special type of ad hoc networks, most ad hoc networks are like those cases where the entities (could be people, devices, and mobile nodes) knowing the identities of other entities instead of "location" want to set up secure communications. Kaya et al.'s multicast scheme [14] attaches joining nodes to the best closest neighbor therefore reducing the cost of request broadcast and reducing the communication and computation cost incurred by the source. The protocol strongly requires the support of on-line certificate authorities, which makes it not suitable for most structure-free ad hoc networks and resource constrained nodes. Rhee et al.'s group key management architecture [10] for MANETs uses the Implicitly Certified Public Keys (ICPK) to eliminate the requirement of on-line server. However, the ICPK exchange for computing a pair-wise key is costly, and the cost of re-keying the group key is $O(\log_2 n)$. Instead of ICPK, Bohio-Miri [12] and Chien-Lin [18], based on ID-based cryptosystem, had proposed the security frameworks for ad hoc networks to get rid of the requirement of on-line servers. However, none of the above schemes considered the anonymity issue.

Our proposed anonymous key agreement schemes are based on ID-based cryptosystems from pairing. However, none of the previous pairing-based key agreement schemes like [1, 9, 11, 18, 20-23] considered the anonymity property, and it seems difficult to achieve the anonymity property by simply extending the previous works, because all the previous key agreement schemes need to exchange entities' identities when they try to establish session keys.

The key management schemes like [16-17], instead of the key agreement issues, focused on the key management issue: how to build the Certificate Authority (CA) [16] service for conventional PKI or the Key Generator Center (KGC) service [17] for ID-based cryptosystem in ad hoc networks. The key management scheme [17] is complementary to our work, and the idea of bootstrapping the KGC can be applied on our schemes for those environments where the entities do not get the public parameters and their private keys from the KGC before the ad hoc network is formed.

2 Preliminaries

We propose our ID-based key agreement schemes from bilinear pairings [6, 8]. In this section, we briefly describe the basic definitions and properties of the bilinear pairing and the assumptions.

2.1 Bilinear Pairing

Let G_1 and G_2 denote two groups of prime order q, where G_1 is an additive group that consists of points on an elliptic curve, and G_2 is a multiplicative group of a finite

field. A bilinear pairing is a computable bilinear map between two groups. Two pairings have been studied for cryptographic use. They are the (modified) Weil pairing $\hat{e}: G_1 \times G_1 \to G_2$ [6] and the (modified) Tate pairing $\hat{t}: G_1 \times G_1 \to G_2$ [8]. For the purposes of this paper, we let e denote a general bilinear map, i.e., $e: G_1 \times G_1 \to G_2$, which can be either the modified Weil pairing or the modified Tate pairing, and has the following three properties:

(1) Bilinear: if $P, Q, R \in G_1$ and $a \in Z_q^*$, $e(P+Q,R) = e(P,R)e(Q,R)$, $e(P,Q+R) = e(P,Q)e(P,R)$, and $e(aP,Q) = e(P,aQ) = e(P,Q)^a$.

(2) Non-degenerate: There exists $P, Q \in G_1$ such that $e(P,Q) \neq 1$.

(3) Computable: There exist efficient algorithms to compute $e(P,Q)$ for all $P, Q \in G_1$.

Definition 1. The bilinear Diffie-Hellman problem (BDHP) for a bilinear pairing $e: G_1 \times G_1 \to G_2$ is defined as follows: Given $P, aP, bP, cP \in G_1$, where a, b, c are random numbers from Z_q^*, compute $e(P,P)^{abc} \in G_2$.

Definition 2. The computational Diffie-Hellman problem (CDHP) is defined as follows: Given $P, aP, bP \in G_1$, where a and b are random numbers form Z_q^*, compute $abP \in G_1$.

Definition 3. The decision bilinear Diffie-Hellman problem (DBDH) for a bilinear pairing $e: G_1 \times G_1 \to G_2$ is defined as follows: Define two probability distributions of tuples of seven elements, $Q_0 = \{\langle G_1, G_2, P, aP, bP, cP, e(P,P)^{abc} \rangle : a,b,c \in_R Z_q\}$ and $Q_1 = \{\langle G_1, G_2, P, aP, bP, cP, e(P,P)^{d} \rangle : a,b,c,d \in_R Z_q\}$. Then, given the tuple $\langle G_1, G_2, P, P_A, P_B, P_C, K \rangle$, decide whether the tuple is from Q_0 or from Q_1.

Definition 4. The Inverse Computational Diffie-Hellman Problem (Inv-CDHP): given P, aP, to compute $a^{-1}P$.

In order to prove the security of our schemes, we define a new problem and prove it is equivalent to other hard problems as follows. To our best knowledge, we do not know there is any formulation of the BoIDHP before, and we refer it the name BoIDHP to differentiate it from the conventional BDHP.

Definition 5. The Bilinear one Inverse Diffie-Hellman Problem (BoIDHP): given bilinear pairing $e: G_1 \times G_1 \to G_2$, $P, aP, bP, cP \in G_1$, where a, b, c are random numbers from Z_q^*, compute $e(P,P)^{abc^{-1}} \in G_2$.

CDHP, BDHP, DBDH, Inv-CDHP assumptions: It is commonly believed that there is no polynomial time algorithm to solve BDHP, CDHP, Inv-CDHP or DBDH with non-negligible probability [1, 6, 7, 19].

Theorem 1. BoICDHP and BDHP are polynomial time equivalent.

Proof: we give a simple proof as follows.
(i) we first prove BDHP $\Rightarrow$ BoIDHP. Given P, aP, bP, cP, we set the input of BDHP as follows. $Q = cP, Q_1 = aP = ac^{-1}Q, Q_2 = bP = bc^{-1}Q, Q_3 = P = c^{-1}Q$, then BDHP outputs $e(Q,Q)^{ac^{-1}bc^{-1}c^{-1}} = e(P,P)^{abc^{-1}}$.
(ii) BoIDHP $\Rightarrow$ BDHP. Given P, aP, bP, cP, we set the input of BoIDHP as follows. $Q = cP, Q_1 = aP = ac^{-1}Q, Q_2 = bP = bc^{-1}Q, Q_3 = P = c^{-1}Q$, then BoIDHP outputs $e(Q,Q)^{ac^{-1}bc^{-1}c} = e(P,P)^{abc}$. □

2.2 Parameters for ID-Based Cryptosystems from Pairing

Let G_1 and G_2 denote two groups of prime order q, where G_1 is a group on the elliptic curves. q is a prime which is large enough to make solving discrete logarithm problem in G_1 and G_2 infeasible. Let P is a generator of G_1, and the *MapToPoint* function [6] encodes the identity of a user to a point in the group G_1. Let us denote such a function as H_1 which takes an input *ID* of any length and outputs a point in the group G_1. The output point is taken as the entity's public key. That is, $Q_A = H_1(ID_A)$ is the public key of entity A with identity ID_A. Let e be a bilinear paring as defined above.

Initially, the key generation center (KGC), which is also a Trusted authority (TA), selects the system parameters $\{G_1, G_2, e, q, P, H_1\}$, chooses a random secret $s \in_R Z_q^*$ as its secret key, computes his public key $P_{KGC} = s \cdot P$ and finally publishes $\{G_1, G_2, e, q, P, H_1, P_{KGC}\}$. For each registered user A with his identity ID_A, his public key is given by $Q_A = H_1(ID_A)$ and the private key is $S_A = s \cdot Q_A$ which is sent by the KGC to the user via a secure channel.

3 Anonymous Key Agreement Schemes

Now we describe our key agreement schemes that consists of two-party key agreement, tripartite key agreement, and group key agreement. In the following, we assume that all entities are properly set up before the ad hoc network is formed. If this assumption does not stand for some applications, the idea of bootstrap the KGC [17] can be applied. In an ID-based scheme, all entities being properly set up mean that a unique identification mechanism is well known among the entities, and these entities get the public parameters and their private keys from the KGC before the ad hoc network is formed. That is, an entity A has got the public parameters and his private key $S_A = s \cdot Q_A$ from the KGC. In addition, in our schemes, all the registered entities get one additional secret from the KGC, the group secret $S_G = \frac{1}{s}P$. In the rest of this paper, both the term node and the term entity denote one mobile node. We also

differentiate the entities in ad hoc networks into two kinds: *group members* denote those entities that have shared a well known identification mechanism and are authorized to join the ad hoc networks, and *group outsiders* denote those entities that may be eavesdroppers or adversaries and are not allowed to join the ad hoc networks. Our proposed schemes satisfy the anonymity against any *active group outsider* and against *passive group members* (who are not the partners of the sessions).

Now we summarize notations used in this paper as follows:

U_i / ID_i : U_i is the *i*th node with identity ID_i .

s / P_{KGC} : s is the secret key of the KGC, and $P_{KGC} = sP$ is KGC's public key.

$S_G = \frac{1}{s}P$: the group secret that is shared among all the registered entities.

Q_{ID_i} / S_i : The public key of *node i* is $Q_{ID_i} = H_1(ID_i)$, and the private key is $S_i = sQ_{ID_i}$.

$Sig_A(m)$: node *A*'s signature on message *m*. Here, we suggest the use of Hess's ID-based signature [7], because it is efficient (it requires only one pairing operation) and has been proven secure in the random oracle model.

$E_k(m)$: the symmetric key encryption using key k . The scheme should satisfy the indistinguishability under chosen plain text attack (IND-CPA) property.

D_{AB} : The pair-wise secret of *node A* and node *B*.

$H_1() / H_2() / H_3()$: $H_1 : \{0,1\}^* \to G_1$ is the *MaptoPoint* function [6]; a one-way hash function $H_2 : G_2 \to \{0,1\}^t$, where t is the bit length of the key for symmetric encryption; a hash function $H_3 : \{0,1\}^* \to \{0,1\}^q$. The hash functions are modeled as random oracles in the security proofs.

3.1 Static Pair-Wise Key Agreement

Initially, each registered node *A* receives its private key $S_A = sQ_A = sH_1(ID_A)$ **,** where $Q_A = H_1(ID_A)$ is the public key. Now *A* computes the shared secret $D_{AB} = e(S_A, Q_B) = e(Q_A, Q_B)^s$ with *B*. *B* computes the shared secret $D_{BA} = e(Q_A, S_B) = e(Q_A, Q_B)^s$. Finally, the shared symmetric secret key is $K = H_2(D_{AB}) = H_2(D_{BA})$ which will be used to encrypt the communications between *A* and *B*. Note that any two nodes can generate this static key without any interaction.

3.2 Dynamic Pair-Wise Key Agreement

To further provide dynamic pair-wise session key, we propose a new two-party key agreement with anonymity as follows. Assume *A* and *B* are close to one another, and they can detect the existence of each other (for example, by broadcasting a special format beacon like that in Aloha network) and want to establish an authenticated session key without disclosing their identities to outsiders. In the following, *sid*

denotes the session identifier that can uniquely identify one session from others, and $A \Rightarrow$ all denotes A broadcasts its messages to its neighbors.

1. $A \Rightarrow$ all: sid, $P_A = xP_{KGC}$

   A first chooses a random integer $x \in Z_q^*$, computes and sends $P_A = xP_{KGC}$ to B.

2. $B \Rightarrow$ all: sid, $P_B = yP_{KGC}$, $E_{k_1}(ID_B \| Sig_B(H_3(sid \| ID_B \| P_A \| P_B)))$

   B chooses a random integer $y \in Z_q^*$, computes $D_{AB} = e(P_A, S_G)^y = e(xP_{KGC}, 1/sP)^y = e(P,P)^{xy}$, $k_1 = H_2(D_{AB})$ and $Sig_B(H_3(m))$, where $m = sid \| ID_B \| P_A \| P_B$. Then B use k_1 as the encrypting key to encrypt the data $ID_B \| Sig_B(H_3(m))$.

3. $A \Rightarrow$ all: sid, $E_{k_1}(ID_A \| ID_B \| Sig_A(H_3(sid \| ID_A \| ID_B \| P_A \| P_B)))$

   Upon receiving the data in Step 2, A first computes $D_{AB} = e(P_B, S_G)^x = e(P,P)^{xy}$ and $k_1 = H_2(D_{AB})$, and then uses k_1 to decrypt the second part of the data to derive $ID_B \| Sig_B(...)$. Now A learns the identity ID_B of its communicating party, and verifies whether the signature $Sig_B(...)$ is valid. If the verification succeeds, then it generates its signature on the data $m = sid \| ID_A \| ID_B \| P_A \| P_B$ as $Sig_A(H_3(m))$, and sends $E_{k_1}(ID_A \| ID_B \| Sig_A(H_3(m)))$ to B. The final session key K_{sess} is computed as $K_{sess} = H_2(k_1 \| sid \| ID_A \| ID_B)$.

   Upon receiving the response $E_{K_1}(ID_A \| ID_B \| Sig_A(H_3(m)))$ from A, B first uses k_1 to decrypt the data and gets $ID_A \| ID_B \| Sig_A(...)$. Now B learns the identity, ID_A, and can verify whether the signature is valid. If the verification succeeds, then B accepts the message, and computes the final session key $K_{sess} = H_2(k_1 \| sid \| ID_A \| ID_B)$.

3.3 Tripartite Key Agreement with Anonymity

We now describe our tripartite key agreement which can be used to set up secure communication among three entities and can be used as a primitive for set up the group key for group broadcasting.

Assume *A, B, and C* are three nodes that detect the existence of each other, and want to establish session keys among them. They can perform the following tripartite key agreement protocol to establish the session key without disclosing their identities to outsiders. Our protocol consists of two rounds where the entities broadcast their ephemeral public keys in the first run and the entities broadcast their encryption on signatures and the identity in the second round. The protocol is described as follows.

Round 1:

1.1. $A \Rightarrow$ all: sid, $P_A = aP_{KGC}$

1.2. $B \Rightarrow$ all: sid, $P_B = bP_{KGC}$

1.3. $C \Rightarrow$ all: sid, $P_C = cP_{KGC}$

A computes $P_A = aP_{KGC}$, where a is a random number chosen by A. A broadcasts (sid, P_A). Likewise, B/C respectively chooses a random integer b/c, computes and broadcasts the ephemeral public keys P_B / P_C respectively.

Round 2:

2.1. $A \Rightarrow$ all: sid, $E_{k_1}(ID_A \| Sig_A(H_3(m_A)))$

2.2. $B \Rightarrow$ all: sid, $E_{k_1}(ID_B \| Sig_B(H_3(m_B)))$

2.3. $C \Rightarrow$ all: sid, $E_{k_1}(ID_C \| Sig_C(H_3(m_C))$

Upon receiving the broadcast data in Step 1, A first computes $k_1 = H_2(sid \| e(P_B, S_G) \cdot e(P_C, S_G) \cdot e(P,P)^a \cdot e(P_B, P_C)^a) = H_2(sid \| e(P,P)^{a+b+c+abcs^2})$ and generates its signature $Sig_A(H_3(m_A))$, where $m_A = sid \| ID_A \| P_A \| P_B \| P_C$. Likewise, B/C respectively computes $k_1 = H_2(sid \| e(P_C, S_G) \cdot e(P_A, S_G) \cdot e(P,P)^b \cdot e(P_C, P_A)^b) = H_2(e(P,P)^{a+b+c+abcs^2})$ / $k_1 = H_2(sid \| e(P_A, S_G) \cdot e(P_B, S_G) \cdot e(P,P)^c \cdot e(P_B, P_A)^c) = H_2(e(P,P)^{a+b+c+abcs^2})$ and generates the signature $Sig_B(H_3(m_B))$ / $Sig_C(H_3(m_C))$, where $m_B = sid \| ID_B \| P_A \| P_B \| P_C$ and $m_C = sid \| ID_C \| P_A \| P_B \| P_C$. The final session key $K_{sess} = H_2(sid \| e(P,P)^{a+b+c+abcs^2} \| ID_A \| ID_B \| ID_C)$.

The proposed tripartite scheme is secure in terms of in-distinguishability and resistance to both the key-compromise impersonation attack and the insider attack against an actively attacker (except the TA) in a modified Bellare-Rogaway model.

3.4 Group Key Management

To derive the group key, we propose to build up the group key by dividing the group into a ternary tree with all the entities at the leaves, and iteratively run the tripartite key agreement protocol or the two-party key agreement, depending on the down-degree of the current parent node, from bottom to top to get the group key. For each derived secret k after applying the key agreement protocol at level i, the value kP_{KGC} will be used as the ephemeral public value for the key agreement protocol at the (i-1)th level. Also the node with the smallest identity in each subgroup will represent the subgroup to participate the (i-1)th level key agreement. The final derived key for the root node is the final group key for the whole group.

Take Figure 1 as an example. Entities 1~8 are arranged in the leaves, and the intermediate nodes represent the sub-groups covering the entities under the nodes. The root node represents the final group key. Initially, all leaves at level 3 respectively involve the protocol instances of their subgroups. Nodes 1, 2, 3 launch the tripartite key agreement to derive the subgroup key, say $k_{1,2,3}$. Nodes 4, 5, 6 involve in another instance to derive the subgroup key, say $k_{4,5,6}$. Node 7 and 8 initiate an instance of two-party key agreement protocol to derive the subgroup key, say $k_{7,8}$. At level 2, Node 1, 4, 7 respectively represents their subgroups to initiate the tripartite key agreement protocol for level 2. In this protocol instance, Node 1 uses

$k_{1,2,3}P_{KGC}$ as its ephemeral public value, Node 4 uses $k_{4,5,6}P_{KGC}$ as its ephemeral public value, and Node 7 uses $k_{7,8}P_{KGC}$ as its ephemeral public value. After this protocol instance, the group key corresponding to Node 12 is $K_{1\sim8} = H_2(sid \| e(P,P)^{k_{1,2,3}+k_{4,5,6}+k_{7,8}+k_{1,2,3}\cdot k_{4,5,6}\cdot k_{7,8}\cdot s^2} \| ID_9 \| ID_{10} \| ID_{11})$. Since each leaf in the tree knows exactly one secret of ($k_{1,2,3}$, $k_{4,5,6}$, $k_{7,8}$), all the leaves can derive the group key $K_{1\sim8}$.

To dynamically adapt to the membership change in ad hoc networks, the ternary tree is updated accordingly and the keys on the path from the lowest updated node to the root are refreshed, using the key agreement protocols. The computational complexity of this approach is $O(\log_3 n)$, which is more efficient than its counterparts [10] whose complexity is $O(\log_2 n)$. The security of the group key management is directly based on that of the two-party key agreement and that of the tripartite key agreement. Since both the two-party key agreement and the tripartite key agreement are secure, the group key agreement is secure.

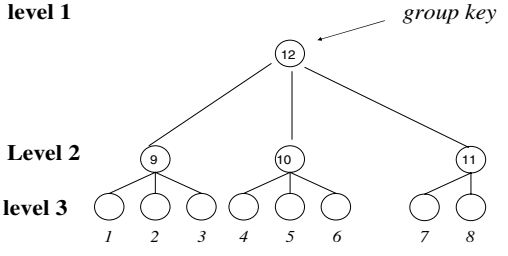


Fig. 1. Bottom-up, divide-and-conquer to derive the group key

Table 1 summarizes the comparisons of our proposed schemes with its counterparts. Asokan-Ginzboorg's key agreement scheme, and the key management schemes [16, 17] and Kaya et al.'s multicast scheme [14] are not listed in the comparisons, because Asokan-Ginzboorg's location-based key agreement schemes are for special ad hoc networks, and Kaya et al. scheme focused only on group management that attaches joining node to the closest neighbor. The proposed schemes, Chien-Lin's scheme [18] and Rhee et al.'s scheme require no on-line server support, which makes them more suitable for ad hoc networks. Also the three schemes provide formal proofs of the protocols, but Bohio-Miri's scheme has security flaws. Our scheme and Chien-Lin's scheme [18] provide efficient static pair-wise key agreement, efficient dynamic two-party key agreement and efficient tripartite key agreement, while Rhee et al.'s scheme only supports their costly two-party key agreement protocol. While Rhee et al.'s two-party key agreement protocol requires 5 message runs, our scheme requires only two message runs. Finally, only the proposed scheme here provides entity anonymity.

Table 1. Comparisons among secure schemes for ad hoc networks

	Rhee [10]	Bohio-Miri [12]	Chien-Lin [18]	The proposed scheme
Types of cryptosystems	ICPK*	ID-based, certificate-based	ID-based	ID-based
On-line server support	No	Yes	No	No
Security property	Formal proof	Security flaws (forgery problem)	Formal proof	Formal proof
Static pair-wise key	No	Yes	Yes	Yes
Cost of dynamic two-party key agreement	5 message runs, 5 T_E for one entity	No dynamic key agreement provided	2 runs, $2T_P+1T_M+1T_{Scalar}$ for one entity	** 3 runs, $2T_P+1T_M+1T_{Scalar}+2T_{ENC}$ for one entity
Efficient tripartite key agreement	No	No	Yes	Yes
Complexity of group key management	Group key agreement in $O(\log_2 n)$	The group key is chosen by the group leader.	Group key agreement in $O(\log_3 n)$	Group key agreement in $O(\log_3 n)$
Entity anonymity	No	No	No	Yes

* ICPK: Implicitly Certified Public keys.

** T_E denotes the cost of one modular exponentiation, T_{ENC} denote the cost of one symmetric encryption, T_P denotes that of one pairing operation, T_M denotes that of one modular multiplication, T_{scalar} denotes that of one scalar multiplication in G_1. Here assume that Hess's signature scheme is used to generate the signature.

4 Conclusions and Future Work

This paper has discussed the infra-structure support property, the poor connectivity property, the anonymity property and the possible resource-limited property of mobile ad hoc networks. Based on ID-based cryptosystem from pairings, we have proposed our key agreement protocols with anonymity, and have proved the security in our model. The benefits of our proposed schemes include: (1) there is no requirement of on-line server support, (2) the protocols are efficient, and (3) the protocols preserve the entities' anonymity. These features make them very attractive to mobile ad hoc networks. As low-cost mobile devices become more and more popular, it is interesting to extend the results to those compromise-prone devices and to those resource-limited devices where public key cryptography is not feasible.

Acknowledgements. We sincerely appreciate the anonymous reviewers for providing us the valuable comments. This research is partially supported by the National Science Council, Taiwan, R.O.C., under contract no NSC 95-2221-E-260 -050 -MY2.

References

1. Joux, A.: A One Round Protocol for Tripartite Diffie-Hellman. In: Bosma, W. (ed.) Algorithmic Number Theory. LNCS, vol. 1838, pp. 385–394. Springer, Heidelberg (2000)
2. Shamir, A.: Identity Based on Cryptosystems and Signature Schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
3. Bellare, M., Rogaway, P.: Provably Secure Session Key Distribution: The Three Party Case. In: 27th ACM Symposium on the Theory of Computing, pp. 57–66. ACM Press, New York (1995)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
5. Canetti, R., Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 451–472. Springer, Heidelberg (2001)
6. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
7. Hess, F.: Efficient Identity Based Signature Schemes Based on Pairings. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 310–324. Springer, Heidelberg (2003)
8. Frey, G., Muller, M., Ruck, H.: The Tate Pairing and the Discrete Logarithm Applied to Elliptic Curve Cryptosystem. IEEE Trans. on I.T. 45(5), 1717–1719 (1999)
9. Chien, H.Y.: ID-Based Tripartite Multiple Key Agreement Protocol Facilitating Computer Auditing and Transaction Refereeing. Journal of Information Management 13(4), 185–204 (2006)
10. Rhee, K.H., Park, Y.H., Tsudik, G.: A Group Key Management Architecture for Mobile Ad-hoc Wireless Networks. Journal of Information Science and Engineering 21, 415–428 (2005)
11. Chen, L., Kudla, C.: Identity Based Authenticated Key Agreement Protocols from Pairings. Cryptology ePrint Archive, Report 2002/184 (2002)
12. Bohio, M., Miri, A.: Efficient Identity-Based Security Schemes for Ad Hoc Network Routing Protocols. Ad Hoc Networks 3, 309–317 (2004)
13. Asokan, N., Ginzboorg, P.: Key Agreement in Ad Hoc Networks. Computer Communications 23, 1627–1637 (2000)
14. Kaya, T., Lin, G., Noubir, G., Yilmaz, A.: Secure Multicast Groups on Ad Hoc Networks. In: Proc. of the 1st ACM Workshop Security of Ad Hoc and Sensor Networks, pp. 94–102 (2003)
15. Varadharajan, V., Shankaran, R., Hitchens, M.: Security for Cluster Based Ad Hoc Networks. Computer Communications 27, 488–501 (2004)
16. Zhu, B., Bao, F., Deng, R.H., Kankanhalli, M.S., Wang, G.: Efficient and Robust Key Management for Large Mobile Ad Hoc Networks. Computer Networks 48, 657–682 (2005)
17. Khalili, A., Katz, J., Arbaugh, W.A.: Toward Secure Key Distribution in Truly Ad-hoc Networks. In: Proc. of the 2003 Symp. on Applications and the Internet Workshop (2003)
18. Chien, H.Y., Lin, R.Y.: Improved ID-Based Security Framework for Ad-hoc Networks. Ad Hoc Networks 6(1), 47–60 (2008)
19. Sadeghi, A.R., Steiner, M.: Assumptions Related to Discrete Logarithms: Why Subtleties make a Difference. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 243–260. Springer, Heidelberg (2001)
20. Smart, N.P.: An Identity Based Authenticated Key Agreement Protocol Based on the Weil Pairing. Electronics Letters 38, 630–632 (2002)
21. Chen, L., Harrison, K., Soldera, D., Smart, N.: Applications of Multiple Trust Authorities in Pairing Based Cryptosystems. HP Journal (Feburary 2003)

22. McCullagh, N., Barreto, P.: A New Two-Party Identity-Based Authenticated Key Agreement. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 262–274. Springer, Heidelberg (2005)
23. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems Based on Pairing. In: the 2000 Symp. On Cryptography and Security (SCIS 2000), Japan, 26-28 (2000)

Appendix. Security Notations and Proofs

The security of the proposed schemes concerns both the privacy of the authenticated session key and the privacy of the identities of the communicating parties. To capture the security of the tripartite key agreement scheme, we consider the in-distinguishability property [3-5], and the resistance to key-compromise impersonation, *known-key attack, forward secrecy* and the insider attack. We, therefore, prove the in-distinguishability in a modified model. Regarding the in-distinguishability, we adopt the BPR2000 model with some modifications- (1) extension to the tripartite case, (2) extension for the Corrupt query, and (3) adaptations to the identity anonymity.

The in-distinguishability of the proposed tripartite key agreement
Since the protocols hide the identities in the communications, in the model, the adversary AD cannot *fully* control the communications (*fully controls the partnership relation*) and, instead, *partially* controls the communications that take place between parties by interacting with a set of $\Pi^i_{U_1,*,*}$ oracles ($\Pi^i_{U_1,*,*}$ denotes that U_1 does not know its partners so far). In our protocol with anonymity, the adversary does not know the identities and the possible matching among oracles. We, therefore, try to model this situation by the following adaptations: (1) the adversary is allowed to send queries to specific oracle instances, but it does not know the partners of the oracle; (2) the challenger (or the simulator) randomly determines the matching among the instantiated oracles, and keeps the matching consistent through all the sessions but keeps the information secret from the adversary. The pre-defined oracle queries include - Hash query**,** Send(U_1, *, *, *i*, *m*), Reveal(U_1, *, *, *i*) , Corrupt(U_1), Test(U_1, *, *, *i*)**,** Sign (U_1, *i*, *m*). Note that, after an oracle has accepted, it knows the identities of its partners.

Security in the model is defined using the game *G*, played between the adversary and a collections of $\Pi^i_{U_x,*,*}$ oracles for players $U_x \in \{U_1, U_2, ..., U_{N_P}\}$ and instances $i \in \{1,...,l\}$. The adversary AD runs the game simulation *G* with setting as follows (we let the simulation *G* randomly determines the matching relationship among oracles, keeps it consistent through the simulation and keeps it secret from AD).

Stage 1: AD is able to send *Hash*, *Sign*, *Send*, *Reveal*, and *Corrupt* queries in the simulation.

Stage 2: At some point during *G*, AD will choose a fresh session and send a *Test* query to the fresh oracle associated with the test session. Depending on the randomly chosen bit *b*, AD is given either the actual session key or a session key drawn from the session key distribution.

Stage 3: AD continues making any *Hash, Sign, Send, Reveal* and *Corrupt* oracle queries to its choice.

Stage 4: Eventually, AD terminates the game simulation and output its guess bit *b'*.

Success of AD in G is measured in terms of AD's advantage in distinguishing whether AD receives the real key or a random value. Let the advantage function of AD be denoted by $Adv^{AD}(k)$, where k is the security parameter and $Adv^{AD}(k) = 2\Pr[b=b'] - 1$.

Definition 6 (Secure tripartite key agreement protocol). A tripartite key agreement protocol is secure in our model if the following thee requirements are satisfied:

Validity: When the protocol is run among three oracles in the absence of an active adversary, the three oracles accept the same key.

Indistinguishability: For all probabilistic, polynomial-time adversaries AD, $Adv^{AD}(k)$ is negligible.

Security against insider impersonation and key-compromise impersonation: Even an insider (and a key-compromise impersonator) cannot impersonate another entity to the third entity and complete the session run with the third one.

Theorem 2. The proposed tripartite key agreement protocol is secure in the sense of Definition 6 if the underlying digital signature scheme is secure against the adaptively chosen message attack and the DBDH is hard.

Definition 7. We say that a tripartite key agreement scheme satisfies the entities anonymity if no probability polynomial time (PPT) distinguisher has a non-negligible advantage in the following game.

1. The challenger sets the system parameters (which might includes the group secret), and determines the private key/ public key pair, S_{ID_i} / Q_{ID_i}, for each $U_i \in \{U_1,...,U_{N_P}\}$. It hands the public parameters to the distinguisher *D*.

2. *D* adaptively queries the oracles defined in Appendix.

3. Once stage 2 is over, the challenger randomly chooses $b_1, b_2, b_3 \in \{1,...,N_P\}$ such that b_1, b_2, and b_3 are all different. The challenger lets U_{b_1}, U_{b_2}, and U_{b_3} be the three entities running a matching session, faithfully follows the protocol specification to generate the communication transcripts *trans** among the three oracles such that they follows the order (U_{b_1}, U_{b_2}, U_{b_3}) in generating their first round messages. It finally hands *trans** to *D*.

4. *D* adaptively queries the oracles as in stage 2 with the restriction that, this time, it is disallowed to send Reveal queries to the three target oracles in stage 3.

5. At the end of the game, *D* outputs $\overline{b}_1, \overline{b}_2, \overline{b}_3 \in \{1,...,N_P\}$ and wins if $\overline{b}_1 = b_1$ or $\overline{b}_2 = b_2$ or $\overline{b}_3 = b_3$. Its advantage is defined to be

$$Adv^{anonymity}_{tripartite}(D) := \Pr[\overline{b}_1 = b_1 \text{ or } \overline{b}_2 = b_2 \text{ or } \overline{b}_3 = b_3] - 3/N_P$$

Likewise, similar definitions can be defined and similar theorems can de derived for the two-party case. Due to page limitation, the detailed definitions and the proofs are omitted in this version.

Buffer Cache Level Encryption for Embedded Secure Operating System*

Jaeheung Lee[1], Junyoung Heo[1], Jaemin Park[1], Yookun Cho[1], Jiman Hong[2], and Minkyu Park[3,**]

[1] Seoul National University
{jhlee,jyheo,jmpark,cho}@ssrnet.snu.ac.kr
[2] Soongsil University
jiman@ssu.ac.kr
[3] Konkuk University
minkyup@kku.ac.kr

Abstract. A cryptographic file system is the representative way of assuring confidentiality of files in operating systems. For secure embedded operating systems, the cryptographic file system could be a practical technique. In general, cryptographic file systems are implemented using a stackable file system or a device driver. These two mechanisms can provide user transparent encryption/decryption of cryptographic file systems. But these mechanisms sometimes encrypt or decrypt data redundantly or unnecessarily. Embedded systems with a low speed CPU and flash storage are more affected by the problems than general systems. We addressed the above mentioned problems by applying an encryption algorithm on buffer caches and enabling one buffer cache to have both encrypted and decrypted data together. Experimental results show that the proposed mechanisms reduce the redundant or unnecessary operations and it can improve the performance of cryptographic file systems.

Keywords: Security, Cryptographic File System, Embedded Operating System, Buffer Cache, Linux.

1 Introduction

Secure data technique is more attractive as the value of information increases and threats to the information increase[1]. Cryptographic file systems are the most practical technique for secure operating systems. Even though the files are stolen by physical or network attacks, it can protect files that contain the valuable information by encrypting the information. Attackers cannot get secure information of files unless they get the key of the cryptographic algorithm.

* This research was supported in part by the Brain Korea 21 project and MIC & IITA through IT Leading R&D Support Project. The ICT at Seoul National University provides research facilities for this study.

** Corresponding author.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 346–355, 2007.

Considering the growing interest in mobile embedded systems, the use of cryptographic file systems in embedded systems will be more important. The securing data in mobile embedded systems is more important than existing general systems because the mobility causes a new threat, that is the lost of the system itself[2].

The performance of a file system is sure to decrease after applying a cryptographic file system. Because cryptographic algorithms require a lot of CPU instruction for encryption and decryption. From the user's point of view, the degradation of the performance must be tolerable. So the reasonable performance is required for cryptographic file systems. Especially, embedded systems are more affected by the degradation of the performance than other systems because they are composed of a low speed CPU, a small memory and batteries.

So far, many cryptographic file systems have been introduced. In this study, we consider cryptographic file systems only in kernel-level [3,4,5,6,7,8,9]. These are categorized by the implementation method: using a stackable file system and using a device driver.

A stackable file system is an easy and efficient way to add new features into existing file systems of a kernel[10,7]. With a stackable file system, you can easily add encryption or compression algorithms into a file system. Kernel modification is a very hard job due to the complexity of a kernel and the difficulty in debugging it. The stackable file system allows a new feature to be integrated into a kernel without influencing another part of a kernel. In addition, it is faster than user-level library because the stackable layer is in a kernel. Therefore, many cryptographic file systems using a stackable file system have been developed.

However, the stackable file system cannot exploit the advantage of buffer caches[11]. If a user process accesses some part of a file twice, encryption or decryption should be repeated twice in a stackable layer. Therefore, redundant encryption or decryption may occur in a cryptographic file system using a stackable file system.

Another popular method to implement cryptographic file systems is a device driver[3,4,5,6]. Strictly speaking, there is no relation between using a device driver and the file systems. Because it locates a cryptographic algorithm in a device driver, a file system does not know the existence of a cryptographic algorithm. As a result, all data read from a disk is always decrypted before being stored in buffer caches. Inversely, all data stored in buffer caches is always encrypted before being stored in a disk. Therefore, unnecessary encryption or decryption may occur in a device driver.

To improve the performance by reducing those redundant and unnecessary encryption/decryption, we propose the mechanism, support in a buffer cache level. Basically, our mechanism locates a cryptographic algorithm in buffer caches and enables a buffer cache to have mixed plaintext and ciphertext.

In the request of a read operation from the VFS(or user application), encrypted data is decrypted before copying the data from a buffer cache to the VFS. The amount of decryption is strictly limited to the size of requested data. If data in a buffer cache is decrypted once, the decrypted data is replaced with

an encrypted one. If the request of a read operation at the same position in a file occurs again, no more decryption is required. This is helpful to remove redundant encryption/decryption of a cryptographic file system using a stackable file system.

By limiting the amount of decryption to the size of requested data, the problem of unnecessary decryption is resolved. To this end, we designed a new buffer cache with a bitmap to indicate whether data is encrypted or decrypted. Therefore, a buffer cache of our mechanism is able to have plaintext and ciphertext together.

The rest of this paper is organized as follows. Section 2 surveys related works. Section 3 describes our mechanism. Section 4 presents our experimentation and results. Section 5 presents our concluding remarks.

2 Related Work

We survey some related work in this section. There are many techniques related to securing files: encryption of a storage device, encryption using a user-level library, encryption in a device driver and encryption using a stackable file system.

Encryption of a storage device can be operated without operating systems. It is integrated with a storage hardware. Therefore, its performance is better than that of other software techniques. DataTravler Elite [12] and SecureIDE [13] fall under this category. Encryption using a user-level library such as crypt(3) and GNU PG[14] are also available. CFS[15] and TCFS[16] are examples of user-level mechanisms. They use a NFS server for applying cryptographic algorithms. These user-level mechanisms can be easily implemented, but it has many problems with respect to key management, consistency, performance and so on. We will not mention these techniques any more because our study focuses on the kernel mechanism.

We focus on two kernel mechanism: encryption in a device driver and a stackable file system. Strictly speaking, encryption in a device driver is not a cryptographic file system because a cryptographic file system is a file system to manage files on a disk securely. However, to simplify the terms, we will refer to both of them as a kind of cryptographic file systems.

In a cryptographic file system using a device driver, encryption is carried out while an I/O operation is being performed. It can be used for the applications that accesses to a storage directly such as a swap device and a database. It can also be used for the general applications that require file systems. It can provide user transparency and good performance, but it cannot encrypt or decrypt a unit of a file. Cryptoloop [3], CryptoGraphic Disk Driver (CGD) [4], SFS [6] and BestCrypt [5] fall into this category.

A cryptographic file system using a device driver outperforms the stackable mechanism. However, the stackable mechanism can provide file encryption. It also provides user transparency like the device driver mechanism.

Cryptfs [7] and Ncryptfs [8] use FiST [17] as a stackable file system. Ncryptfs is an improved version of Cryptfs. Ncryptfs can authenticate several users

simultaneously and apply cryptographic algorithms dynamically. It also provides challenge-response authentication. It uses block cipher and applies CFB (Cipher FeedBack) mode for inode blocks, ECB (Electronic CodeBook) for data blocks.

EFS (Encryption File System) is a cryptographic file system based on MS Windows NT[9]. It exists in a kernel, but it requires user DLL for encryption and user authentication.

3 Buffer Cache Level Support

The main objective of our mechanism is to exploit the advantage of buffer caches. In addition, our mechanism enables a buffer cache to be encrypted or decrypted partially. In this study, we use a block cipher as a cryptographic algorithm and an ECB(Electronic CodeBook) mode as a block cipher mode. With this combination, random access and equal length of plaintext and ciphertext can be achieved.

Fig. 1 shows how decryption is performed in a read operation. When a user requests an operation that reads data in the block 2 of a file, the block is loaded to a buffer cache and copied to a user area. In the system using a stackable file system, Fig. 1(a), decryption is performed right before copying data from a buffer cache to a user area. If another read operation that requests data which is already requested in a previous read operation is performed, decryption will be repeated. Such redundant operation can occur because a stackable file system does not care whether a buffer cache exists or not.

In case of using a device driver, Fig. 1(b), decryption is performed right before loading data from a file to a buffer cache. Buffer caches always have decrypted data. Therefore, redundant decryption does not occur. However, decryption in a device driver level may cause unnecessary decryption. As a device driver does

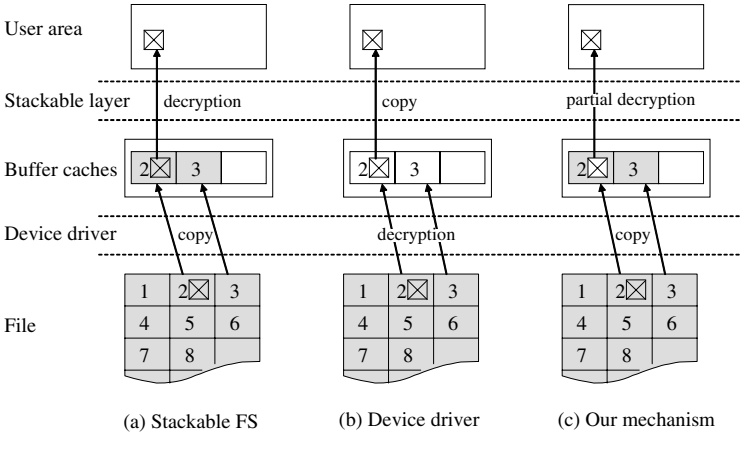


Fig. 1. Decryption in Read Operation

not have any information about the read operation, the entire block 2 is read and decrypted even though the part of the block 2 is required.

Another case of unnecessary decryption in a device driver level may occur because of a read-ahead technique. Operating systems sometimes read the next data of current read in advance. A read-ahead assumes that disk accesses are sequential[18]. However, a read-ahead algorithm does not always succeed as predicted.

In our mechanism, Fig. 1(c), the block 2 of a file is copied to buffer caches but not decrypted. Only the requested area of the block 2 is decrypted and stored back in buffer caches. After this partial decryption, the requested data is copied to a user area. Because unused area of the block 2 is not decrypted, this partial decryption can remove the unnecessary decryption. We can also expect that the redundant decryption will be removed by the buffer caches because the decrypted result is stored in buffer caches.

In a write operation, redundant encryption can occur as in a read operation. We consider only overwrite operations that write data in the existing part of a file. In case of a append operation, three mechanisms are similar. Fig. 2 shows the case of a write operation. In a repeated write operation, the stackable layer encrypts the data redundantly. In case of using a device driver and our mechanism, this redundancy does not occur.

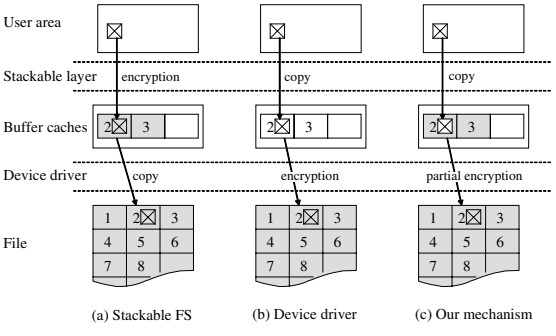


Fig. 2. Encryption in Write Operation

We add new information in a buffer cache to support partial encryption/decryption of a buffer cache. This information enables a buffer cache to distinguish which part of a buffer cache is encrypted or decrypted. The information, BITMAP is shown in Fig. 3. The DATA in a buffer cache is divided into cipher blocks. If the bit is 1, the corresponding cipher block is encrypted. Otherwise, the cipher block is decrypted. Block cipher algorithms use a cipher block as a basic encryption or decryption unit. In case of AES, the size of a cipher block is 16 bytes. If the size of a buffer cache(DATA) is 4096 bytes, the size of BITMAP is 32 bytes. That is, additional 32 bytes per each buffer cache is necessary when AES is used as a cryptographic algorithm.

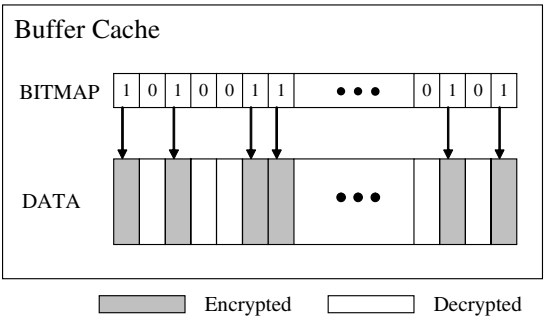


Fig. 3. Information for Partial Encryption/Decryption in Buffer Cache

4 Performance Evaluation

We implemented our buffer cache level support on Linux kernel version 2.6.11. In addition, we made other mechanisms because we want to exclude other factors which affect the performance except buffer caches.

Table 1 shows the experimental setup. We selected the embedded system that has a lower CPU than a desktop and a NAND flash for a storage. The reason is that the performance improvement is more important to the mobile embedded system than other general systems. In that respect, our mechanism is more suitable for the mobile embedded systems.

Table 1. Experimental setup

CPU	Intel Xscale PXA270 520MHz
Main memory	SDRAM 64Mbytes
Storage	NAND Flash - page size 512+16 bytes - erase block size 16K+512 bytes
Operating system	Linux kernel version 2.6.11
File system	YAFFS[19]
Size of buffer cache	4Kbytes
Cryptographic algorithm	AES(ECB mode) with 128bits key

In our experiment, AES was used for encryption. The key length was 128 bits and ECB(Electronic CodeBook) mode was used for block encryption. The throughput of AES was about 240 Kbytes/s.

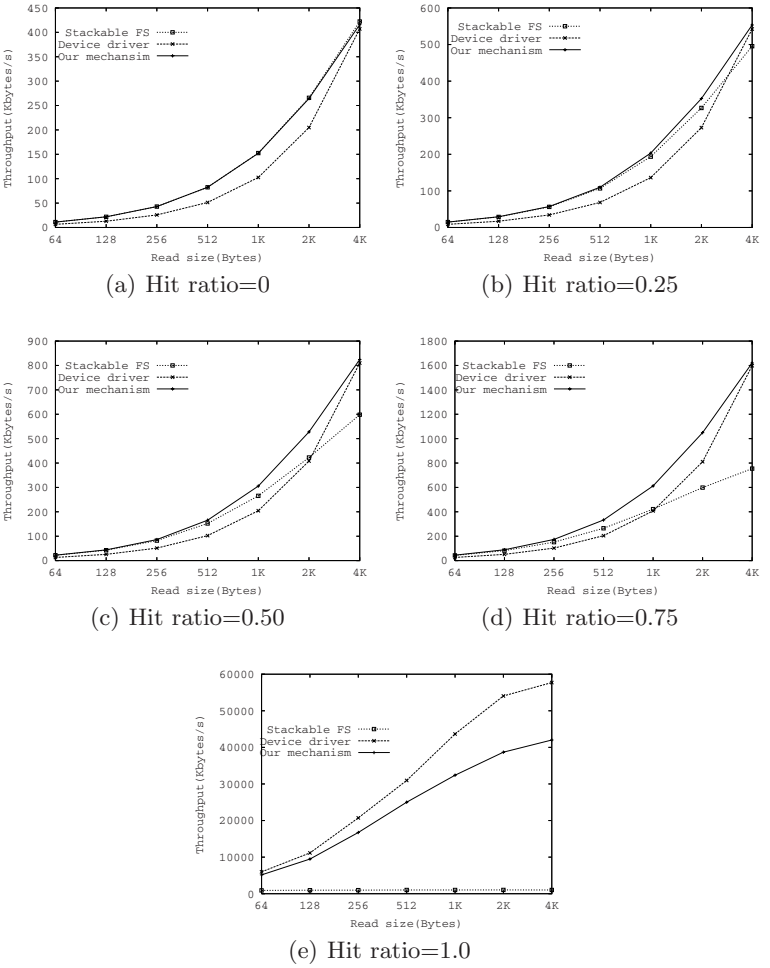
General file system benchmarks are not suitable to measure the effect of a cache. We made two type of synthesized workload to compare our mechanism with other cryptographic file systems in terms of buffer cache hit ratio.

– Workloads with requests of a read operation with cache hit ratio 0, 0.25, 0.50, 0.75 and 1.0

– Workloads with requests of a write operation with cache hit ratio 0, 0.25, 0.50, 0.75 and 1.0

We applied these workloads for various sizes of data: 64bytes, 128bytes, 256bytes, 512bytes, 1Kbytes, 2Kbytes and 4Kbytes. We got the average throughput by repeating the experiments 1000 times.

Fig. 4 shows the results of read operations with the workloads. As the read size increases, the throughput also increases. Before copying data to a user area, the systems read 4Kbytes data from a file regardless of the actual read size. 4Kbytes is the size of a buffer cache. A read operation of 64bytes data results in reading unnecessary 4Kbytes - 64bytes. Therefore, the larger read size makes the larger throughput.



(a) Hit ratio=0

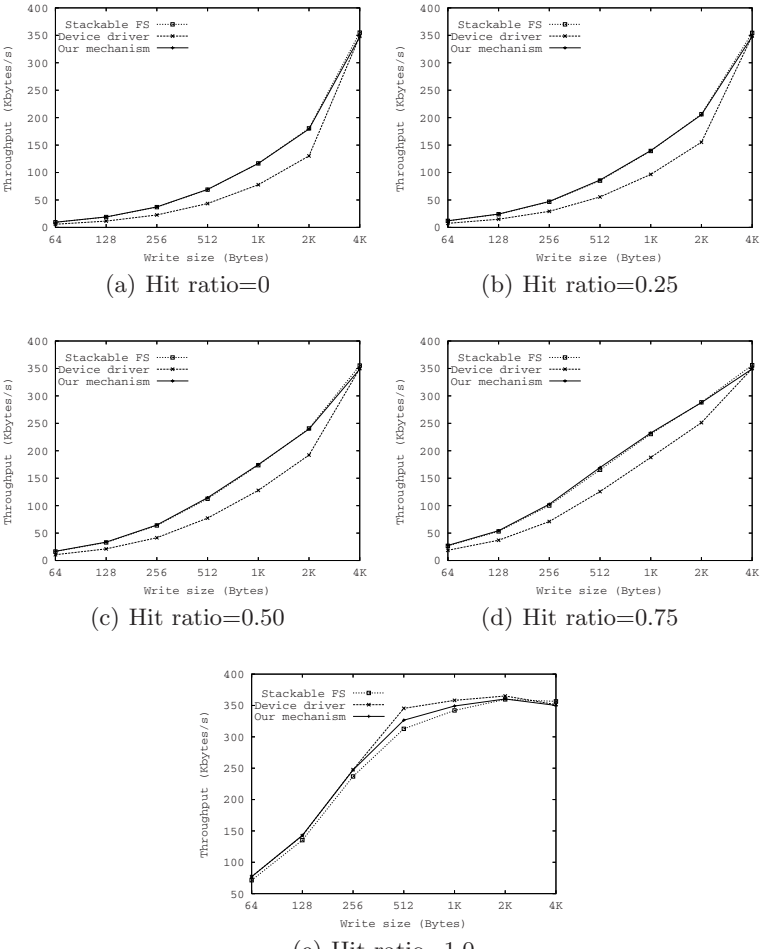
(b) Hit ratio=0.25

(c) Hit ratio=0.50

(d) Hit ratio=0.75

(e) Hit ratio=1.0

Fig. 4. Read with cache hit ratio=0, 0.25, 0.50, 0.75 and 1.0



(a) Hit ratio=0

(b) Hit ratio=0.25

(c) Hit ratio=0.50

(d) Hit ratio=0.75

(e) Hit ratio=1.0

Fig. 5. Write with cache hit ratio=0, 0.25, 0.50, 0.75 and 1.0

Except Fig. 4(e), hit ratio=1.0, our mechanism performs better than others. In case of hit ratio=1.0, the read operations request the data already loaded in a buffer cache. Therefore, unnecessary decryption is never occurred in case of using a device driver. In other graphs of Fig. 4, the performance of the system using a device driver is less than that of our mechanism due to the unnecessary decryption.

In Fig. 4(a), hit ratio=0, the throughput of the system using a stackable file system is similar to that of our mechanism. Our mechanism cannot exploit the buffer cache when hit ratio is 0. In case of using a device driver, it always decrypts entire buffer cache, even though less data is requested. This results in the degradation of the performance in case of using a device driver.

As the hit ratio increases, the relative performance of the system using a stackable file system decreases and the relative performance of the system using a device driver increases. However, the relative performance of our mechanism is not affected largely by the hit ratio because our mechanism can fully exploit the buffer caches. The decrease of the relative performance of the system using a stackable file system results from the redundant decryption.

Fig. 5 shows the results of write operations with the workloads. Like the read operations, the throughput is increases as the write size increases. Except Fig. 5(e), hit ratio=1.0, our mechanism outperforms the case of using a device driver. The system using a device driver performs unnecessary encryption while committing a buffer cache into a disk. This results in the performance degradation. The throughput of the system using a stackable file system is equal to ours because the repeated write is not considered in this experiment. In case of 4Kbytes, other mechanisms must encrypt entire buffer cache like the system using a device driver. Therefore, when the write size is 4Kbytes, the throughput of the system using a device driver is equal to that of others.

From the Fig. 4 and Fig. 5, we can know that the redundant and unnecessary encryption/decryption can be reduced by our mechanism. The system using a device driver outperforms others sometimes. However, this is occurred in a special case, hit ratio=1.0 and this special case does not almost happen in real computer systems.

5 Conclusions and Future Work

Many cryptographic file systems have been developed and used in real systems. In general, kernel-level techniques are preferred because they are more cost-effective than hardware techniques and outperform user-level techniques. However, existing kernel-level cryptographic file systems have some drawbacks. The system using a stackable file system overlooks the effect of buffer caches and the system using a device driver causes unnecessary decryption of data. We modified buffer caches of a kernel to encrypt or decrypt data on a buffer cache partially. This modification is helpful to eliminate unnecessary or redundant operations of encryption/decryption and improve the performance.

References

1. Hasan, R., Myagmar, S., Lee, A., Yurcik, W.: Toward a threat model for storage systems. In: Proceedings of International Workshop on Storage Security and Survivability(StorageSS) (2005)
2. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: Design challenges. ACM Transactions on Embedded Computing Systems 3, 461–491 (2004)
3. GNU: The GNU/Linux CryptoAPI (2003)
4. Dowdeswell, R., Ioannidis, J.: The cryptographic disk driver. In: Proceedings of the Annual USENIX Technical Conference, FREENIX Track (2003)

5. Jetico Inc.: Bestcrypt corporate edn. (2001)
6. Gutmann, P.C.: Secure file system(SFS) for DOS/Windows (1994)
7. Zadok, E., Badulescu, I., Shender, A.: Cryptfs: A stackable vnode level encryption file system.Technical Report CUCS-021-98, Computer Science Department, Columbia University (1998)
8. Wright, C., Martino, M., Zadok, E.: Ncryptfs: A secure and convenient cryptographic file system. In: Proceedings of the Annual USENIX Technical Conference, pp. 197–210 (2003)
9. Microsoft Corporation.: Encrypting file system for Windows 2000 (1999)
10. Zadok, E., Badulescu, I.: A stackable file system interface for Linux. In: Proceedings of the 5th Annual Linux Expo, pp. 141–151 (1999)
11. Wright, C., Dave, J., Zadok, E.: Cryptographic file systems performance: What you don't know can hurt you. In: Proceedings of the Second IEEE International Security In Storage Workshop, pp. 47–62 (2003)
12. Kingston Technology company.: DataTraveler Elite (2006)
13. ABIT Computer corporation.: Secure IDE (2003)
14. GNU.: GNU Privacy Guard (1999)
15. Blaze, M.: A cryptographic file system for UNIX. In: CCS 1993. Proceedings of the 1st ACM conference on Computer and communications security, pp. 9–16 (1993)
16. Cattaneo, G., Catuogno, L., Sorbo, A.D., Persiano, P.: The design and implementation of a transparent cryptographic file system for UNIX. In: Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 199–212 (2001)
17. Zadok, E., Nieh, J.: FiST: A language for stackable file systems. In: Proceedings of the Annual USENIX Technical Conference, pp. 55–70 (2000)
18. Bovet, D.P., Cesati, M.: Understanding the Linux Kernel. O'Reilly (2006)
19. Aleph One.: YAFFS: the NAND-specific flash file system (2002)

SOM-Based Anomaly Intrusion Detection System

Chun-dong Wang, He-feng Yu, Huai-bin Wang, and Kai Liu

School of Computer Science and Technology, Tianjin University of Technology, Tianjin 300191, China
michael3769@163.com, emaif@163.com, hbwang@tjut.edu.cn, liukai@163.com

Abstract. In this paper, a SOM-based anomaly intrusion detection system is proposed, which can contract high-dimension data to lower, meanwhile keeping the primary relationship between clustering and topology. During the experiment, the theory of SOM is used to train three SOMs on the layers of system, process and network. Although our experiment environment is simpler than the real one, the result shows that it has its reference value for us to build intelligent IDSs. Through the analysis of the monitoring results on the three layers from the hacking tools (NMAP, HYDRA), it is suggested that the approach of detecting and the parameters chosen be correct and effective.

Keywords: SOM, neural network, anomaly-based intrusion detection, U-matrix, cluster.

1 Introduction

Along with increasing popularization of the Internet day by day, continuous updating and variety of attack behaviors make the traditional rule-based IDS gradually lose its ideal effect. SOM-based IDS, as the representative of the new generation of intelligent IDS, compared with the traditional IDS, has the following special advantages:

(1) The analysis technique of the traditional IDS is mainly based on the model of statistics [1], and depends on several assumptions. SOM-based IDS can be trained through a great deal of instances, can learn knowledge by itself and acquire the ability of prognostication. The whole process is completely abstract calculation, with no emphasis on the assumption of the distribution of the data.

(2) The traditional IDS depicts attack characteristics to be limited by a fixed sequence, and the threshold value is mostly based on the experience. False positive and false negative usually happen and it can not easily identify new attack methods. In the terms of the ability of self-applicable and fault tolerant [2], SOM-based IDS need not understand the detail of knowledge, and can master the inherent relationship of each generous character of the system by itself. After mastering the normal working mode [3] of the system, SOM-based IDS can react to all affairs which deviate from the normal working mode, and then discover new attacks.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 356–366, 2007.

2 Self-organizing Maps (SOMs)

SOM was proposed by professor Kohonen, the neural network expert in 1981. Early researches on neural network-based IDS mostly adopted BP (Back-Propagation) neural network [4-5] for modeling, as well as multi-layer perceptron and Hamming, but they all have restrictions and weaknesses [6]. By contrast with the above mentioned neural networks, the best advantage of SOM is the ability of unsupervised learning, which can transplant the system to new surroundings, and the training data has no marks.

2.1 About SOM Algorithm

The algorithm of SOM is recursive. First, every neuron corresponds to a N-dimensions vector $Wi=[w_{i1},w_{i2},\ldots,w_{iN}]$. At every stage of training, sampling vector $X_k=[x_1,x_2, \ldots,x_N]$ is selected from the training set randomly, then calculate the distance between X_k and all the weight vectors. c is the BMU(best-matching unit), and the minimum distance between c and X_k is:

$$\| X_k - W_c \| = \min_{i=1}^{M} \| X_k - W_i \| \tag{1}$$

Next, update the weight vector of the neuron which is in neighbourhood zone of the winner cell's topology. The rule is as follow:

$$W_i(k+1) = \begin{cases} W_i(k) + \alpha(k)[X_k - W_i(k)], & i \in N_c(k) \\ W_i(k), & i \notin N_c(k) \end{cases} \tag{2}$$

In Formula 2, N_c refers to neighbourhood zone of the centre neuron w_c. In the process of learning, the initialization of $N_c(k)$ can be big, then contracts gradually, as follow:

$$N_c = INT(N_c(0)(1 - k/L)), \quad k = 0,1,2,\ldots L \tag{3}$$

In Formula 3, $N_c(0)$ means the initial neighbourhood radius, L, the times of the iteration, INT(.), the integral function. $N_c(k1)$, $N_c(k2)$, $N_c(k3)$ stand for the topology neighbourhood zone of the winner cell whose iterative times are k_1, k_2, k_3($k_1 < k_2 < k_3$).

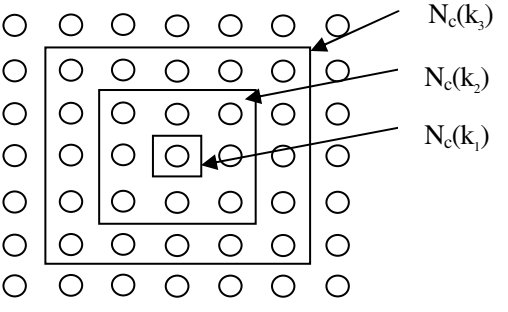


Fig. 1. topology adjacent domains on two- dimension network

Usually, the learning rate $\alpha(k)$ $(0<\alpha(k)<1)$ is a constant, which is close to 1.0 in the beginning, then lessens gradually. For example, $\alpha(k)$ can be 0.8(1-k/L). With the increase of the times of the iteration, $\alpha(k)$ tends to zero, which ensures the learning process to refrain from rash action.

2.2 The Steps of the Learning Algorithm of SOM

The concrete steps of the learning algorithm of SOM are as follows:

Step 1. Setting variables and parameters: Let $X(k) = [x_1(n), x_2(n),\ldots, x_N(n)]^T$ be the input vector, or training sample, $W_i(k)=[w_{i1}(n), w_{i2}(n), \ldots, w_{iN}(n)]T$ be the weight vector, i=1,2, …, M, and the total times of iteration be L.

Step 2. Initialization: Initialize the weight vector W_i with a small random number in a certain interval. Let the neighbourhood radius be $N_c(0)$; the learning rate be $\alpha(k)$; and then normalize weight vector $W_i(0)$ and all the input vector X.

$$X^{'} = \frac{X}{\| X \|} \tag{4}$$

$$W_i^{'}(0) = \frac{W_i(0)}{\| W_i \|} \tag{5}$$

In the above formulas, $\| W_i(0) \| = \sum_{j=1}^{N} [w_{ij}(0)]^2$ and $\| X \| = \sum_{j=1}^{N} (x_i)^2$ are Euclidean norm of the weight vector and input vector.

Step 3. Data sampling. Select training samples X' from the input space.

Step 4. Approximate matching: According to the standard of the minimum Euclidean distance:

$$\| X^{'} - W_c^{'} \| = \min_i \| X^{'} - W_i^{'} \| \, i = 1,2,...,M \tag{6}$$

select winner cell c, implement the competitive process of neurons.

Step 5. Updating: Update the weight vectors of the cordial neuron, who are in the topology neighbourhood zone of the winner cell $N_c(n)$ under the following rules:

$$W_i^{'}(k+1) = W_i^{'}(k) + \alpha(k)[X - W_i^{'}(k)] \tag{7}$$

Step 6. Updating the learning rate $\alpha(k)$ and the topology neighbourhood zone, and then normalize the weights after learning.

$$\alpha(k) = \alpha(0)(1 - \frac{k}{L}) \tag{8}$$

$$N_c(n) = INT\left[N_c(0)\left(1 - \frac{k}{L}\right)\right] \tag{9}$$

$$W_i^{'}(k+1) = \frac{W_i(k+1)}{\| W_i(k+1) \|} \tag{10}$$

Step 7. Judging whether the times of the iteration k exceeds L or not, if k<= L then turn to step 3, otherwise end the process of iteration.

During the training, output neurons are sorted through adjusting their weight vector, in order to be close to the probability density. Though produced randomly at the beginning, the weight vector of the output neurons get closer and closer to the distribution of the input data after long time running, through updating the weight vector continuously.

3 Data Sampling, Preprocess, Standardization and Training

3.1 Data Sampling and Preprocess

In this paper, three hosts are used to constitute an LAN. One of the hosts, running Red hat 7.2, works as the testing platform. We mainly sample and preprocess the data from three layers (System layer, Process layer and network layer). The procedure is carried out by the shell program. First of all, the sampling data consist of 1000 sets of relevant data, which imitate the normal operation, and the sample interval is 10 seconds. The normal operations include the usage of familiar basic orders and applications, compilation of programs, application of ftp and telnet, and Web browse etc. Although time and amount of sampling are not enough, we imitated a great deal of actual operation during samplings, which is to a degree representative. As is the choice of parameters, we mainly consider those parameters affecting the system most and likely to be abnormal, while attack happening or after.

On the system layer, we choose 9 Characteristic Parameters:
- -S1: usage of virtual memory (kb)
- -S2: spare memory (kb)
- -S3: exchange speed from disk to memory (kb/s)
- -S4: exchange speed from memory to disk (kb/s)
- -S5: read-in speed of block device (kb/s)
- -S6: write-out speed of block device (kb/s)
- -S7: times of interruption per second (including timer interruption)
- -S8: CPU processing time of user process (%)
- -S9: CPU processing time of system process (%)

On the process layer, we choose 6 Characteristic Parameters:
- -P1: total number of process
- -P2: number of process at the state of running
- -P3: CPU time occupancy of every process at the state of running (%)
- -P4: memory occupancy of every process at the state of running (%)
- -P5: virtual memory occupancy of every process at the state of running (%)
- -P6: initial time of every process at the state of running

On the network layer, we choose 6 Characteristic Parameters:
- -N1: number of TCP connection
- -N2: port number of the connection
- -N3: IP address of the connection
- -N4: connection state
- -N5: number of error packet accepted
- -N6: number of error packet sent

Besides numeral quantity, there is also non-numerical quantity in the characteristic data. For numeral quantity, it keeps the initialized value. For IP address, it intercepts the last part. For the non- numerical quantity, it is inverted to the numeral quantity. For example, for the state of TCP, we respectively use: 1,2,3,4,5,6,7,8,9,10,11,12 to represent: Established, SYN-Sent, SYN-Recy, Closed, Listen etc; for the starting time of the process, we change the notation of ":" into ".".

The corresponding parameter is mainly obtained from the filtration of the results, which come from different UNIX system commands, including: Vmstat, ps, Netstat, top and so on. The programme is completed with script language. The interception, conversion and coordination of the data are finished under these commands, including: sed, awk, cat, cut, grep etc. The data of sampling is stored in three files (systemlayer.dat, processlayer.dat, networklayer.dat) according to a certain format.

3.2 Standardization

Among the characteristic values selected, the differences are very obvious. So, in order to balance the effects of the training result, we can standardize every characteristic value into the area of 0~1, which is the demand of the application of SOM. The formula of standardization is as follows:

$$nv[i] = \frac{v[i]}{\sqrt{\sum_{K} v[k]^2}} \tag{11}$$

In Formula 11, nv[i] is the standardized value of i, v[i], the value size of i, and K, the number of the characteristic vector.

3.3 Data Training

Map training is carried out according to the above-mentioned algorithm, mainly using SOM_PAK [7-8]. Three Maps should be trained on three different layers in our experiment. The number of neurons in each output layer of each Map is 12x8=96. The process of training is divided into two stages. At the first stage, the weigh vector of each neuron is sorted. At the beginning, the neighbourhood radius is chosen relatively bigger, generally equaling to the diameter of the Map, finally changing into 1, and the bigger learning rate should also be chosen, gradually changing into 0 with the increase of the training times. In order to make the weigh vector of each neuron more accurate, the second stage should be adjusted. At this stage, the smaller learning rate and the neighbourhood radius should be chosen correspondingly.

U-matrix (unified distance matrix) is a visual method of SOM cluster structure. The U-matrix chart shows the distance between the weight vector of a certain neuron and that of its adjacent neuron. Usually, different gray levels are used to express the size of the distance. Fig2, 3, 4 are three U-Matrix figures, respectively expressing the results of SOM training on three layers (system layer, process layer and network layer). Each small black mark in the figure means a neuron unit.

Table 1. Training parameters of SOMs on layers of system, process and network

	System layer	Process layer	Network layer
Dimensions of input vector	9	6	6
Topology	Rhombus	Rhombus	Rhombus
Adjacent domain function	Bubble	Bubble	Bubble
Dimension of Map's X direction	12	12	12
Dimension of Map's Y direction	8	8	8
Function of learning rate	Linear	Linear	Linear
Times of training at phase one	10000	10000	10000
Initial value of learning rate at phase one	0.1	0.3	0.3
Initial radius at phase one	10	10	10
Times of training at phase two	100000	100000	100000
Initial value of learning rate at phase two	0.02	0.01	0.02
Initial radius at phase two	3	3	3

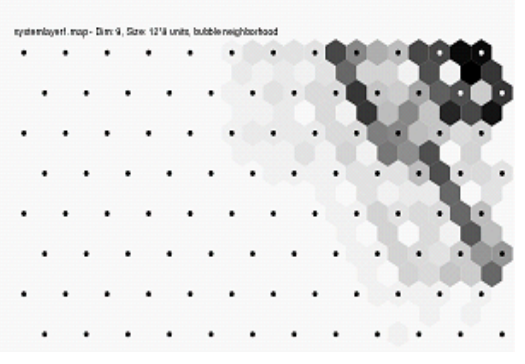


Fig. 2. U-Matrix map of system layer

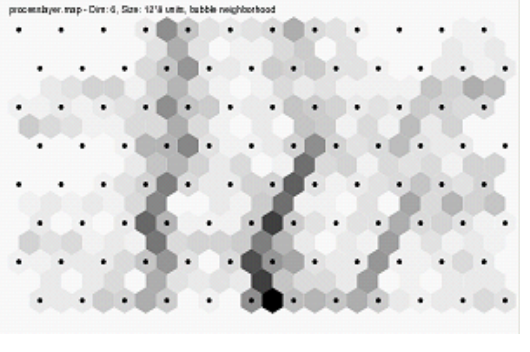


Fig. 3. U-Matrix map of process layer

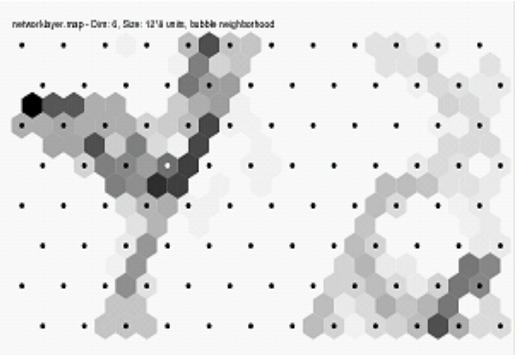


Fig. 4. U-Matrix map of network layer

In the learning algorithm of SOM, different choices of training parameters have different influence on training results. In this paper, different Maps are mainly obtained through changing the training stages, learning rate and initialized radius at the two stages. Finally, make sure the parameter value relatively small, and then, train the sampling data according to the above-mentioned parameter. The results can be viewed in Table 2. The main content of the result is the training time and corresponding quantization error.

Table 2. Training time and quantization error at two phases

Index of performance	System layer		Process layer		Network layer	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
Training time	<1s	12s	<2s	10s	<2s	11s
Quantization error	235.6	78.3	137.3	23.8	30.8	0.3

Table 3. Alarm threshold values on the system layer

Y / X	0	1	2	3	4	5	6	7
0	802	120	59	558	92	23	883	214
1	49	135	70	91	35	68	26	55
2	88	145	101	140	0	89	75	215
3	95	103	112	157	0	91	206	68
4	682	150	142	0	0	111	83	83
5	417	168	166	413	95	70	87	72
6	0	220	0	98	99	145	86	86
7	396	0	0	0	60	167	90	89
8	1034	0	204	0	0	55	96	290
9	347	1117	0	551	2881	263	0	166
10	0	657	0	0	0	0	32	143
11	6868	0	120	538	0	0	481	34

Table 4. Alarm threshold values on the process layer

X \ Y	0	1	2	3	4	5	6	7
0	6.2	3.7	90.6	0	13.7	38.8	41.4	332.2
1	0	12.1	77.0	50.1	0	6.0	75.9	100.1
2	27.1	0	60.6	20.9	6.8	19.4	140.1	7.1
3	123.6	0	18.6	120.5	33.3	10.4	25.5	20.1
4	0	0	44.7	0	36.2	14.7	103.3	17.2
5	45.8	0	0	11.2	136.4	0	0	82.9
6	0	227.2	0	0	16.1	4.3	162.3	0
7	40.7	125.9	95.0	39.5	0	15.7	0	56.7
8	15.5	0	56.9	12.4	58.7	12.5	0	58.7
9	8.1	2.0	17.7	0	0	36.1	44.5	75.1
10	0	21.8	0	13.6	0	94.5	72.2	38.3
11	60.7	28.6	111.0	7.6	17.9	56.8	126.6	559.0

Table 5. Alarm threshold values on the network layer

X \ Y	0	1	2	3	4	5	6	7
0	0.99	0	0	0.87	0.083	0	0.39	0.24
1	0	0.33	0	0	0	0.13	0	5.07
2	4.88	0	0	0	0.64	0	0.21	0
3	4.41	0.29	0.01	0	0	0	0	0.04
4	5.69	0	0	0.18	0.81	0	1.94	0
5	5.34	0	0.21	2.47	0	0.00	0	0.00
6	0.21	0	0	0	0	0	0	0
7	0	0	0.00	0	0.36	0	0	0
8	1.50	0	0	0	0	0	0	5.88
9	0	0	0.00	0	0	1.50	0	0
10	0.58	0	0	0	0	0	0	0
11	1.58	0	2.90	0	4.59	0	0	4.23

Table 2 shows the average value of the quantization error. In order to determine the alarm threshold values, we must follow the formula 1 to work out the distance between the data of normal training instances and the corresponding BMU, here it is called BMU Distance. After working out every BMU Distance of every input vector, several distance values can be got from one BMU, and the biggest one is used as the alarm threshold. But not every neuron has such a distance value, and the alarm threshold value is specified as 0 when this happens.

Table 3 shows the alarm threshold values of every neuron of the corresponding system layer MAPs. The most left column of the table shows the X-coordinate value of the MAP, respectively getting the integer between 0 and 11; the most above row shows the Y-coordinate value of the MAP, respectively getting the integer between 0 and 8.

The alarm threshold values of process layer and network layer can be obtained using the same method, and respectively show in Table 4 and Table 5.

4 Experiment Result and Its Analysis

Using the learning algorithm of SOM mentioned above, three Self-Organizing Maps were trained from the training data on the layers of system, process and network. And the alarm thresholds were determined. If considering every BMU as a cluster, then the cluster can be used to determine the normal or abnormal of new instances. This paper uses Distance Function $f_d(s,K)$ to measure the distance between new instance s and cluster K, and uses it to decide whether the new instance abnormal or not. The concrete function is as follows:

$$f_d(s,Normal) = \min\{f_d(s,K_i) \mid K_i \in C\} \tag{12}$$

$$x_{abnormal}(s) = \begin{cases} 1 & if \quad f_d(s,Normal) \geq t_i \\ 0 & otherwise \end{cases} \tag{13}$$

In the above formulas, C is the set of normal subspace. t_i is used to distinguish the threshold value between the normal class and the abnormal. f_d uses Euclidean distance:

$$f_d(s,K_i) = \| s - w_{K_i} \| \tag{14}$$

In the above formulas, w_{K_i} is the weight vector matrix of the cluster K_i.

In order to achieve the purpose of intruding the important system, the main purpose of the inner intruder was not to intrude the Client, but running suspicious procedures via the Client. On the account of detecting the abnormal of Client in this experiment, we mainly detect the abnormal situation of system while it is running suspicious operation or suspicious procedures. So, anomaly intrusion on the Client is a vital aspect of internal network intrusion detection.

This paper uses NMAP and Hydra to produce the abnormal data of the system, in the meantime, the monitor procedure Monitor was run at the backstage. Through the detection during running these two tools, the abnormal circumstances of different layers are different, which mainly shows that the ratio of the abnormal data in the whole detected data is different. The size of ratio also reflects that the influences on different layers are different, also shows that, in the meantime, the invader is not continuous to make a system working abnormal during intrusion. Therefore, in order to enhance the detection rate, the sampling data should be sampled several times in the process of anomaly-based intrusion detection. Detection rate on different layers during intrusion with the same set of data can be seen from the Table 6.

Table 6. Detection rate of different layers during intrusion

Tools	Detection rate of system layer	Detection rate of process layer	Detection rate of network layer
NMAP	0%	36%	100%
HYDRA	66%	50%	86%

Supposing the time of intrusion is Ta, and the probability of discovering abnormity effectively with sampling one set of data at t_i is p, so, after continuous sampling n sets of data in Ta, the final anomaly-based intrusion detection rate is:

$$P(A) = 1 - (1 - p)^n \tag{15}$$

Under the situation of not changing the threshold value of warning, if you want to make the rate of anomaly-based intrusion detection achieve more than 99%, just ensure $1-(1-p)^n$ >0.99, the same as $(1-p)^n$ <0.01. Fig 5 shows the relation diagram of n and p, when P(A) is more than 99% in the Formula 15. The numerical label in the form is the corresponding coordinate of Y axis. As long as values are taken above the curve, the total detection rate can achieve more than 99%.

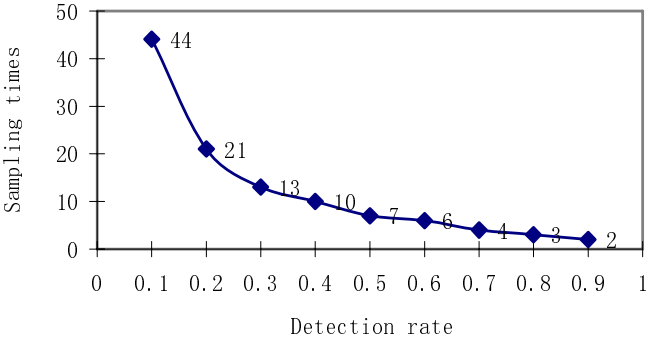


Fig. 5. Relation diagram of n and p

The detection result above shows that the way of sampling many times makes the detection rate raise to a satisfactory extent, while not needing to change the size of the threshold value of warning, either raise the false positive rate. From Table 6 and Fig 5, we discover that, in order to make the rate of anomaly-based intrusion detection produced by Hydra on the system layer achieve more than 99%, we should sample 6 times during the running of Hydra; in order to make the rate of the anomaly-based intrusion detection produced by NMAP and Hydra on the process layer achieve more than 99%, we should sample 7 times and 13 times respectively during their running; in order to make the rate of the anomaly-based intrusion detection produced by NMAP and Hydra on the network layer achieve more than 99%, we should sample 1 time and 3 times respectively during their running. Result shows that the method of

anomaly-based intrusion detection and the choice of the monitor parameter are viable and meaningful.

5 Conclusions

In this paper, a novel way of anomaly-based intrusion detection using SOMs is proposed. During the experiment, we use the theory of SOM to train three SOMs on the layers of system, process and network. Although our experiment environment is simpler than the real one, the result shows that it has its reference value for us to build intelligent IDSs. In larger and more complicated real experiment environment, the characteristic value should be selected more extensively, the bound of the time for training the normal sets of data should be a little bit larger, the data should also be a little bit more, and the dimension of the Maps should also be chosen a little bit larger. Thus, training time will consumedly increase, but as long as the training time is restricted in a certain bound, it is acceptable.

Acknowledgements. "Computer Software and Theory" of Tianjin Pivot Subject Establishment, and "The Higher Education Institution Science and Technology Development Fund" (No. 2006BA19) of Tianjin Municipal Education Commission, and Wireless Network Research Institute of Tianjin University of Technology supported this work.

References

1. Ord, K.: Outliers in statistical data: V.barnett and t.lewis. International Journal of Forecasting 12(1), 175–176 (1996)
2. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences, vol. 30. Springer, Heidelberg (1995) (second extended edition 1997)
3. Zanero, S., Savaresi, S.M.: Unsupervised Learning Techniques for an Intrusion Detection System [C]. In: Proceedings of 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus (2004)
4. Dangfeng, Z., Chunhui, H.: Research on Intrusion Detection Technique Based on Rapidly BP Learning Algorithm. Network Security Technology and Application (8), 36–37, 33 (2006)
5. Shengjun, W., Changzhen, H., Fei, J.: An Intrusion Detection Method Based on Improved BP Neural Network Algorithm. Computer Engineering 31(13), 154–155, 158 (2005)
6. Hagan, M.T., Dcmuch, H.B., Beale, M.: Neural Network Design. China Machine Press, Beijing (2002)
7. Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J.: SOM_PAK, The Self-Organizing Map Program Package, Version 3.1 (1995)
8. The Self-Organizing Map Program Package, http://www.cis.hut.fi/research/som_pak/

TCP-Taichung: A RTT-Based Predictive Bandwidth Based with Optimal Shrink Factor for TCP Congestion Control in Heterogeneous Wired and Wireless Networks

Ben-Jye Chang[1], Shu-Yu Lin[1], and Ying-Hsin Liang[2]

[1] Department of Computer Science and Information Engineering
Chaoyang University of Technology, Taiwan, ROC
{changb,s9227623}@mail.cyut.edu.tw
[2] Department of Computer Science and Information Engineering
Nan-Kai Institute of Technology, Taiwan, ROC
t136@nkc.edu.tw

Abstract. TCP congestion control works well in wired Internet network but it is difficult to determine a good congestion window in a heterogeneous wireless network that consists of the wired Internet and various types of wireless networks. Therefore, we propose herein a novel adaptive window congestion control, namely TCP-Taichung, for TCP connections in heterogeneous wireless networks. The proposed RTT-based congestion control is designed for the Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery phases to increase throughput while supporting high fairness. In addition, an optimal shrink factor is proposed to determine the optimal *cwnd* and *ssthresh* for the cases of network congestion or wireless-link error. Numerical results demonstrate that TCP-Taichung outperforms other approaches in goodput and fairness under various wireless network topologies. Especially, in the case of 10% packet loss rate in wireless link, the proposed approach increases goodput up to 200% as compared with NewReno.

Keywords: TCP, RTT, congestion control, error wireless link, heterogeneous wireless networks.

1 Introduction

ALL IP based access for Internet rich resources anytime anywhere makes different wireless network standards and technologies have been developed progressively. Several successful IP-based wireless standards include IEEE 802.11 family [1] and 802.16 [2] for wireless networks, and 3G [3]/4G [4] for mobile communications. The important protocols of all IP based wireless networks are IP [5] and TCP [6] that provides reliable end-to-end connections in transport layer and connectionless-oriented hop-by-hop routing in network layer, respectively. Although TCP works well in the wired Internet, it is hard to determine a good

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 367–378, 2007.

congestion window (*cwnd*) in wireless networks. In addition, TCP congestion control [7] can be divided into four phases, including the slow start, Congestion Avoidance, Fast Retransmit and fast recovery phases. In [8], Hoe proposed a bandwidth delay product based computation for determining *initial ssthresh*, in which the first three ACKs are required for sender to determine it. The main ideas of these mechanisms are based on sliding window and additive increase multiplicative decrease (AIMD) [9] algorithms. We can classify these TCP congestion control mechanisms into two types: the AIMD-based mechanism and the estimated available bandwidth mechanism.

First, several AIMD-based algorithms [10,11,12,13] have been proposed. TCP-Tahoe [12] TCP-Reno [13] and TCP-NewReno [10] are typical AIMD-based algorithms. [14,15] proposed an additive increase adaptive decrease (AIADD) mechanism to improve such a problem by estimating available bandwidth. Second, several approaches of estimated available bandwidth have been proposed [14,15,16,17,18,19]. TCP-Westwood [14,15] is proposed to estimate available network bandwidth by using the ACK reception rate at sender. Although TCP-Westwood improves throughput, it suffers from inaccurate estimation of available bandwidth. Then, [17,18] proposed TCP-Westwood+ for estimating network available bandwidth more accurate and then increases throughput. Based on TCP-Westwood, TCP-Jersey [16] was proposed to provide the Explicit Congestion Notification (ECN) in a router's Active Queue Management (QAM). [20] proposed TCP-Vegas to determine whether a network occurs congestion or not by using the difference of the Expected (i.e., $cwnd/RTTmin$) and Actual (i.e., $cwnd/RTT$) rates. Unfortunately, the optimal lower and upper bounds of α and β are difficult to determine in TCP-Vegas.

Although many approaches have been proposed to support or improve congestion control in TCP, it is difficult to provide high throughput while providing good fairness and friendliness in heterogeneous wireless networks. Therefore, in this paper, we propose an adaptive RTT-based *cwnd* control with a precise shrink factor approach to achieve high throughput while providing good fairness.

The remainder of the paper is organized as follows. Section II describes the network model of TCP connections and performance metrics for evaluation. Section III then describes the TCP-Taichung approach for controlling congestion window in TCP under heterogeneous wireless networks. Section IV evaluates the performance of the proposed RTT-based approach. Section V draws several important conclusions.

2 Network Model

We model a TCP/IP network as a logical graph, $G = (V, E)$, which consists of a set of nodes, V, and a set of logical end-to-end connections, E. Two end types are considered in a TCP/IP connection, including the TCP connection server (V_s) and client (V_c). Fig. 1 demonstrates an example of four TCP connections, including $E^1_{V^2_s,V^7_c}$, $E^2_{V^1_s,V^5_c}$, $E^3_{V^4_s,V^3_c}$, and $E^4_{V^8_s,V^6_c}$ in heterogeneous wireless

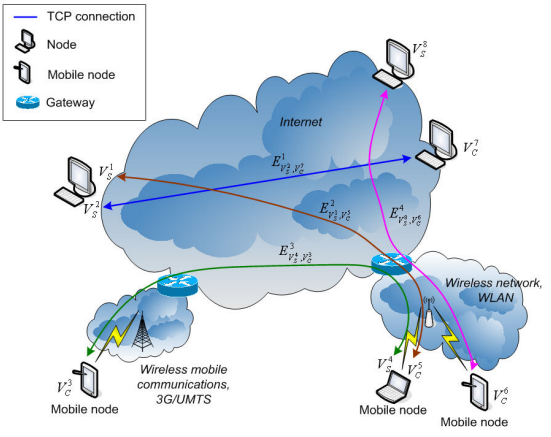


Fig. 1. An example of TCP connections in heterogeneous wireless networks

networks, in which connections $E^2_{V_s^1,V_c^5}$ and $E^4_{V_s^8,V_c^6}$ have a single wireless link and connection $E^3_{V_s^4,V_c^3}$ has two wireless links.

Various performance metrics, including goodput and fairness, are adopted to evaluate the TCP-Taichung approach and other approaches under various network topologies with different error rate of wireless links. First, since packets transmission from sender may be lost or discarded due to network congestion and error wireless links, not all transmitted packets will arrive at receiver. Therefore, we adopt average goodput to evaluate the congestion control mechanisms. Average goodput is defined by

$$Average\ Goodput = \frac{\sum_{n=1}^{N} \frac{MaxACKSeqNo_n \times MSS_n}{TotalTransmissionTime}}{N}, \tag{1}$$

where $MaxACKSeqNo_n$ denotes the maximum ACK number of connection n, MSS_n is the maximum segment size of connection n, and N is the total number of connections. Second, fairness is another important metric for evaluating whether all connections fairly utilize the bandwidth of the network or not. A good TCP congestion control approach should achieve a fair usage of bandwidth among all connections. The fairness metric is defined by

$$Fairness = \frac{\left(\sum_{n=1}^{N} S_n\right)^2}{N \times \sum_{n=1}^{N} (S_n)^2}, \tag{2}$$

where S_n is the throughput of connection n. The range of $Fairness$ is between one and $1/N$, where $Fairness = 1$ is the most fair result.

3 Adaptive Network Bandwidth Approach

This section first briefly describes the problems of congestion control in TCP connections under heterogeneous wireless networks. Then, we detail the motivations of the proposed TCP-Taichung approach. Finally, the optimal congestion control is described in two aspects of the network bandwidth is sufficient or not. First aspect proposes two adaptive control mechanisms to determine the *optimal cwnd*, $cwnd_{opt}$, and the ideal slow start threshold, $ssthresh_{ideal}$, in the increasing *cwnd* phase while the expected bandwidth is enough. Second aspect provides the determinations of the *optimal cwnd*, $cwnd_{opt}$, and the *network ssthresh*, $ssthresh_{net}$, in the decreasing *cwnd* phase while the expected network bandwidth is insufficient.

3.1 In the Increasing *cwnd* Phase While the Expected Network Bandwidth Is Enough

The algorithm of determining the optimal congestion window ($cwnd_{opt}$) Assume that MSS is S bits and the data rate of is R bit per second and the minimum round-trip time of TCP connections is RTT_{min} when the network traffic is not saturation. The total time of transmitting a packet in such a light-traffic network is thus $(S/R) + RTT_{min}$(sec). In addition, in such a light-traffic network, the increment of congestion window (*cwnd*) of sender will not affect the round trip time and the total time of transmitting *cwnd* packets should be less than $(S/R) + RTT_{min}$, as shown below

$$cwnd \cdot (S/R) < (S/R) + RTT_{min}, \tag{3}$$

that is,

$$cwnd < \frac{RTT_{min}}{S/R} + 1. \tag{4}$$

Based on the analysis, we obtain the round trip time (RTT) of a connection, which is close to RTT_{min}, if *cwnd* is less than $\frac{RTT_{min}}{S/R} + 1$. On the other hand, if $cwnd \geq \frac{RTT_{min}}{S/R} + 1 = k$, the network traffic becomes saturation and RTT will be increased based on $(cwnd - k) \cdot S/R$ till congestion. As a result, the value of $\frac{RTT_{min}}{S/R} + 1$ is the optimal window size of the connection, which is denoted as $cwnd_{opt}$. In other words, when *cwnd* is larger than $cwnd_{opt}$, the network will enter into the saturation state.

The algorithm of determining the ideal ssthresh ($ssthresh_{ideal}$). In TCP, the Slow Start threshold, *ssthresh*, is an important parameter for a sender to determine when to enter into the Congestion Avoidance phase. If *ssthresh* is set too small, the sender will enter into the Congestion Avoidance phase early that brings two disadvantages: low goodput and unfairness. On the other hand, if *ssthresh* is set too large, the sender will send too many packets in a congested network, which will cause the congested network more serious. Since the congested intermediate router may discard the over-sent packets, this causes too

many duplicate ACKs or timeout. In the worst case, sender will enter into the Slow Start phase. *Cwnd* will be set to one and result in low goodput. Therefore, the determination of an *ideal ssthresh*, *i.e.*, $ssthresh_{ideal}$, is proposed in this paper for achieving high goodput and high fairness.

Since *ssthresh* is affected by traffic load, an ACK-based mechanism is proposed to determine the $ssthresh_{ideal}$ adaptively. We first define the received inter-ACK period of two continuous ACKs at sender as $InterACK$ that is a parameter represented current traffic load. Then, the new expected smooth $InterACK$ is denoted as $InterACK_{new}$, which is computed on line based on the Exponential Weight Moving Average (EWMA) model,

$$InterACK_{new} = \alpha \cdot InterACK_{avg} + (1 - \alpha) \cdot InterACK_{old}, \tag{5}$$

where $InterACK_{avg}$ denotes the average inter-ACK time of the connection, $InterACK_{old}$ is the inter-ACK time between current and previous ACKs, and α is a constant value ranging of $(0, 1)$.

Based on receiving each new ACK packet, sender can adaptively determines the expected smooth $InterACK$. Consequently, the *ideal ssthresh* of the Slow Start phase can be formulated by

$$ssthresh_{ideal} = \frac{RTT_{min}}{InterACK} + 1, \tag{6}$$

which is also the *optimal ssthresh* of the first *ssthresh* in the Slow Start phase.

The determination of adaptive network bandwidth and network ssthresh After obtaining the *ideal ssthresh*, the adaptive network bandwidth can be estimated by

$$Adaptive\ network\ bandwidth = \frac{ssthresh_{ideal}}{RTT}, \tag{7}$$

where RTT is the round-trip time of the last received ACK. Under the estimated bandwidth, *cwnd* will increase exponentially until *cwnd* exceeds the *network ssthresh*, $ssthresh_{net}$, in the Slow Start phase and then enters into the Congestion Avoidance phase. As the *ideal ssthresh* analysis in previous subsection, the *network ssthresh* of the expected available bandwidth is an important threshold that should be addressed for achieving high goodput and fairness. Therefore, we first determine the *network ssthresh* by

$$ssthresh_{net} = (Adaptive\ network\ bandwidth) \cdot (RTT_{min}). \tag{8}$$

That means the *idela ssthresh* of the expected adaptive network bandwidth will occur at the situation of RTT_{min}. In consequence, after applying (7) to (8), we thus have

$$\begin{aligned} ssthresh_{net} &= (Adaptive\ network\ bandwidth) \cdot (RTT_{min}) \\ &= \frac{ssthresh_{ideal}}{RTT} \cdot RTT_{min}, \end{aligned} \tag{9}$$

or

$$ssthresh_{net} = \left(\frac{RTT_{min}}{InterACK} + 1\right) \cdot \frac{RTT_{min}}{RTT}, \tag{10}$$

where the term $\frac{RTT_{min}}{RTT}$ is called a shrink factor of the *ideal ssthresh*. Since network bandwidth is shared among all connections, RTT of an adaptive network bandwidth should be less than RTT_{min}. The *network ssthresh* can be formulated by shrinking the *ideal ssthresh* based on the shrink factor. Consequently, when $cwnd \geq ssthresh_{net}$, sender will enter into the Congestion Avoidance phase. In addition, we consider the value of *ssthresh* in the Congestion Avoidance phase, which is also determined based on the shrink factor of *ssthresh* when three duplicate ACKs received or timeout occurs.

Based on the analysis of $cwnd_{opt}$, $ssthresh_{ideal}$ and $ssthresh_{net}$, the algorithm of controlling congestion window while receiving ACKs at sender is demonstrated in Fig. 2. There are two significant contributions. First, the expected smooth $InterACK$ is computed based on the EWMA model for achieving accurate estimation of $InterACK$. Second, the condition of switching from the Slow Start phase to the Congestion Avoidance phase is based on two thresholds of $ssthresh_{net}$ and $ssthresh_{ideal}$ for accurately predicting when will occur network saturation. The variation of congestion window of receiving ACK in Slow Start of TCP-Taichung is compared with NewReno. In Fig. 3, we can observe that TCP-Taichung avoids occurring timeout in the first Slow Start phase and results in higher goodput than that of NewReno, in which the goodput of the proposed approach is 8779980 bps but that of NewReno is 8177664 bps. The accurate analysis of *ideal ssthresh* and the optimal congestion window in Slow Start results in high goodput.

ReceiveACK()
{
  if$\left(\frac{InterACK}{InterACK_{min}} < cwnd \,\|\, InterACK_{org} > InterACK\right)$ {
  // Filter InterACKdiff
    $InterACK = (\alpha) * InterACK_{org} + (1-\alpha)InterACK$
  }
  $ssthresh_{net} = \left(\frac{RTT_{min}}{InterACK} + 1\right) + \left(\frac{RTT_{min}}{RTT}\right)$
  if($cwnd < ssthresh_{net}$ && $cwnd < ssthresh_{ideal}$) {
    $cwnd++$ //expontial increasing phase
  }
  else {
    $cwnd += \frac{1}{cwnd}$ //linear increasing phase
  }
}

Fig. 2. The algorithm of controlling congestion window while receiving ACKs at sender

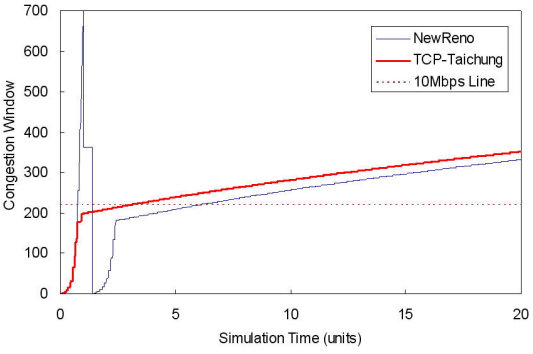


Fig. 3. The variation of congestion window in Slow Start

3.2 In the Decreasing *cwnd* Phase While the Expected Network Bandwidth Is Insufficient

When *cwnd* exceeds $ssthresh_{net}$ and $ssthresh_{ideal}$, sender enters into the Congestion Avoidance phase and *cwnd* increases linearly while receiving each ACK. In this phase, the available network bandwidth becomes worse and buffers of intermediate routers will stuff with unsent packets and increase the RTT. If these delayed packets exceed the receiving timeout, receiver will send a duplicate ACK request back to sender. When sender receives three duplicate ACKs that means network congestion becomes worse, *cwnd* should be reduced for routers to process these queued packets, i.e., the flight packets. In Congestion Avoidance, most of TCP congestion control approaches significantly reduce *cwnd*, if sender receives three duplicate ACKs. For instance, *cwnd* is reduced to one in Taho and becomes half of original *cwnd* in Reno and NewReno. Since these *cwnd* reducing mechanisms are unsystematic and are independent on the available network bandwidth, which results in large variation of *cwnd* and low throughput.

Therefore, an expected network bandwidth based algorithm is proposed herein for reducing *cwnd* adaptively. In the case of receiving three duplicate ACKs in Congestion Avoidance, the network will become saturation and cause congestion seriously. For changing the network state from saturation to non-saturation, *cwnd* is set to the optimal *cwnd* that is the window size of the non-saturation state while with the same expected actual rate. In other words, the optimal *cwnd* is at the threshold between the saturation and non-saturation states. The optimal *cwnd* is computed by

$$Expected\ actural\ rate = \frac{cwnd}{RTT} = \frac{cwnd_{opt}}{RTT_{min}}, \tag{11}$$

that is,

$$Optimal\ cwnd(\text{or } cwnd_{opt}) = cwnd \cdot \frac{RTT_{min}}{RTT}. \tag{12}$$

Finally, the algorithm of reducing congestion window by the shrink factor while receiving three duplicate ACKs at sender is shown in Fig. 4.

if(three DUPACKs are received) {

$$ssthresh_{net} = cwnd \cdot \left(\frac{RTT_{min}}{RTT}\right)$$

$$cwnd_{opt} = ssthresh_{net}$$

}

if(timeoput expired) {

$$ssthresh_{net} = cwnd \cdot \left(\frac{RTT_{min}}{RTT}\right)$$

$$cwnd = 1$$

}

Fig. 4. The variation of congestion window in Slow Start

4 Numerical Results

This section evaluates the proposed TCP-Taichung approach for TCP congestion control in heterogeneous wireless networks by comparing various performance metrics, including goodput and fairness. Several compared approaches includes NewReno [10] and Westwood+ [17,18]. The Network Simulator (NS-2) [21] is adopted for all simulations with the following network parameters. The packet size is 512 bits. Router buffer is determined by Bandwidth-Delay Product (BDP) rule-of-thumb [22]. Impatient variant is adopted for responding partial ACKs for all TCP connections. In wireless networks, the uniform packet loss model is adopted at the sender and receiver nodes.

4.1 Scenario 1: *N* TCP Connections in a Wired Network

In scenario 1, total n (i.e., $n = 1,\ 2$ and 10) TCP connections are operated in a wired network with a bottleneck link, which bandwidth is 10 Mbps and delay is 25 ms. The other links except the bottleneck link are with the bandwidth of 100 Mbps and delay of 10ms as shown in Fig. 5. Two-type of simulations are evaluated including the cases without CBR and with CBR background traffic.

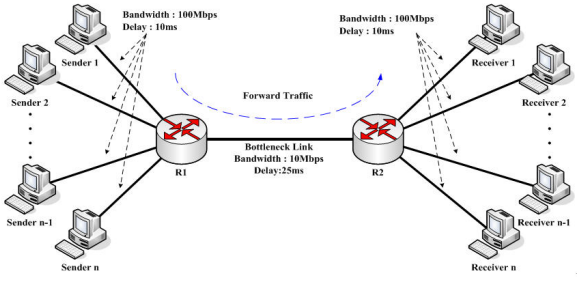


Fig. 5. Wired network with a single bottleneck link

Table 1. Goodput of compared approaches without CBR traffic (Wired network)

	Goodput (bps)		
Number of connections	TCP-Taichung	NewReno	Westwood+
1	9043681	8885084	8575426
2	4527145	4454523	4371722
10	901849	896581	893390

Table 2. Fairness of compared approaches without CBR traffic (Wired network)

	Fairness		
Number of connections	TCP-Taichung	NewReno	Westwood+
2	0.999965	0.99993674	0.99992643
10	0.999403	0.99953054	0.99726527

Table 3. Goodput of compared approaches with CBR traffic (Wired network)

	Goodput (bps)		
Number of connections	TCP-Taichung	NewReno	Westwood+
1	4286341	3978199	4164608
2	2174054	2120765	2171043
10	451027	450265	447422

Without CBR background traffic, the average goodput and fairness of TCP-Taichung, NewReno, and Westwood+ are demonstrated in Tables 1-2. In Table 1, TCP-Taichung yields the highest goodput under the number of connections is 1, 2, or 10. Meanwhile, Westwood+ yields the worst goodput among all compared approaches under different number of connections. In Table 2, all approaches yield very competitive fairness, in which the results are very close to one. $Fairness = 1$ means the fairest result. In Table 3, TCP-Taichung outperforms NewReno and Westwood+ in average goodput, in which Westwood+ yields the worst goodput. In addition, the goodput of all approaches decreases as the number of connections increases.

4.2 Scenario 2: *N* TCP Connections in a Wired Network with a Single Wireless Link

For evaluating goodput and fairness affected by network congestion and wireless error link, scenario 2 is simulated for all compared approaches. In Fig. 6, total 20 TCP connections are operated in a wired network with a single wireless network, in which bandwidth and delay of the wired bottleneck link is 10 Mbps and 25 ms, respectively. The packet loss rate of wireless links is between 0 and 0.1.

In Fig. 7, goodput of all approaches decreases as the wireless packet loss rate increases. The proposed approach yields the best goodput under all different wireless packet loss rate, but NewReno yields the worst goodput; especially, the

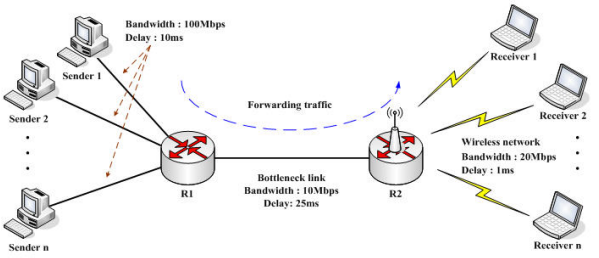


Fig. 6. 20 TCP connections in a wired network with a single wireless link

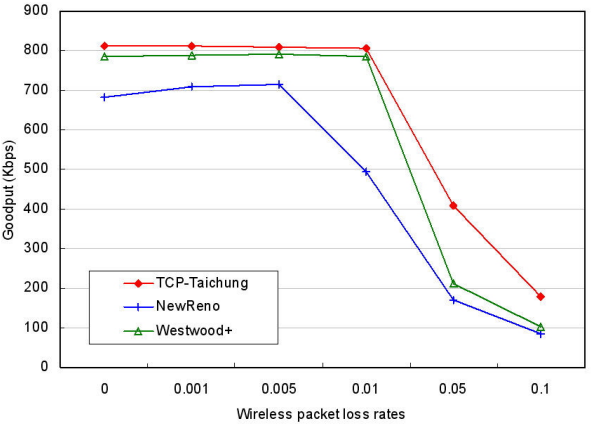


Fig. 7. Goodput of compared approaches (with a single wireless network)

Table 4. Fairness of compared approaches (with a single wireless network)

	Fairness		
Packet loss rate	TCP-Taichung	NewReno	Westwood+
0	0.988328	0.999261	0.923017
0.001	0.993481	0.996917	0.973306
0.005	0.992687	0.996184	0.991795
0.01	0.996392	0.99666	0.995311
0.05	0.994997	0.99684	0.996087
0.1	0.988709	0.984752	0.996548

goodput of the proposed approach is 210% higher than that of NewReno and 175% higher than that of Westwood+. Furthermore, fairness of all approaches under different wireless packet loss rate are evaluated as shown in Table 4. The proposed approach yields very competitive fairness to that of NewReno. Fairnesses of the proposed approach of different wireless packet loss rates are all above 0.9883.

In summary, the proposed approach results in the highest goodput not only in the wired network but also in the heterogeneous wired and wireless network. However, goodput of NewReno is better than that of Westwood+ under a wired network with CBR background traffic, but Westwood+ outperforms NewReno in goodput. Furthermore, all approaches yield very competitive fairness. In the wireless network, the *cwnd* variation of the proposed approach is more stable than that of NewReno.

5 Conclusions

In this paper, we proposed a RTT-based predictive bandwidth approach, namely TCP-Taichung, to determine the optimal congestion window, network *ssthresh*, and ideal *ssthresh* for TCP congestion control in heterogeneous wireless networks. Numerical results show that TCP-Taichung outperforms NewReno and Westwood+ in goodput and fairness in both of wired network and heterogeneous wireless networks.

Acknowledgments

This work was funded in part by National Science Council, Taiwan, ROC, under Grant NSC 95-2221-E-324-002 for B.-J. Chang.

References

1. IEEE 802.11 WG, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE std. 802.11 (1999)
2. IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE Std. 802.16 (October 2004)
3. http://www.3gpp.org
4. Santhi, K.R., Srivastava, V.K., SenthilKumaran, G., Butare, A.: Goals of true broad band's wireless next wave (4G-5G). IEEE VTC 2003-Fall 4, 2317–2321 (2003)
5. Internet Protocol, IETF RFC 791 (1981)
6. Transmission Control Protocol, IETF RFC 793 (1981)
7. Transmission Control Protocol, IETF RFC 2581 (1999)
8. Hoe, J.C.: Improving the start-up behavior of a congestion control scheme for TCP. In: ACM SIGCOMM 1996, pp. 270–280 (August 1996)
9. Chiu, D., Jain, R.: Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. Journal of Computer Networks and ISDN systems 17(1), 1–14 (1989)
10. The Newreno Modification to TCP's Fast Recovery Algorithm, IETF RFC 2582 (1999)
11. The Newreno Modification to TCP's Fast Recovery Algorithm, IETF RFC 3782 (2004)
12. Jacobson, V.: Congestion avoidance and control. In: ACM SIGCOMM 1988, pp. 314–329 (August 1988)

13. Stevens, W.: TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms, IETF RFC 2001 (1997)
14. Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M.Y., Wang, R.: TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In: ACM Mobicom 2001, pp. 287–297 (July 2001)
15. Wang, R., Valla, M., Sanadidi, M.Y., Ng, B.K.F., Gerla, M.: Efficiency/friendliness tradeoffs in TCP Westwood. IEEE ISCC 2002, 304–311 (July 2002)
16. Kai, X., Ye, T., Ansari, N.: TCP-Jersey for wireless IP communications. IEEE JSAIC 22(4), 747–756 (2004)
17. Grieco, L.A., Mascolo, S.: Performance evaluation of Westwood+ TCP over WLANs with local error control. In: IEEE LCN2003, pp. 440–448 (October 2003)
18. http://www-ictserv.poliba.it/mascolo/tcp%20westwood/modules.htm
19. Brakmo, L.S., O'Malley, S.W., Peterson, L.L.: TCP Vegas: new techniques for congestion detection and avoidance. In: ACM SIGCOMM, pp. 24–35 (1994)
20. Mo, J., La, R.J., Anantharam, V., Walrand, J.: Analysis and comparison of TCP Reno and Vegas. In: IEEE INFOCOM 1999, vol. 3, pp. 1556–1563 (1999)
21. ns-2 Network Simulator, http://www.isi.edu/nsnam/ns/
22. Villamizar, C., Song, C.: High Performance TCP in the ANSNET. ACM SIGCOMM Computer Communication Review 24(5), 45–60 (1994)

Dynamic Rate Adjustment (DRA) Algorithm for WiMAX Systems Supporting Multicast Video Services

Ray-Guang Cheng, Wei-Jun Wang, and Chang-Lueng Chu

Department of Electronic Engineering,
National Taiwan University of Science and Technology,
Taipei, Taiwan, R.O.C.
crg@mail.ntust.edu.tw

Abstract. This paper presents a dynamic rate adjustment (DRA) algorithm for WiMAX systems supporting multicast services. A scalable video coder with layered coding capability is assumed to be used to encode the multicast video information as a single base layer and multiple enhancement layers. The DRA algorithm first determines the portion of the base layer according to the QoS requirement of the multicast video service. It then dynamically adjusts the remaining portions of the enhancement layers to maximize the average throughput of the cell. In this paper, an analytical solution is presented to determine the best portions of the enhancement layers based on the estimated users' signal-to-noise-ratio (SNR). The accuracy of the analysis is verified via simulations. Simulation results indicate that DRA always achieves a higher average throughput than that of either uniform allocation algorithm or location-based allocation algorithm.

Keywords: multicast video services, scalable video coding, WiMAX.

1 Introduction

In recent years, broadband and wireless are two of the key technologies that lead to the remarkable growth of the telecommunications industry. Many industry observers believe that to combine the convenience of wireless with the rich performance of broadband will be the next frontier for growth in the industry. IEEE 802.16, which is also known as WiMAX, is one of the air interface standards designed for offering broadband wireless access services in a metropolitan area network (MAN) [1]. With WiMAX, end users are expected to be able to enjoy multimedia applications, such as real-time audio and video streaming, multimedia conferencing, and interactive gaming, in a more flexible manner (i.e., anytime and anywhere).

Currently, most of the network operators use their network resource to provide point-to-point services. However, there is a strong interest for them to offer multicast and broadcast services (MBS) over their broadband wireless access networks. The MBS allow unidirectional point-to-multipoint transmission of multimedia data (e.g. text, audio, picture, video) from a single source point to a multicast group in a multicast area. Normally, users are expected to be charged for subscribing these multimedia services and thus, they may demand for a certain level of quality of

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 379–388, 2007.

service (QoS). In MBS, there may be only two or hundreds of users that simultaneously subscribe the same service from the same base station (BS). Hence, from the profit point of view, the network operator may try to accommodate more users at the same time since a single copy of the MBS packets need to be transmitted. However, users at different locations may experience wireless channel errors at the same time. The network operator has to reserve extra radio resource in order to guarantee the QoS for users with bad channel condition. Therefore, from the spectral-efficiency point of view, the network operator may reject some service requests from users with bad channel conditions. Hence, one of the challenges for MBS is to achieve both high transmission efficiency and good scalability (with respect to number of users) [2].

The wireless channel error is characterized as bursty and location-dependent. Hence, users at different locations may observe different channel states at the same time [2]. In IEEE 802.16, techniques of automatic repeat request (ARQ) and adaptive modulation and coding (AMC) are supported to combat these wireless channel errors. ARQ is a packet re-transmission technique to achieve reliable data transmissions at the link layer. ARQ-enabled connections require each transmitted packet to be acknowledged by the receiver; unacknowledged packets are assumed to be lost and are retransmitted. In [2], the authors proposed an ARQ-based method to combat with wireless channel errors for video multicast service over WLAN. However, these methods are not applicable in WiMAX since IEEE 802.16 [1] does not support ARQ for their multicast connections. The ARQ-based approaches have the following disadvantages. First, either BS should reserve dedicated resource or the users may contend with each other to send negative acknowledgments (NACKs) in uplink. The reservation may waste network resource and the contention may cause unnecessary latency. Second, the sender may have to re-transmit the same packet multiple times if it receives multiple NACKs from different receivers. The implosion of NACK may reduce the transmission efficiency.

AMC is another effective mechanism to maximize throughput at the physical layer under a time-varying channel. IEEE 802.16 supports a number of modulations and forward error correction (FEC) coding schemes. IEEE 802.16 allows the AMC scheme to be changed on a per-user and per-frame basis based on the reported channel qualities [3]. The adaptation algorithm typically uses the highest modulation and coding scheme that can be supported by the signal-to-noise ratio (SNR) at the receiver such that each user is provided with the highest possible data rate in its respective link. AMC relies on instantaneous channel measurement of MS's uplink signal strength and thus, is not suitable for unidirectional multicast connections. With AMC, the BS shall reserve extra dedicate uplink resource for users to report their channel qualities. Even though, it is not easy for the BS to decide the best modulation and coding scheme for the multicast packets based on multiple channel feedbacks sent by users at different locations (i.e., a decision that favors users with good SNRs may not be proper for users with poor SNRs and vice versa).

Recently, many advanced video encoding techniques have been developed. Among them, the scalable video coding offers the users with capability of reconstructing lower resolution or lower quality signals from partial bit streams. This allows network providers with simple solutions in adaptation to network and terminal capabilities [3]. For example, MPEG-2 has implemented the layered coding, where video information is encoded as a single base layer (BL) and multiple enhancement layers (ELs). The

standalone availability of enhancement information (without the BL) is useless, because differential encoding is performed with reference to the BL [4]. With scalable video coding, the network operators may utilize different AMC modes to protect the BL and ELs of the multicast video. Therefore, we may guarantee a minimum level of video quality for users with poor SNRs while offering better video quality for users with good SNRs.

This paper focuses on the radio resource allocation issue of wireless video multicast services adapting scalable video coding. A dynamic rate adjustment (DRA) algorithm is proposed to determine the best modulation and coding scheme for transmitting the BL and EL(s) portions of the multicast video. The rest of the paper is organized as follows. The system model adopted by this paper is described in Section 2. Section 3 presents the details of the proposed DRA algorithm. Section 4 presents the simulation results. Conclusions and future work are finally drawn in Section 5.

2 System Model

IEEE 802.16 standard families support several physical layers. For operational frequencies between 10-66 GHz, the physical layer of single-carrier modulation (SC) is supported. For frequencies below 11GHz, where propagation without a direct line-of-sight (LOS) must be accommodated, three alternatives physical layers are provided: single-carrier modulation (SCa), frequency-division multiplexing (OFDM), orthogonal frequency-division multiple access (OFDMA). In this paper, OFDM physical layer is utilized as an example and the results can be easily extended to other physical layers. Table 1 summarizes the SNR requirements and the data rates for various combinations of modulation and coding scheme in the IEEE 802.16 OFDM physical layer [1].

Table 1. The SNR requirements and the data rates supported by IEEE 802.16 OFDM

	Coding rate	SNR requirement (dB)	Data rate (Mbps)
BPSK	1/2	6.4	2.5
QPSK	1/2	9.4	5
QPSK	3/4	11.2	7.5
16QAM	1/2	16.4	10
16QAM	3/4	18.2	15
64QAM	2/3	22.7	20
64QAM	3/4	24.4	22.5

The system model adopted in this paper is shown in Fig. 1, in which a single cell comprising one BS and multiple subscriber stations (SSs) is considered. In this paper, n modes of modulation and coding schemes of the OFDM physical layer, which are referred as AMC modes herein, are used to transmit the layered coding of multicast video packets as an example. In this paper, $n = 4$ is assumed and the chosen AMC

modes include BPSK with coding rate 1/2; QPSK with coding rate 1/2; 16QAM with coding rate 1/2, and 64QAM with coding rate 2/3. Conceptually, the cell can be divided into *n* non-overlapping areas based on the SNR requirements of the four AMC modes. Let the number of SSs that use AMC modes of 64QAM, 16QAM, QPSK, and BPSK be N_1, N_2, N_3, and N_4, respectively. Note that, N_1, N_2, N_3, and N_4 can be estimated based on the signal qualities of SSs measured during periodic ranging.

Let θ be the percentage of BL in the encoded multicast video. Note that, θ can be determined, for example, based on the basic QoS requirement of the perceived multicast video. It is assumed that BS uses the most robust AMC mode (i.e., BPSK) to transmit the BL portion of the multicast video such that the BL can be correctly received by all SSs in the cell. This assumption ensures that basic QoS of all SSs in the cell can be guaranteed. It is further assumed that the EL generated by the scalable video coder contains (*n*-1) portions and the percentage of each portion can be adjusted by BS during runtime [5]. Let α, β, and γ be the percentage of the three portions of EL, respectively, as illustrated in Fig. 2. Note that, $\alpha + \beta + \gamma = 1-\theta$ and the standalone availability of higher part of the EL (e.g., α portion) is useless because differential encoding is performed with reference to the lower parts of the ELs (i.e., β, and γ portions) [4]. In this paper, BS shall transmit the α, β, and γ portions of EL using AMC modes of 64QAM, 16QAM, and QPSK, respectively.

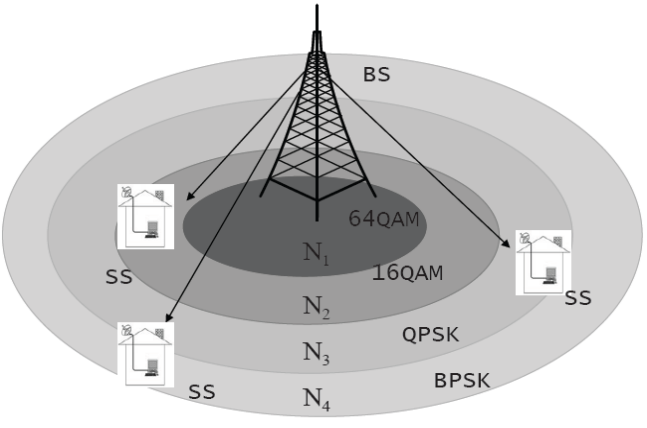


Fig. 1. Multicast system architecture

In this paper, the *average throughput* and *the net profit of the cell* are chosen to be the two key performance indexes for a multicast video service. The definition of the two parameters is given as below:

$$\text{Average throughput} = \frac{D_R}{T \times N_S} = \frac{\left(\sum_{i=0}^{N_S} (P_B D_T + P_{E,i} D_T) \right)}{T \times N_S}, \tag{1}$$

and

$$\text{Net profit} = \text{Total revenue - Total cost} = \left(\sum_{i=0}^{N_S} (P_B D_T + P_{E,i} D_T) \right) \times C_d - TC_a, \tag{2}$$

where D_R is total amount of data received by all SSs; T is total transmission time required to transmit the BL and EL of the multicast video; N_S is the total number of SSs (i.e., $N_S=N_1+N_2+N_3+N_4$); $P_B D_T$ is the percentage of the data coded by BL, which is common for all SSs; $P_{E,i}$ is the EL percentage received by the i-th SS; D_T is total transmission data amount; C_d is the profit received for a unit data volume of the multicast video, and C_a is the cost required for a unit air time.

The main focus of the paper is to maximize the average throughput and the total profit of the cell by adjusting α, β, and γ.

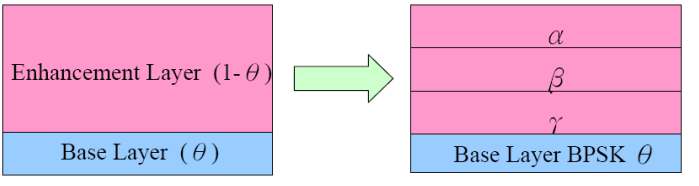


Fig. 2. Multicast system architecture

3 DRA

In this section, the proposed DRA algorithm is introduced.

The average throughput of the cell defined in Eq. (1) can be written as a function of α, β, and γ, which is given by

$$\text{Average throughput} = \frac{\theta(N_1+N_2+N_3+N_4)+\gamma(N_1+N_2+N_3)+\beta(N_1+N_2)+\alpha N_1}{(\frac{\theta}{2.5}+\frac{\gamma}{5}+\frac{\beta}{10}+\frac{\alpha}{20})\times(N_1+N_2+N_3+N_4)}. \tag{3}$$

The maximization of Eq. (3) is a linear fractional programming problem [6] for given constants N_1,N_2, N_3, N_4, and θ. By theory of linear programming, if optimal solution of linear programming exists, optimal solution might be on the boundary of feasible region (especially on the apex of feasible region) [7], as shown in Fig. 3. The boundary of feasible region is given by the equations of $\alpha+\beta=1-\theta$, $\alpha+\gamma=1-\theta$, and $\beta+\gamma=1-\theta$. It can be shown that extreme value of Eq. (3) is given by

$$\begin{cases} \alpha=0, \beta=0, \gamma=1-\theta, \text{ if } \theta > \dfrac{N_1+N_2-N_3}{2N_3-N_4}, \\ \alpha=0, \beta=1-\theta, \gamma=0, \text{ if } \dfrac{N_1-N_2}{6N_2-N_3-N_4} < \theta < \dfrac{N_1+N_2-N_3}{2N_3-N_4}, \\ \alpha=1-\theta, \beta=0, \gamma=0, \text{ if } \dfrac{N_1-N_2}{6N_2-N_3-N_4} > \theta. \end{cases} \tag{4}$$

Normally, the net profit defined in Eq. (2) will be maximized when the average throughput of the cell reaches its maximum. However, the net profit could become negative if the air time cost factor C_a is relatively higher than that of the subscription fee factor C_d (e.g., the cell is almost fully loaded). In this case, the best strategy is to send BL only. Figure 4 shows that case that the net profit may become negative when the air time cost is relatively high or the total number of SSs subscribing the multicast video service is relatively small.

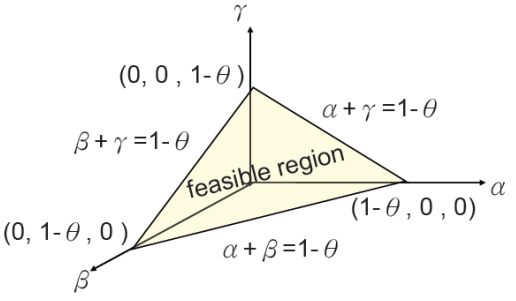


Fig. 3. Feasible region

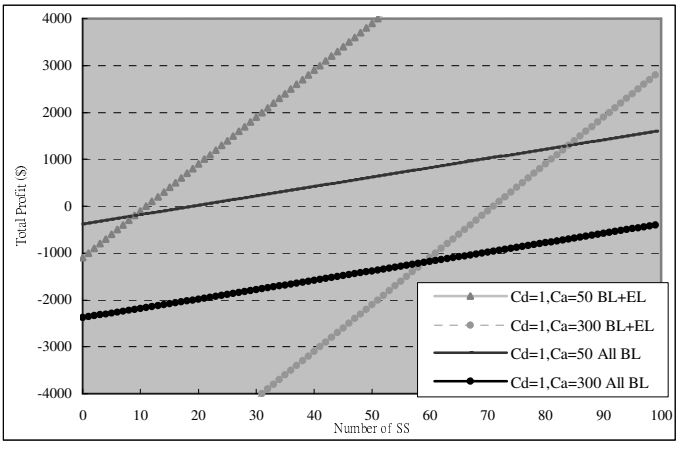


Fig. 4. Net profit with various number of SSs

The proposed DRA algorithm is executed as follows. Initially, the BS has to determine the BL percentage of the multicast video, θ, based on basic QoS requirement of the given video multicast service. The BS then estimates the number of SSs in each non-overlapping area, N_1, N_2, N_3, and N_4, according to the SNR estimated during network entry. With N_1, N_2, N_3, and N_4, the BS will determine three portions of EL, α, β, and γ,. according to Eq.(4) and transmit these portions using AMC modes of 64QAM, 16QAM, and QPSK, respectively. Note that, the BS may decide not to transmit the EL part if the air time cost is relatively high. In other words, Eq.(2) must be kept non-negative.

4 Simulation Results

In this section, the numerical analysis is first verified by the simulation and the effectiveness of the proposed algorithm is illustrated. In the simulation, a single cell with radius r = 3.4 km was assumed. A transmission bandwidth of 6 MHz operating in 3.5 GHz band was investigated. The channel model of ECC-33 was used, in which the BS transmission power of 43 dBm; BS antenna height of 17 m, and SS antenna height of 10 m, were assumed [8]. With these setting, the path loss from the BS to a given SS can be calculated and the SNR experienced by the SS can be easily obtained. The population of N_1, N_2, N_3, and N_4 can then determined according to the SNR requirement of the four AMC modes. In the following examples, the simulation results of the proposed DRA algorithm are all coincided with the numerical analysis.

In the first simulation, SSs were uniformly distributed within the cell. Figure 5 shows the average throughput of the cell using different rate assignment algorithms for θ =0.2. Two rate assignment algorithms were chosen as the benchmarks. The first algorithm, which is referred as the uniform allocation algorithm herein, assigns the portions of EL uniformly. That is, $\alpha = \beta = \gamma = (1-\theta)/3$. The second algorithm, which is referred as the location-based allocation algorithm, assigns the portions of EL based on the number of SSs in the non-overlapping areas, which gives $\alpha = \frac{N_1(1-\theta)}{N_1+N_2+N_3}$, $\beta = \frac{N_2(1-\theta)}{N_1+N_2+N_3}$, and $\gamma = \frac{N_3(1-\theta)}{N_1+N_2+N_3}$. From Eq. (4), it can be found that the average throughput of the cell reaches its maximum when $\alpha = 0, \beta = 0, \gamma = 1-\theta$. That it, BS shall transmit the BL and EL with BPSK and

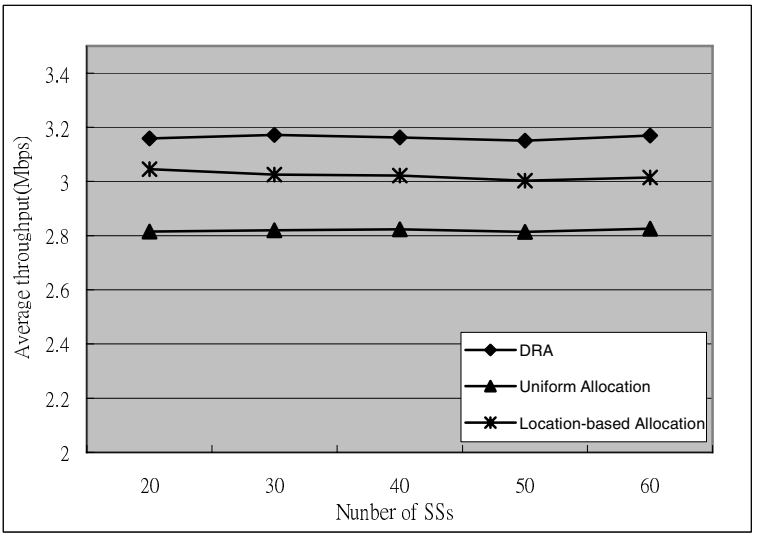


Fig. 5. The average throughput of the cell for uniformly distributed SSs

QPSK, respectively. It is shown in Fig. 5 that DRA achieves the maximum average throughput for different N_s, which verifies the numerical analysis.

Figure 6 shows the case that the distance between each SS and BS is uniformly distributed for θ =0.3. It can also be found that DRA achieves the highest average throughput of the cell compared with uniform allocation and location-based allocation algorithms.

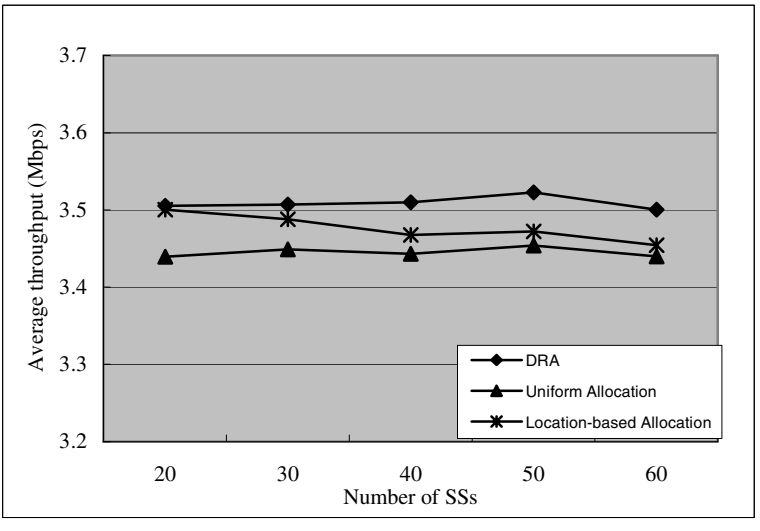


Fig. 6. The average throughput of the cell for uniformly distributed distance

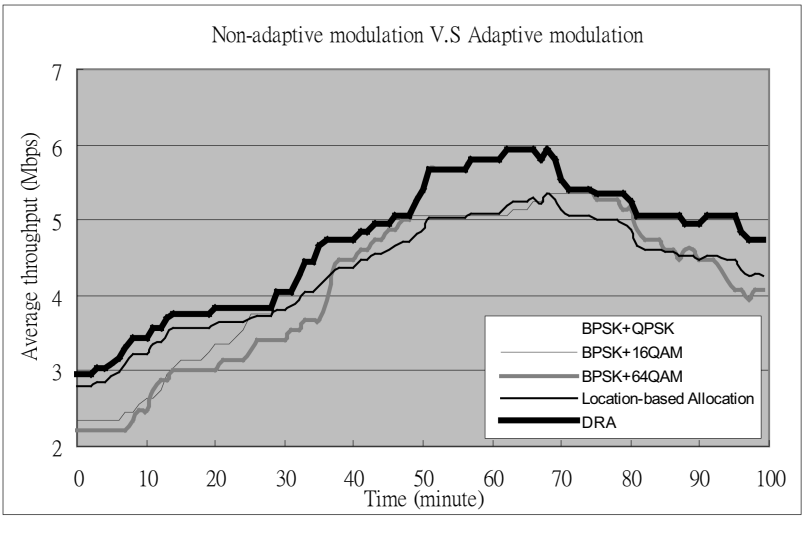


Fig. 7. Average throughput of the cell for mobile SSs

Although the DRA algorithm is designed for fixed SSs, the results can be extended to accommodate mobile SSs assuming the location of SS can be monitored by BS through, for example, periodic ranging. The BS will utilize DRA to dynamically adjust the AMC mode based on the estimated N_1, N_2, N_3, and N_4 obtained in each reporting interval. The simulation result is illustrated in Fig. 7. In the simulation, each SS was assumed to move toward BS in first 50 minutes and then moves away from BS in next 50 minutes. The total simulation time was 100 minutes. Initially, Ns was set to 50 and is uniformly distributed in the cell; the velocity of each SS was uniformly distributed from 0 to 1 m/s. θ= 0.2 and the periodic ranging interval of 1 minute were assumed. In Fig. 7, the dotted line represents the AMC mode decided by DRA. It can be found that the proposed DRA algorithm can always achieve the maximum throughput regardless of the location distribution of SS. In the figure, DRA chooses BPSK+QPSK as the best AMC mode at the beginning since the number of SS is uniformly distributed within the cell. As time went by, the SS moved toward BS, which resulted in increased N_1 and reduced N_4. Hence, the best AMC mode changes to BPSK+64QAM. After 50 minutes, all SSs moved away from BS, which changed the AMC mode to be BPSK+16QAM and then BPSK+QPSK, consequently.

5 Conclusion

This work presented the DRA for WiMAX systems supporting video multicast services. A layered coding scheme is chosen to encode the multicast video information into one BL and three ELs. The portion of the BL is suggested to be determined based on the QoS requirement of the multicast services. A DRA algorithm is then proposed to allocate the three portions of EL in order to maximize the average throughout of the cell. An analytical method is presented to calculate the three portions of ELs and the accuracy of the analysis is verified via simulations. Simulation results indicate that DRA achieves a higher average throughput than that of either location-based allocation algorithm or uniform allocation algorithm.

Acknowledgments. This work was supported in part by National Science Council, Taiwan, under Contract No. NSC 96-2219-E-011-005, 96-2219-E-011-007, and 95-2218-E-011-008, and by BenQ Inc. under Contract No. 0247.

References

1. IEEE 802.16 Std.: IEEE standard for local and metropolitan networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems (2004)
2. Xu, D., Li, B., Nahrstedt, K.: QoS-directed error control of video multicast in wireless networks. In: IEEE International Conference on Computer Communications and Networks, pp. 257–262 (1999)
3. Andrews, J.G., Ghosh, A., Muhamed, R.: Fundamental of WiMAX, understanding broadband wireless networking. Prentice Hall, Englewood Cliffs (2007)
4. Ohm, J.R.: Advances in scalable video coding. Proceedings of the IEEE 93(1), 42–58 (2005)

5. Zhuo, L., Lam, K.M., Shen, L.: Adaptive forward error correction for streaming stored MPEG-4 FGS video over wireless channel. In: IEEE Workshop on Signal Processing Advances in Wireless Communications, pp. 26–30 (2004)
6. Hiller, F.S., Lieberman, G.J.: Introduction to operation research, 7th edn. McGraw-Hill, New York (2001)
7. Ignizio, J.P., Cavalier, T.M.: Linear programming, 1st edn. Prentice Hall, Englewood Cliffs (1993)
8. Abhayawardhana, V.S., Wassell, I.J., Crosby, D., Sellars, M.P., Brown, M.G.: Comparison of empirical propagation path loss models for fixed wireless access systems. In: IEEE Vehicular Technology Conference (VTC), pp. 73–377 (2005)

Efficient and Load-Balance Overlay Multicast Scheme with Path Diversity for Video Streaming

Chao-Lieh Chen[1], Jeng-Wei Lee[2], Jia-Ming Yang[2], and Yau-Hwang Kuo[2]

[1] Department of Electronic Engineering,
Kun-Shan University, Yung-Kang, Tainan County, Taiwan
frederic@ieee.org
[2] Department of Computer Science and Information Engineering
National Cheng Kung University, Tainan City, Taiwan
{lijw,abby,kuoyh}@cad.csie.ncku.edu.tw

Abstract. An overlay multicast is proposed to solve the scalability and deployment problems in IP Multicast. We propose a scheme, Topology-aware Load-balance Hierarchical Independent Tree (TLHIT), with topology-aware, load-balance and path diversity properties to improve the performance of overlay multicast. Compared to traditional methods, the proposed TLHIT constructs not only node-disjoint but also path-disjoint multicast trees where each node serves as an interior node in only one tree and different trees do not contain the same path. Moreover, TLHIT ensures load-balance property by building the multicast trees based on *n*-ary full tree. It ensures that each node serves almost the same amount of child nodes. Simulation results show that the reliability, efficiency, and load-balance properties of the proposed TLHIT are assured.

Keywords: overlay multicast, topology-aware, load-balance, hierarchical independent tree, node-disjoint, path-disjoint.

1 Introduction

With the rapid growth of internet technology, more and more one-to-many transmission services are developed including video streaming, distributed simulations, video-conferencing, multi-party games, content distribution, and so on. Thus, IP multicast at the network layer has been proposed for realization of these services. However, it has not been widely deployed yet because of high cost to upgrade the network infrastructure. Recently, overlay multicast is proposed as an alternative to provide the multicast services. In this way, the participating nodes organize themselves into an overlay structure and the efficiency of the overlay can be optimized by adapting to network dynamics and considering application level performance.

In overlay network, each participating node has potentially multiple paths to communicate with another node. Therefore, multi-tree multicast [1][2][3] is proposed to improve the fault-tolerance if compared to single-tree multicast [4][5]. However, how to use these trees more efficiently is still an open problem. Hence, multi-tree multicast with path diversity in overlay network attracts lots of interest in recent

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 389–399, 2007.

years. The topology-aware hierarchical arrangement graph (THAG) [1] constructs multi-tree multicast applications with diverse paths. In THAG, all participating nodes are divided into a number of arrangement graphs and several node-disjoint multicast trees are embedded in each arrangement graph. Node-disjoint trees mean that any node serves as interior node in only one tree. Though THAG constructs node-disjoint trees, it leads to unbalance load problem because each node is responsible for handling traffics to different number of child nodes, especially the source node. The situation gets worse as the growth of multicast group members.

In this paper, we propose a load-balance scheme called Topology-aware Load-balance Hierarchical Independent Tree (TLHIT) scheme which construct a virtual graph (VG) at first, and the multicast trees are embedded in VG based on n-ary full tree. Therefore, each node in TLHIT serves almost the same number of child nodes. Moreover, the multicast trees in TLHIT are independent, where independent means that each tree is both node-disjoint and path-disjoint. Hence, the load-balance and fault-tolerant ability are further improved in TLHIT. Moreover, when the number of multicast group members is larger than the capacity of the constructed VG, TLHIT extends the original VG into a larger one by duplicating several child VGs and assembling these VGs into hierarchical structural. As a node joining the multicast service, TLHIT selects a suitable position in the VG not only in accordance with the network conditions but also keeps all the nodes in TLHIT with balance load.

The rest of the paper is organized as follows. In Section 2, background and relative work is introduced. In Section 3, we explain how to design the TLHIT. The simulation results are shown in Section 4 and the reliability, efficiency, and load-balance of THAG and TLHIT are compared in several simulations. The conclusions are drawn in Section 5.

2 Background and Related Work

In this section, we present some background knowledge about multi-tree and application-layer multicast streaming. They are multi-path streaming, multiple descryiption coding (MDC) and topology-aware multicast streaming. Each mechanism is implemented in application layer.

2.1 Multi-path Streaming

Unlike traditional methods which embed redundant bits into each packet to provide error-correction ability, the concept of multi-path is to transmit data through different paths and provide path diversity to avoid data sharing the same congested or troublesome interior nodes or links. It prevents multimedia data from burst error and degraded quality significantly. There are two major multi-path mechanisms -- relay server [6][7] and overlay network [8][9].

- Relay server: For path-diversity, a relay server is responsible for relaying data to a destination. As shown in Figure 1, when node A wants to transmit data to node B, node A transmits partial data to the destination via a relay server indirectly and the other data to destination directly. However, this mechanism has some disadvantages. First, the performance of a relay server is limited and there is a tradeoff between cost and performance. Second, good deployment of relay servers is necessary and it affects overall system performance.

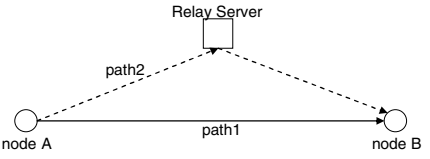


Fig. 1. Achieving path diversity using relay server

- Overlay network: Overlay network is built on top of another network. Each node in overlay network communicates with another node by virtual or logical links which correspond to a direct link or many physical links in the underlying network. It means that the source node has potentially multiple paths to communicate with all the other participating nodes through directly link or other relay nodes. Path diversity can be obtained by a good choice of relay nodes. However, communication between pair-wise nodes bypassing others potentially increases latency. The proposed TLHIT is based on overlay network and focus on how to achieve path diversity with acceptable latency.

2.2 Multiple Description Coding (MDC)

Multiple Description Coding (MDC) [10] is utilized when a media stream needs to be separated into several parts and transmitted in each multicast tree. In MDC, a media stream is encoded into several parts referred to as descriptions. Any combination of received descriptions can be used to decode the original media stream with acceptable quality. Media quality is improved as the number of received descriptions increases and the best media quality is obtained when all the descriptions are received.

2.3 Topology-Aware Hierarchical Arrangement Graph (THAG)

There are some related works on application-layer multicast for media streaming such as THAG [1], which embeds multicast trees in arrangement graphs and provides node-disjoint characteristics in each tree. In THAG, each node serves as interior node which is responsible for forwarding data to other nodes in exactly one multicast tree. Thus, the influence of any node failure is minimized and the fault-tolerance is improved. Figure 2 shows an example of arrangement graphs.

However, THAG does not consider the unbalance load problem of each node. As shown in Figure 2, the load of the source node is much heavier than all the other nodes. And the situation gets worse as the growth of the arrangement graph. Moreover, THAG only guarantees the interior nodes in each multicast tree are different, but it may share the same congested or troublesome path and deteriorate the system performance. Hence, in this paper, we propose a mechanism to build multicast trees with node-disjoint, path-disjoint and load-balance properties.

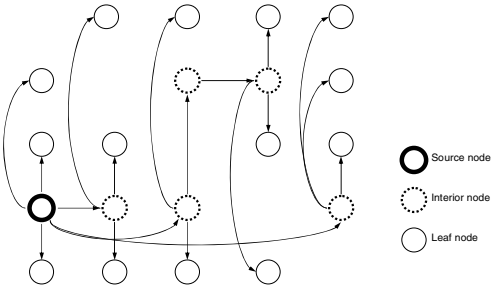


Fig. 2. One multicast tree in THAG

3 Topology-Aware Load-Balance Hierarchical Independent Tree (TLHIT)

In TLHIT, a virtual graph (VG) with n independent multicast trees is built. Independent trees mean that each participating node only acts as the interior node (i.e. non-leaf node) in one of n multicast trees and paths in each tree are distinct. Based on the structure of the VG, as one member joining the multicast group, it selects an appropriate vacant position in VG according to its network conditions. Further, TLHIT extends original VG into hierarchical structure by duplicating several VGs to support more multicast members.

3.1 Independent Multicast Trees in Virtual Graph

To enhance the performance of multi-tree multicast applications, TLHIT considers the following three requirements as constructing a VG:

- *Node-disjoint*: To mitigate the influence of node failure or leaving, we must make sure each participating node only acts as the interior node one time. It means that if one node is selected to be an interior node in one tree, it must be a leaf node in all the other trees.
- *Path-disjoint*: Transmission in the same path may result in path congestion or high relation of loss behavior [11], which increases end-to-end delay and reduces the performance of the MDC streaming. Hence, the path-disjoint is considered in our VG.
- *Load-balance*: In practice, the resources of a node are limited (e.g. computation capability and network bandwidth) and unbalanced load may affect scalability of the multi-tree multicast applications. Hence, the algorithm should achieve load-balance.

For satisfying the three conditions, TLHIT constructs a VG which contains n multicast trees. The parameter n is user-defined and can be decided by the number of descriptions in the MDC method. At first, the nodes in VG are separated into n root sets. All nodes in a root set are organized into an n-ary full tree and leaf nodes are responsible for connecting the other root sets. Each root set forms a multicast tree.

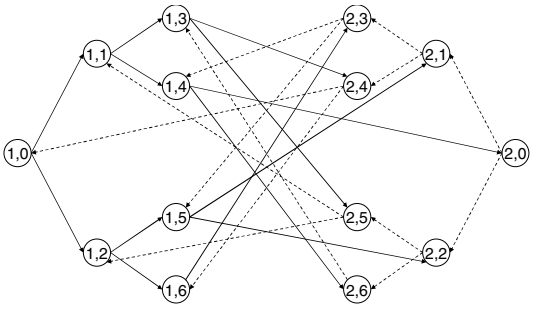


Fig. 3. VG with two multicast trees

Figure 3 illustrates a construction of the VG with two root sets. The solid and dotted lines represent different multicast trees. If the media stream is fragmented into n descriptions, i-th description is transmitted by i-th root set. Hence, via this simple concept, TLHIT guarantees that each node transmits only one description and receives the other descriptions from the other root sets and therefore the first requirement, node-disjoint, is achieved. The topology generate algorithm is described below. To meet the path-disjoint requirement, TLHIT must guarantee that each description is transmitted in different link. Denote $N_{s,i}$ as the node with a pseudo-address i in the s-th root set. Line 6 to line 17 show that when the number of child of leaf node $N_{s,i}$ in root set s is less than n, $N_{s,i}$ chooses an unselected node $N_{m,j}$ as its child node. Line 8 ensures $N_{m,j}$ not to choose $N_{s,i}$ as his child node when m-th set becomes root set. Therefore, path-disjoin is achieved. Moreover, the multicast trees are based on n-ary full tree such that the interior nodes in each tree are responsible for the same number of child nodes. Line 7 and line 14 ensure that each node have at most n child nodes. Therefore, TLHIT satisfies the requirement of load-balance. Line 4 ensures each participating nodes to be selected in each multicast tree. The double-slashes are remarks.

```
0 Algorithm Topology-generate
1 Input: n root sets 1, ..., n; //Each of which contains k nodes
  (k=1+n+n^2). The nodes in each root set are organized into a n-
  ary full tree.
2 Input: Node addresses N_{s,j}; // Each node is given a pseudo-
  address where s is the root set the node belongs to and j is
  the node ID. The pseudo-address of root node is N_{s,0}. The
  pseudo-addresses of child nodes of the parent node N_{s,j} are
  N_{s,n*j+1}, N_{s,n*j+2}, …, and N_{s,n*j+n}.
3 For each root set s= 1, 2, …, n do {
4     Mark all the nodes as unselected except the nodes in root
    set s;
5     While there is any node marked as unselect do {
6         For each leaf node N_{s,i}, i=n+1, n+2, …,n^2+n, in root set
        s, do {
7             While number of child of N_{s,i} ≤ n, do {
8                 j = (i+1)%k; //modulus %
9                 For m=1, 2, …, n, m≠s, do {
10                    If N_{m,j} is marked as unselect, then {
```

```
11                          s = s ∩ N_{m,j}; // N_{m,j} chooses Node N_{s,i}
                                              as parent
12                          Marked N_{m,j} as selected;
13                      }
14                      If number of child of N_{s,i} ≥ n, then
15                          Break;
16                  }
17              } // number of child of N_{s,i} ≥ n
18          } // end for each leaf node
19      } // each node is selected
20 } // end for each s
```

3.2 Extending Virtual Graph to Hierarchical Structure

This section describes how to extend the VG into a hierarchical structure for accommodating more multicast group members. As presented above, the capacity of VG is very limited. Typically, there are at most 14 nodes in the VG with two multicast trees. Hence, TLHIT extends the original VG into a large one when the number of multicast group members is larger than the capacity of the VG. In addition to the tree requirements discussed above, the extended VG must also remain the structure of serving nodes unchanged. This ensures multimedia services are not affected by the extending process.

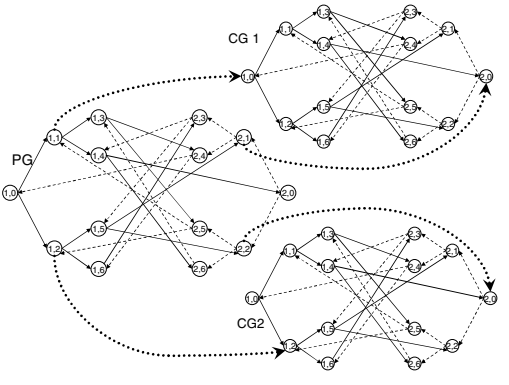


Fig. 4. Hierarchical structure with two root sets

When an "over-capacity" event is triggered, TLHIT duplicates several child VGs (short for CGs) and connects the CGs to the original parent VG (short for PG). The path-disjoint requirement is still retained as the CGs inherits the characteristics of the PG and each leaf node does not have direct connection to each other. Hence, the remaining problem is how to retain the node-disjoint and load-balance properties. The problem can be reduced to determinations of the number of CGs to duplicate and the optimal source nodes positions in CGs. In the PG, let $n_{nonroot}^{(\min)}$ be the total number of nodes satisfying the two conditions that first having connections to any CGs and second having minimum hop count to their root. Then, the number of CGs to

duplicate is $n_{CGs} = \frac{n_{nonroot}^{(\min)}}{n}$. Figure 4 shows an example of hierarchical structure with two root sets. The nodes $N_{1,1}$, $N_{1,2}$, $N_{2,1}$ and $N_{2,2}$ in PG satisfy the two conditions to become the source node of CGs. Hence, when the original VG is full-filled, two CGs are duplicated. If another "over-capacity" event is triggered again, 8 CGs will be created since there are 16 nodes satisfying the two conditions (i.e. $N_{1,3}$, $N_{1,4}$, $N_{1,5}$, $N_{1,6}$, $N_{2,3}$, $N_{2,4}$, $N_{2,5}$ and $N_{2,6}$ in PG, $N_{1,1}$, $N_{1,2}$, $N_{2,1}$ and $N_{2,2}$ in CG1, $N_{1,1}$, $N_{1,2}$, $N_{2,1}$ and $N_{2,2}$ in CG 2). Node-disjoint is assured since each source node of a CG belongs to a different root set in PG.

3.3 Member Joining to Virtual Graph

When a new member joins a multicast group, TLHIT assigns it to an appropriate vacant position in the VG according to end-to-end delays between the joining node and source nodes and ensures neighbor nodes are either in the same VG or in the CGs produced by the VG. But the ancestor of the vacant position is responsible for all its data transmission jobs. Hence, inappropriate member join will cause unbalanced load. Considering the tradeoff between the end-to-end delay and load balance, we divide the member joining into two states called *the locating* and *the replacing states*. During the locating state, a member is assigned to a vacant position according to loading situation while during the replacing state each node periodically detects the network condition and adjusts its position in the VG to enhance overall system performance.

The member join algorithm, MemberJoin(v_i, G), is described below. Suppose the node v_i is joining the multicast group. Let G denotes the original VG or one of the CGs closest to v_i, and s is the closest source node in G.

```
0 Algorithm MemberJoin(v_i, G)
1 Input: v_i; //the new join node.
2 Input: G; //one of the VGs which is closest to v_i
3 If there is any vacant position in G, then {//Locating state
4     v_i joins G by replacing a specific vacant position;
5     return;
6 } //end Locating state
7 Else {//Replacing state
8     Calculates end-to-end delay D(v_i, s);
9     For each v_j in G do {
10         If D(v_j, s) < D(v_i, s) then {
11        v_i = v_j; // v_j replaces v_i and joins G;
12        } //end delay comparison
13    } //end for each V_j
14    find G' which is a CG of G closest to v_i;
15    MemberJoin(v_j, G'); //recursive
16 } //end Replacing state
```

As shown in the member join algorithm above, line 4 to line 6 refers to the joining procedure when v_i is in the locating state that it does not belong to any VG. This situation may happen either when the first time this node joins the multicast group or when another member node has shorter end-to-end latency to the source node than it. During locating state, the v_i searches for a root set in G with maximum number of vacant positions. Then, a member is assigned to the vacant position closest to the

source of the root set. As shown starting from line 7, when VG contains no vacant position, v_i enters into replacing state. During replacing state, v_i calculates the end-to-end delay D(v_i, s) and compares to D(v_j, s) of each node j in the root set of s. As shown in line 9 to line 13, if a node v_j having D(v_j, s) larger than D(v_i, s), v_j is replaced with v_i. Finally, v_j enters locating state and searches for another vacant position. Otherwise, as shown in lines 14 to 15, the node v_i joins G' which is a CG of G closest to it. Because the multicast group members may change as time goes by. Each node in TLHIT runs member join algorithm periodically to see whether there exist a better position or not. Thus, we can ensure each node is always in proper position in TLHIT.

4 Simulation Results

In this section, the metrics of evaluating the performance of THAG and TLHIT are described as follows:

- *Average received descriptions (ARD)*: ARD represents the average number of descriptions received by the nodes. Video quality improves as the number of received descriptions increases. Hence, ARD represents not only fault-tolerant but also QoS of a system.
- *Stretch*: stretch presents the average number of interior nodes from the source to each participating node in overlay multicast trees. The stretch shows propagation delay in TLHIT comparing to the unicast case.
- *Stress*: stress represents the number of descriptions a node needs to forward. This metric shows whether this system is load-balance or not.
- *Delay Distribution*: this metric shows the difference of propagation delay of each node.

In the simulations, both THAG and TLHIT construct two virtual graphs with 1750 nodes, respectively. A virtual graph is constructed according to G_6 and s trees are embedded where s varies from 2 to 6. The value of propagation delay between any pair of nodes is randomly generated ranging from 20ms to 120ms. We use the same topology in THAG and TLHIT.

4.1 Average Received Descriptions and Stretch

Figure 5 presents ARD v.s. the number of trees with different node failure probabilities 2%, 5%, and 10%. The simulation results represents that the nodes using TLHIT receives more descriptions than those using THAG and it means the fault-tolerant of TLHIT is better than THAG. Hence, the multicast group member in TLHIT gets a better video quality. Furthermore, only the first source node needs to execute the topology-generation algorithm in TLHIT. The member join algorithm of TLHIT is based on the same algorithm of THAG. Hence, the complexity of TLHIT and THAG system is the same, but the performance of TLHIT is better than THAG. Straight lines also indicate that the TLHIT are not affected by the number of trees. Because of the node-disjoint property, the TLHIT will not generate enormous number of losses when node failure occurs. And then we compare the stretch in TLHIT and THAG in cases of different number of nodes in the system. As shown in Figure 6, the stretch of TLHIT is smaller than that of THAG. The result means that the data delivery in TLHIT has shorter latency than that in THAG.

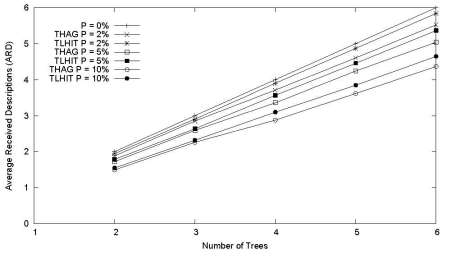


Fig. 5. Comparison of average received descriptions in THAG and TLHIT when group size = 1750

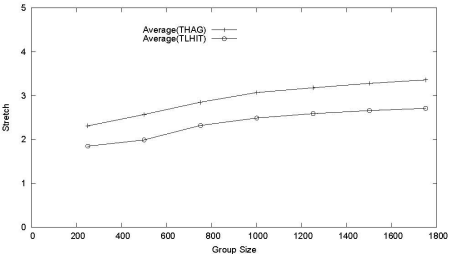


Fig. 6. Stretch versus group size, s = 6

4.2 Stress and Delay Distribution

As shown in Figure 7, 23% of the group members in THAG forward less than five descriptions which are nearly idle and 12% of the group members need to forward more than ten descriptions. It means that the duty of data forwarding for each group member is unfair in THAG. In TLHIT, 94% of the group members forward the same number of descriptions. The result indicates that the load-balance of TLHIT is assured.

In THAG and TLHIT systems, one description is transmitted to each group member through different number of interior nodes. So, each group member will experience different latency transmitting this description. The delay distribution represents the difference of propagation delays among all group members. As shown in Figure 8, the variance of propagation delay in THAG is much larger than in TLHIT. In TLHIT, the propagation delay is centralized from 200ms to 300ms. Only a few members' delay value is greater than 450ms. On the contrary, the maximum propagation delay is up to 550ms and the variance of the delay distribution is large in THAG. Figure 8 also shows the advantage of load-balance.

Fig. 7. Stress distributions of THAT and TLHIT when s = 6 and group size = 1750

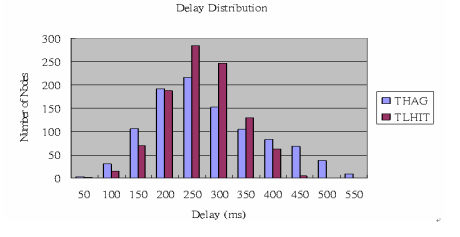


Fig. 8. Delay distribution in THAG and TLHIT, s = 6, group size = 1750

The simulation result of ARD, stretch, stress, and delay distribution show that the proposed TLHIT provides more reliable and efficient multicast in overlay networks.

5 Conclusions and Future Work

In this paper, path diversity using independent trees in overlay multicast is proposed to improve the performance of media streaming service. Two schemes to construct diverse paths for participating node are compared. One is THAG and the other is TLHIT. THAG makes all the multicast trees node-disjoint. In addition to the node-disjoint property, TLHIT builds multicast trees which are path-disjoint and load-balance to minimize the influence of failures. The reliability and efficiency of THAG and TLHIT are compared through several simulations. The average received descriptions (ARD) shows that each node has higher probability to receive more descriptions in TLHIT. The stretch and delay distribution show that each node experiences a shorter latency in TLHIT and the delay variance of each node is small. Moreover, the stress shows that the duty of each node is much more balanced in TLHIT. Hence, the simulation results indicate that TLHIT is a more reliable, efficient and load-balance scheme for multimedia streaming service.

In future, we intend to further improve the TLHIT scheme with the ability of detecting limits of participating nodes. A powerful node with higher bandwidth in the multicast group should undertake more data transmissions. Thus, we can avoid the bottleneck caused by weak nodes and this context-aware TLHIT provide optimal performance for video streaming.

Acknowledgement

The authors would like to thank the National Science Council in Taiwan R.O.C for supporting this research, which is part of the three projects numbered NSC 95-2221-E-168-029, NSC 94-2213-E-006-081 and NSC 95-2219-E-006-007.

References

[1] Tian, R., Zhang, Q., Xiang, Z., Xiong, Y., Li, X., Zhu, W.: Robust and Efficient Path Diversity in Application-Layer Multicast for Video Streaming. IEEE Transactions on Circuits and Systems for Video Technology 15(8), 961–972 (2005)

[2] Padmanabhan, V.N., Wang, H.J., Chou, P.A., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking. In: Proc. ACM NOSSDAV, Miami Beach, FL, pp. 177–186 (May 2002)

[3] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-bandwidth content distribution in a cooperative environment. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, Springer, Heidelberg (2003)

[4] Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable application-layer multicast. In: Proc. ACM SIGCOMM, pp. 205–217 (August 2002)

[5] Chu, Y., Rao, S., Seshan, S., Zhang, H.: Enabling conferencing applications on the internet using an overlay multicast architecture. In: Proc. ACM SIGCOMM, pp. 55–67 (August 2001)

[6] Liang, Y.J., Steinbach, E.G., Girod, B.: Real-time voice communication over the Internet using packet path diversity. In: Proc. ACM Multimedia 2001, (September/October 2001) pp. 431–440 (2001)
[7] Aopstolopoulos, J.: Reliable Video Communication over Lossy Packet Networks using Multiple State Encoding and Path Diversity. Visual Communications and Image Processing, 392–409 (January 2001)
[8] Andersen, D., Balakrishnan, H., Kaashoek, F., Morris, R.: Resilient Overlay Networks. In: Proc. 18th ACM Symposium on Operating Systems Principles, Banff Canada, pp. 131–145 (October 2001)
[9] Chu, Y., Rao, S., Seshan, S., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS, pp. 1–12 (June 2000)
[10] Goyal, V.K.: Multiple description coding: Compression meets the network. Signal Processing Magazine 18(5), 74–93 (2001)
[11] Bolot, J.: End-to-End Packet Delay and Loss Behavior in the Internet. In: Proceedings of ACM SIGCOMM, pp. 289–298 (September 1993)

A Cross Layer Time Slot Reservation Protocol for Wireless Networks

Bih-Hwang Lee, Chi-Ming Wong, and Hung-Chi Chien

Abstract. The function of medium access control (MAC) protocol and the capacity of physical layer sufficiently affect the system performance in the wireless networks. In this research, we propose a MAC protocol based on direct sequence code division multiple access (DS-CDMA) for the wireless networks to guarantee quality-of-service (QoS) which depend on physical layer information, named the cross layer time slot reservation (CLTSR) protocol. A channel is divided into control and data channels to transmit control and data packets in the corresponding control and data frames, respectively. The data frame is further subdivided into several time slots; each time slot can transmit different traffic types such as constant bit rate (CBR), variable bit rate (VBR), and available bit rate (ABR). Each station maintains the available spreading code (ASC) table to understand which time slots and spreading codes have been reserved. CLTSR will allocate time slots and the spreading code for the data frame by using the fixed time slot allocation (FTSA) or the mixed time slot allocation (MTSA). The QoS can be guaranteed by providing the reservation of time slots and the spreading code. A Markov model is applied to analyze the CLTSR DS-CDMA system; the analytical and simulation results show that the proposed CLTSR performs has been improved.

Keywords: medium access control (MAC), direct sequence code division multiple access (DS-CDMA), cross layer, quality-of-service (QoS).

1 Introduction

Wireless local area network (WLAN) has been widely used recently, which uses IEEE 802.11x protocol to coordinate all stations. Each station contends with each other by using the carrier sense multiple access with collision avoidance (CSMA/CA) to access the channel [1],[2]. CSMA/CA uses backoff algorithm to avoid collision and reduce the collision probability in the wireless environment. Several backoff algorithms have been proposed to improve system performance [3],[4], however, the performance is still limited by the physical capacity due to the use of single shared channel. Therefore, it is necessary to improve the cross layer functions between the physical layer and MAC layer.

For many years, some researches regarding the random access networking based on direct sequence code division multiple access (DS-CDMA) are studied [5]-[7], because multiple access interference (MAI) may be possibly suppressed in MAC layer. Generally in the controlled access CDMA (CA-CDMA) system, there exist control and data channels separately [8]. In the control channel, each station uses the signals of request-to-send (RTS) and clear-to-send (CTS) to contend the channel; and

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 400–409, 2007.

it also detects the degree of MAI to suitably adjust the transmitting power. However, this method needs more computational times and can not guarantee quality of service (QoS) in wireless network. Some MAC layer designs, such as the modified multi-carrier CDMA based on IEEE 802.11a [9], CDMA based ad hoc network [10] and adaptive acquisition collision avoidance multiple access (AACAMA) [11], just give the way to get spreading codes and cannot limit the number of the used spreading codes to suppress MAI. They also can not guarantee QoS in wireless network. In the distributed channel allocation protocol (DCAP) [12], each frame is divided into several time slots; each node contends to get a pair of time slot-spreading code (TC) to transmit data. Similarly, it cannot limit the number of the used spreading codes and the allocation of time-slots, hence system performance will be degraded especially while the allocating algorithm doesn't provide the different maximum numbers of spreading codes for the different traffic types such as constant bit rate (CBR), variable bit rate (VBR) and available bit rate (ABR). Furthermore, both RTS and CTS in DCAP contain all TC pairs such that their packet sizes become longer, hence system performance will be degraded in heavy traffic.

In this paper, we use the DS-CDMA technology in physical layer and propose a cross layer time slot reservation (CLTSR) protocol in MAC layer for wireless system. CLTSR also divides the channel into control channel and data channel; each station uses the modified RTS (MRTS) and modified CTS (MCTS), data slot request (DSR) and short ACK (SACK) to reserve time slots and spreading code in the data frame by four way hand shaking (MRTS-MCTS-DSR-SACK). CLTSR supports the maximum numbers of the spreading codes for different traffic types to suppress MAI. All reserved time slots and spreading codes are put in DSR such that both MRTS and MCTS can keep shorter sizes. We provide two kinds of time slot allocation algorithm in MAC layer: fixed time slot allocation (FTSA) and mixed time slot allocation (MTSA) to allocate time slots and spreading codes depending on the required bandwidth for each traffic type, therefore the guaranteed QoS and the improved system performance can be achieved.

Next section will describe the proposed CLTSR based on DS-CDMA technology in wireless ad hoc network. Section III will do the theoretical analysis using the finite state Markov chain, and section IV will simulate the system operation and then show that the result. Finally, the conclusion is drawn in section V.

2 Description of the Proposed CLTSR

To design CLTSR, two frame structures for control channel and data channel are used and shown in Fig. 1. Data can be transmitted in the k^{th} data frame according to the competitive result in the $(k-1)^{th}$ control frame. A data frame is subdivided into n time slots as shown in Fig. 2, which can support different traffic types such as CBR, VBR and ABR. In the DS-CDMA system, the maximum number of channels can be simultaneously used to transmit data for CBR, VBR and ABR depending on the different tolerable bit error rate (BER, P_e) [13], which is assumed that BPSK modulation scheme is used in the additive white Gaussian noise (AWGN) channel as shown in Eq. (1) [14],

$$P_e = Q\left(\sqrt{3N/K-1}\right) \tag{1}$$

where N is the spreading factor, and K is the maximum number of channels used to transmit simultaneously, and Q(x) is the complementary error function. We assume the maximum sets of spreading codes to transmit data for CBR, VBR and ABR are 41, 17, 11 corresponding to the bit error rate of 10^{-3}, 10^{-6}, 10^{-9}, respectively, if the spreading factor is 128.

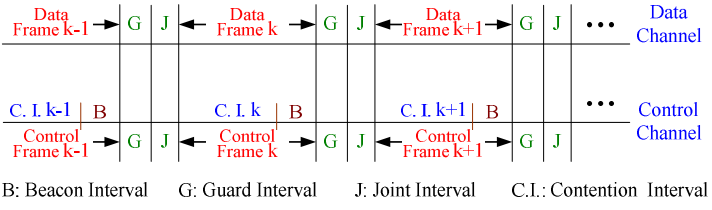


Fig. 1. The structure of control frame and data frame

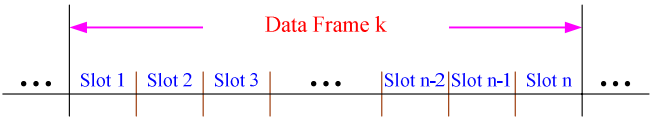
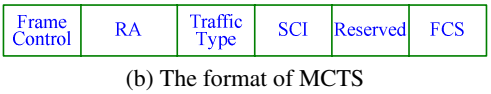


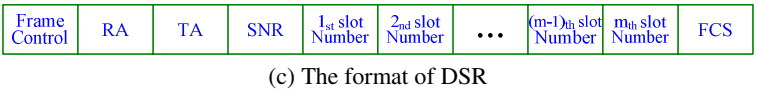
Fig. 2. The format of data frame

A control frame is divided into contention interval and beacon interval. The beacon interval is used to maintain system synchronization, while the contention interval is used to get the permission to send data in the next data frame. The contention is done by CSMA/CA with four-way hand shaking (MRTS-MCTS-DSR-SACK) procedure, where the formats of the corresponding control packets are shown in Fig. 3.

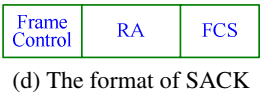
(a) The format of MRTS



(b) The format of MCTS



(c) The format of DSR



(d) The format of SACK

Fig. 3. The format of control packets

The field of traffic type may consist of two bits to represent three traffic types. The spreading code index (SCI) is indicated in the field of "SCI", which may has six bits to indicate at most 41 sets of spreading codes. The fields of "RA" and "TA" represent the MAC addresses for receiver and transmitter respectively. The field of the slot number reservation (SNR) is used to reserve the number of time slots to be transmitted in the data frame, whose content depends on the transmission bandwidth requirement. Let N_s be the number of time slots to be reserved in the data frame and to be obtained by Eq. (2).

$$N_s = \left\lceil \frac{r_b}{r_f * b_s} \right\rceil \tag{2}$$

where r_b is the required transmission bandwidth in bit per second (bps); r_f is the number of data frames per second; and b_s is the transmitted bits in each time slot. The field of "Slot Number" represents the reserved time slots in the data frame, and the amount of "Slot Number" is the same as SNR. Thus, the size of DSR packet depends on how many time slots are needed. The functions of "Frame Control" and "FCS" are the same as the IEEE 802.11 standard [15].

Next, we introduce the procedure of CLTSR to access channel through the control packets.

Step 1: Use allocation algorithm to reserve time slots and spreading codes for transmission in the next data frame.
Step 2: Use the four-way handshaking procedure to contend and get the permission of time slots and spreading codes.
Step 3: Monitor the DSR packets from other stations to check the confliction. It must run time slot allocation algorithm again when the content of its DSR packet conflicts with other stations, and go to step 2.
Step 4: Use those time slots and spreading codes to transmit frames until the transmission finished, then releases the time slots and spreading codes.

Table 1. Initialization of ASC Table

		Slot 1	Slot 2	…	Slot n-1	Slot n
Traffic types		0	0	…	0	0
SCI	1	TCI(1,1)	TCI(1,2)	…	TCI(1,n-1)	TCI(1,n)
	2	TCI(2,1)	TCI(2,2)	…	TCI(2,n-1)	TCI(2,n)
	:	:	:	…	:	:
	11	TCI(11,1)	TCI(11,2)	…	TCI(11,n-1)	TCI(11,n)
	:	:	:	…	:	:
	17	TCI(17,1)	TCI(17,2)	…	TCI(17,n-1)	TCI(17,n)
	:	:	:	…	:	:
	41	TCI(41,1)	TCI(41,2)	…	TCI(41,n-1)	TCI(41,n)

Each station in the system must cooperate each other well by some distributed resource allocation algorithms because there has no base station. FTSA and MTSA allocate time slots and spreading codes according to the available spreading code (ASC) table in each station, where the initialization of ASC table is shown in table 1. A time slot spreading code index (TCI) pair is a pair of time slot and spreading code in the ASC table, while the marked TCI pair in ASC table means that this TCI pair has been reserved already. The traffic type in each time slot is decided by the first station of using it, which is set to 0, 1, or 2 for CBR, VBR, and ABR, respectively. The FTSA algorithm is described as follows.

Step 1: Search from ASC table to find all time slots having the same traffic type and available TCI pairs relative to those time slots.
Step 2: Allocate number of TCI pairs which obtained by Eq. (2) from those available TCI pairs with a spreading code in best fit.
Step 3: Try to find enough free time slots and set them to this traffic type if the TCI pairs is not enough.
Step 4: Go to step 1 if there are still no enough free time slots, then the station enters blocking state.

The MTSA algorithm is obtained by modifying the FTSA algorithm and described as follows.

Step 1: The same as step 1 in FTSA.
Step 2: The same as step 2 in FTSA.
Step 3: The same as step 3 in FTSA.
Step 4: Try to find TCI pairs of VBR traffic to be used for CBR traffic.
Step 5: The same as step 4 in FTSA.

3 Theoretical Analysis

This section analyzes the proposed CLTSR with FTSA because MTSA is similar to FTSA. We assume that the arrival traffics are the Poisson process with the mean arrival rates λ_C, λ_V, and λ_A, for the traffic types CBR, VBR and ABR, respectively. The service processes are also assumed to be exponentially distributed with the mean service rates μ_C, μ_V, and μ_A, for the traffic types CBR, VBR and ABR, respectively. Throughput can be obtained depending on how many TCI pairs are occupied by each traffic type. The TCI pair occupied by any traffic can be modeled by Markov chain. We assume that there are n time slots in each data frame, and the maximum number of TCI pairs can be occupied by CBR, VBR, and ABR are n_C, n_V, and n_A, respectively, but Eq.(3) must be satisfied.

$$\left\lceil \frac{n_C}{41} \right\rceil + \left\lceil \frac{n_V}{17} \right\rceil + \left\lceil \frac{n_A}{11} \right\rceil \le n \tag{3}$$

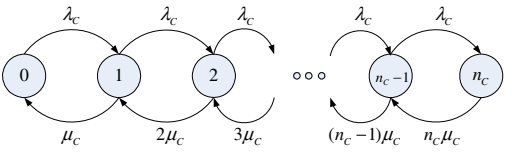


Fig. 5. State transition diagram of the CBR traffic

The number of TCI pair occupied by CBR traffic can be modeled by Markov chain with n_C states and its state transition diagram is shown as Fig. 5. We assume that only one TCI pair can be occupied whenever the CBR type traffic arrives. The state transition of the Markov chain can be characterized by a generator matrix. Let us denote $\mathbf{Q}_C$ be the generator matrix for CBR traffic shown as Eq. (4), while $\mathbf{P}_C$ represents the steady state probability vector for CBR traffic, where $\mathbf{P}_C = [p_{C0} \quad p_{C1} \quad p_{C2} \quad \cdots \quad p_{Cn_C}]$. The element of $\mathbf{P}_C$ represents the steady state probability of each state in Fig. 5, which can be obtained by solving the balance equation shown as Eq. (5).

$$\mathbf{Q}_C = \begin{bmatrix} -\lambda_C & \lambda_C & 0 & 0 & & 0 & 0 & 0 \\ \mu_C & -\lambda_C-\mu_C & \lambda_C & 0 & & 0 & 0 & 0 \\ 0 & 2\mu_C & -\lambda_C-2\mu_C & \lambda_C & & 0 & 0 & 0 \\ 0 & 0 & 3\mu_C & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & & 0 & \lambda_C & 0 \\ 0 & 0 & 0 & 0 & & (n_C-1)\mu_C & -\lambda_C-(n_C-1)\mu_C & \lambda_C \\ 0 & 0 & 0 & 0 & & 0 & n_C\mu_C & -n_C\mu_C \end{bmatrix} \tag{4}$$

Then, we have $\mathbf{P}_C \, \mathbf{Q}_C = \mathbf{0}$ (5)

Similarly, the generator matrices for VBR and ABR traffics can also be obtained.

The TCI pairs occupied by VBR or ABR traffics are random variables, because the required bandwidths for VBR and ABR traffics are also random variables; therefore, we should transform the distribution of bandwidth requirement to the distribution of TCI pairs' requirement for VBR and ABR traffics. Let us denote $f_{VR}(r)$ be the probability density function of the bandwidth requirement for VBR traffic with boundary between R_{min} and R_{max} (bps). Let m_l and m_u represent the minimum and maximum number of TCI pairs occupied by VBR traffic and have the relationship with $m_u > m_l > 1$, where $m_l = \frac{R_{\min}}{B_s}$ and $m_u = \frac{R_{\max}}{B_s}$ if B_s is the transmission rate of a TCI pair in data frames per second. The probability distribution of the number of TCI pairs' requirement for VBR traffic can be represented by a vector $\mathbf{b}_V = [b_{V0} \quad b_{V1} \quad b_{V2} \quad \cdots \quad b_{Vm}]$, where $m = m_u - m_l$. The element b_{Vk} ($k = 0, 1, \ldots, m$) in vector $\mathbf{b}_V$ represents the probability that there are ($m_l + k$) TCI pairs required by VBR traffic and can be calculated by Eq. (5).

$$b_{Vk} = \begin{cases} \int_0^{R_{\min}} f_{VR}(r)dr, & \text{for } k = 0 \\ \int_{(m_l+k-1)B_s}^{(m_l+k)B_s} f_{VR}(r)dr, & \text{for } k = 1, 2, ..., m-1 \\ \int_{(m_u-1)B_s}^{\infty} f_{VR}(r)dr, & \text{for } k = m \end{cases} \tag{6}$$

Note that $\sum_{k=0}^{m} b_{Vk} = 1$. Actually, CBR traffic can be regarded as a special case of VBR traffic, as $m_l = m_u = 1$ and $m = 0$. Therefore, the arrival and departure rates for VBR traffic from a state can be obtained by the vectors $\lambda_V \mathbf{b}_V$ and $\mu_V \mathbf{b}_V$, respectively.

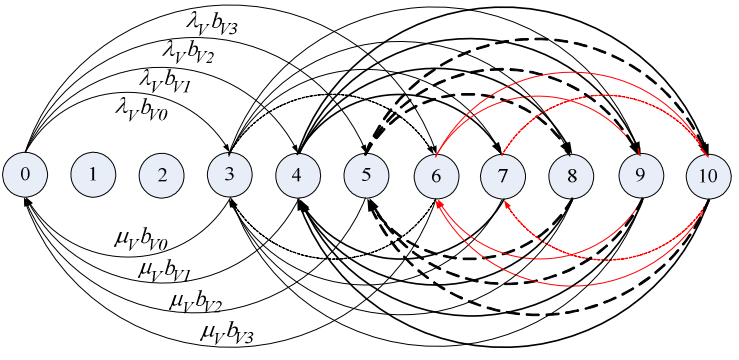


Fig. 6. Example state transition diagram of VBR traffic

For example, if m_l and m_u equal 3 and 6, respectively, then m equals 3. If n_V equals 10, the state transition diagram is shown as Fig. 6 and the reduced generator matrix $\mathbf{Q}_V$ has the dimension of 9×9 shown as Eq. (7). In Fig. 6, it is obvious that the states 1 and 2 will never be reached so that their probabilities are zero. The transition probabilities $p_{01}, p_{02}, ..., p_{0(m_l-1)}$ and $p_{10}, p_{20}, ..., p_{(m_l-1)0}$ will be zero, because the minimum requirement of TCI pairs is $m_l > 1$. Some states may have more than one possible transitions for $m \geq m_l$. For example, the 6th state may be transited from the initial state with probability $\lambda_V b_{v3}$ or from the 3rd state with probability $\lambda_V b_{v0}$, while the transition rate from the 9th state to the 6th state can be obtained to be $3\mu_V b_{v0}^2 + 2\mu_V b_{v0} b_{v3}$.

$$\mathbf{Q}_V = \begin{bmatrix} q_{v00} & \lambda_V b_{V0} & \lambda_V b_{V1} & \lambda_V b_{V2} & \lambda_V b_{V3} & 0 & 0 & 0 & 0 \\ \mu_V b_{V0} & q_{v11} & 0 & 0 & \lambda_V b_{V0} & \lambda_V b_{V1} & \lambda_V b_{V2} & \lambda_V b_{V3} & 0 \\ \mu_V b_{V1} & 0 & q_{v22} & 0 & 0 & \lambda_V b_{V0} & \lambda_V b_{V1} & \lambda_V b_{V2} & \lambda_V b_{V3} \\ \mu_V b_{V2} & 0 & 0 & q_{v33} & 0 & 0 & \lambda_V b_{V0} & \lambda_V b_{V1} & \lambda_V b_{V2} \\ \mu_V b_{V3} & 2\mu_V b_{V0} & 0 & 0 & q_{v44} & 0 & 0 & \lambda_V b_{V0} & \lambda_V b_{V1} \\ 0 & 2\mu_V b_{V1} & 2\mu_V b_{V0} & 0 & 0 & q_{v55} & 0 & 0 & \lambda_V b_{V0} \\ 0 & 2\mu_V b_{V2} & 2\mu_V b_{V1} & 2\mu_V b_{V0} & 0 & 0 & q_{v66} & 0 & 0 \\ 0 & 2\mu_V b_{V3} & 2\mu_V b_{V2} & 2\mu_V b_{V1} & 3\mu_V b_{V0}^2 + 2\mu_V b_{V0} b_{V3} & 0 & 0 & q_{v77} & 0 \\ 0 & 0 & 2\mu_V b_{V3} & 2\mu_V b_{V2} & 3\mu_V b_{V1} b_{V0} + 2\mu_V b_{V1} b_{V3} & 3\mu_V b_{V0} & 0 & 0 & q_{v88} \end{bmatrix} \tag{7}$$

Similarly, the reduced generator matrix for ABR traffic $\mathbf{Q_A}$ can also be obtained by substituting λ_A and μ_A for λ_V and μ_V, respectively. The steady state probability vector for ABR traffic $\mathbf{p}_A$ can be obtained by solving the balance equation shown as Eq. (8).

$$\mathbf{p}_A \mathbf{Q}_A = \mathbf{0} \tag{8}$$

4 Simulation Results

In this section, we simulate the proposed CLTSR system by using Visual C++ program and compare with the theoretical analysis. Let us assume that data channel has the bandwidth of 10.6 Mbps; each time slot has the length 2 kbits (about 0.1887ms); each data frame has 164 time slots (about 30.94ms); and there yields about 32 data frames per second. Let us also assume that the arrival traffics of CBR, VBR and ABR occupy 19%, 4% and 77%, respectively. The average bandwidth requirement for CBR, VBR and ABR traffics are 64, 512, and 256 kbps, respectively. Similarly, the mean access duration for CBR, VBR and ABR traffics are 5, 10 and 1.56 minutes, respectively. Throughput is calculated by $(N_{pkt} \times l)/T_{sim}$, where l, N_{pkt} and T_{sim} are the packet length, the total number of the successful transmitted packets and the simulation time. The blocking probability is obtained by $N_b/(N_b + N_s)$ where N_b and N_s are the number of blocking and success packets, respectively.

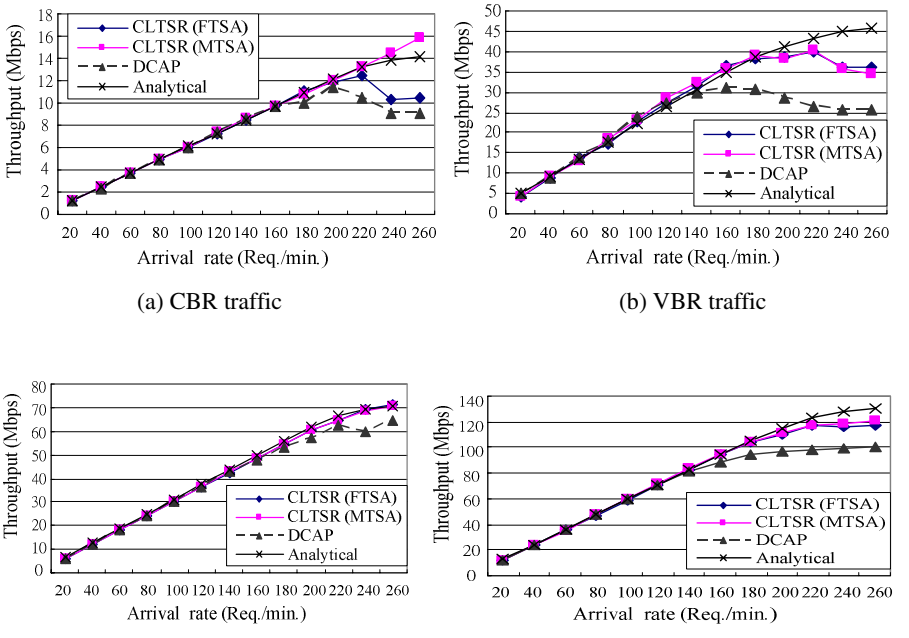


Fig. 7. Throughput versus arrival rate for CLTSR, DCAP and analytical solutions

Fig. 7 shows the simulated throughput for FTSA/MTSA in CLTSR, DCAP and theoretical analysis obtained from the Markov model above. We see that the proposed Markov model fits the simulation results very well except arrival traffic higher than 220 (Req./min.). Although the CBR traffic occupy 19% of arrival traffic, the allowable 41 sets of spreading codes in a time slot and 64Kbps constant bandwidth requirement will make it occupying time slot in the data frame slower than ABR traffic in almost no available time slots can be used. And it also means that when the arrival traffic is heavy (over 220 (Req./min.)), the throughput of CBR traffic will lower than theoretical analysis. But once the MTSA is used, we can see the throughput of CBR traffic even higher than theoretical analysis, as shown in Fig. 7 (b). One way to make throughput always near the theoretical analysis is expanded time slot or channel capacity in the data channel. Of course, due to the DCAP hasn't good time slot allocation algorithm to allocate time slot, it will have lower performance than CLTSR system. Similarly, the VBR traffic has same situation as the CBR traffic, as shown in Fig. 7 (c). And we can see the throughput of VBR using MTSA is same as using FTSA. So, the MTSA never affect the throughput in VBR traffic but enhance the throughput in CBR traffic. In the Fig. 7 (d), the FTSA, MTSA and theoretical analysis are almost same everywhere in the ABR traffic. Fewer sets of spreading codes, smaller access duration and higher occupation of arrival traffic make ABR traffic has more stable situation.

5 Conclusions

We have proposed the CLTSR for DS-CDMA based wireless network. The special reservation protocol which combines the function of DS-CDMA in physical layer and time slots/spreading code allocation in MAC layer not only improve system performance but also provide the guarantee of QoS. It is shown that the Markov chain models are accurate in predicting the behavior of CLTSR under the assumption that the arrival traffic has Poisson distribution. The MTSA is suggested to use because the performance of VBR and ABR traffic are almost same in both FTSA and MTSA, while the MTSA has better performance in CBR traffic. This is a special MAC protocol design dependent on the capacity of physical layer. Due to the advance multiple access and channel estimation technology in wireless communication networking, we need a better MAC protocol to use them. Then, the performance with respect to the entire system can have the best situation. This is the motive that we begin to design the CLTSR.

References

1. Bianchi, Y.G.: Performance Analysis of the IEEE 802.11 Distributed Coordination Function. IEEE Journal on Selected Areas in Communications 18(3), 535–547 (2000)
2. Robinson, J.W., Randhawa, T.S.: Saturation throughput analysis of IEEE 802.11e enhanced distributed coordination function. IEEE Journal on Selected Areas in Communications 22(5), 917–928 (2004)
3. Kwon, Y., Fang, Y., Latchman, H.: Design of MAC Protocol With Fast Collsion Resolution for Wireless Local Area Networks. IEEE Trans. Wireless Commun. 3(3), 793–807 (2004)

4. Haas, Z.J., Deng, J.: On Optimizing the Backoff Interval for Random Access Schemes. IEEE Trans. Commun. 51(12), 2081–2090 (2003)
5. Raychaudhuri, D.: Performance analysis of random access packet-switched code division multiple access systems. IEEE Trans. Commun. COM-29(6), 895–901 (1981)
6. Morrow, R., Lehnert, J.: Packet throughput in slotted ALOHA DS/SSMA radio systems with random signature sequences. IEEE Trans. Commun. 40(7), 1223–1230 (1992)
7. Sato, T., Okada, H., Yamazato, T.: Throughput Analysis of DS/SSMA Unslotted ALOHA System with Fixed Packed Length. IEEE Journal on Selected Areas in Communications 14(4), 750–756 (1996)
8. Muqattash, A., Krunz, M.: CDMA-Based MAC Protocol for Wireless Ad Hoc Networks. In: Proceedings of the 4ACM international symposium on Mobile ad hoc networking & computing (July 2003)
9. Orfanos, G., Habetha, J., Liu, L.: MC-CDMA Based IEEE 802.11 Wireless LAN. In: IEEE Computer Society's 12th Annual International Symposium, pp. 400–405 (October 2004)
10. Fantacci, R., Ferri, A., Tarchi, D.: A MAC technique for CDMA based Ad-Hoc networks. In: IEEE Wireless Communications and Networking Conference, vol. 1, pp. 645–650 (March 2005)
11. Qiang, G., Liu, Z., Ishihara, S., Mizuno, T.: CDMA-based Carrier Sense Multiple Access Protocol for Wireless LAN. In: IEEE Vehicular Technology Conference 53rd, vol. 2 (May 2001)
12. Yang, H., Kim, K.: Multimedia Ad Hoc Wireless LANs with Distributed Channel Allocation Based on OFDM-CDMA. IEICE Trans. Commun. E86-B(7) (2003)
13. Akyildiz, I.F., Levine, D.A., Joe, I.: A slotted CDMA protocol with BER scheduling for wireless multimedia networks. IEEE/ACM Trans. Networking 7(2), 146–158 (1999)
14. Rappaport, T.S.: Wireless Communications PRINCIPLES AND PRACTICE, 2nd edn. pp. 621–650. Prentice Hall, Englewood Cliffs (2002)
15. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, ANSI/IEEE std. 802.11

An Efficient Handoff Strategy for Mobile Computing Checkpoint System

Chaoguang Men[1,2], Zhenpeng Xu[2], and Dongsheng Wang[1,2]

[1] National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China
[2] Research Center of High Dependability Computing Technology, Harbin Engineering University, Harbin, Heilongjiang, 150001, P.R. China
{mencg,wds}@tsinghua.edu.cn,
{menchaoguang,xuzhenpeng}@hrbeu.edu.cn

Abstract. The *Eager*, *Lazy* and *Movement-based* strategies are used in mobile computing system when handoff. They result in performance loss while moving the whole checkpoint on fault-free or slow recovery while not moving any checkpoint until recovery. In the paper, a compromise strategy is proposed. The whole recovery information are broken into two parts, which one little part with high-priority should be transferred to the new cell during handoff and another large part with low-priority should be transferred only when the mobile host recovers from a fault. From the view of mobile host, it seems that all recovery information reside on the local mobile support station. The strategy guarantees little performance losing when fault-free and quick recovery when fault occurs. Experiments and analysis show the handoff strategy performance overcomes others.

Keywords: mobile computing, fault tolerant, checkpoint, handoff, rollback recovery.

1 Introduction

Checkpointing and rollback-recovery has been an attractive technique for providing fault-tolerance in mobile computing system [1]. Due to the mobility of the hosts, limited bandwidth, highly unreliable wireless link, mobile hosts disconnect from network voluntarily, power restriction and limitation of storage space in mobile devices, conventional checkpointing recovery schemes used in wired distributed network cannot be directly applied to mobile environment [2]. When a mobile host (*MH*) moves from one cell to another, *Eager*, *Lazy* and *Movement-based* strategies are used, which move the whole recovery information to new mobile support station (*MSS*) or not move any recovery information until a *MH* recovers [3]. A strategy which moves the whole recovery information in fault-free will depress the performance of system due to transfer useless information and others which not move any recovery information until a fault occurs will delay the system recovers from a fault due to recovery information can not be gotten in time. No one strategy is

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 410–421, 2007.

excellent in every circumstance. A compromise strategy is proposed. Only a few part of recovery information is moved to the new local cell when a *MH* moves, which little useless work is done when fault-free or quick recovery can be done when a fault occurs.

The paper is organized as follows: Section 2 introduces the system model and definitions. Section 3 presents a checkpoint and recovery strategy with an efficient handoff scheme for mobile computing. Section 4 gives its correctness proofs. Section 5 compares the handoff scheme with others. Section 6 draws a conclusion.

2 Preliminaries

A mobile computing system $MCS=\langle N, C\rangle$ is composed of a set of nodes N and a set of channels C. The set of nodes $N=M\cup S$ can be divided into two types, $M=\{MH_1, MH_2,..., MH_n\}$ is the set of *MH*s, which are able to move while retaining their network connections and $S=\{MSS_1, MSS_2,..., MSS_m\}$ is the set of static nodes acting as the *MSS*s. The set of channels $C=W\cup W'$ can be divided into two disjoint sets, the set of high-speed wired channels W, where $W=S\times S$ is the type through that static nodes are connected, and the set of low bandwidth wireless channels W', where $W'=S\times M$ is the type through that *MH*s are connected to a *MSS*. A cell is a geographical area covered by a *MSS*. A *MH* residing in the cell of MSS_p can directly communicate with MSS_p through a wireless channel. In a cell of MSS_i, let $CL_i=\{MH_j \mid MH_j\in MSS_i, 0<j<n+1\}$ denotes the active nodes or sleeping nodes identified by $Active\_MH\_List_i$ or $Disconnected\_MH\_List_i$ respectively, then there exists a channel $\langle MSS_i, MH_j\rangle \in W'$ only if $MH_j\in CL_i \Rightarrow MH_j\notin CL_k$, $\forall k\neq i$, assuming that the geographical cells around each of the *MSS* do not overlap. The *MH*s have limited battery power and hence cannot keep communication with the *MSS*s for long, hence they often disconnect from the network. Such disconnections can be voluntary without any fault or involuntary due to abruptly running out of battery. The mobile computing system model is described in Fig. 1.

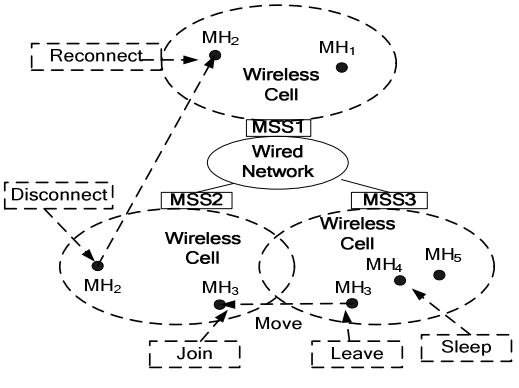


Fig. 1. Mobile computing system model

Distributed computations running concurrently on different *MH*s consist of a set of N processes denoted by P_1, P_2, ..., P_n. Processes do not share a global memory or a global physical clock, and they communicate with each other only through message passing. For simplicity, we assume that only one process runs on each *MH*. So we can use the terms '*MH*' and process interchangeably. We assume that each of the channels is bidirectional with reliable *FIFO* delivery of messages and the message transfer delays are finite but arbitrary. Processes follow the piece-wise deterministic execution model, and the underlying computation is asynchronous. The fault model is assumed to be *fail-stop* and all faults can be detected immediately, which results in halting failed process, initiating recovery action are considered to be transient and the same fault would not repeat when the process restarts.

Let $Rcv(i,\alpha)$ denotes the αth message receiving event of a process P_i; the state interval $I(i,\alpha)$ denotes the sequence of states generated between $Rcv(i,\alpha\text{-}1)$ and $Rcv(i,\alpha)$, where $\alpha>0$ and $Rcv(i,0)$ denotes the initial event. Then, the dependency relation of processes caused by the message communication can be defined as follows:

Definition 1. *Dependency Relation*: A state interval $I(i,\alpha)$ is said to be dependent on another state interval $I(j,\beta)$ if one of the following conditions is satisfied and the dependency relation is denoted by $I(j,\beta)\rightarrow I(i,\alpha)$:

(i). $i=j$ and $\alpha=\beta+1$
(ii). For an event $Rcv(i,\alpha)$, the corresponding message-sending event happens in $I(j,\beta)$
(iii). For any $I(k,\gamma)$, $I(j,\beta)\rightarrow I(k,\gamma)$ and $I(k,\gamma)\rightarrow I(i,\alpha)$ [2].

With the pessimistic message logging scheme, an interval $I(i,\alpha+1)$ can be fully recovered after a fault if the event, $Rcv(i,\alpha)$, has been stably logged; Otherwise, the interval becomes lost. During the rollback-recovery of a process, the dependency relation may cause an inconsistency problem.

Definition 2. *Orphan interval*: An interval on which depends any lost interval is called an orphan state interval.

Definition 3. *Consistent Recovery*: The recovery from a fault $F(i,f)$ is said to be consistent, if and only if there is not any orphan state interval, that is, for any $I(i,\alpha)\in L(i,f)$ there exists no $I(j,\beta)$, such that $I(i,\alpha)\rightarrow I(j,\beta)$. Where $F(i,f)$ denotes the fth fault of P_i and $L(i,f)$ denotes the set of lost state intervals caused by $F(i,f)$ [3].

The handoff and location scheme are supplied to support the mobility of *MH*. When a *MH* leaves a cell and enters another cell, it must end its current connection by sending a leave(r) message to its local *MSS*, where r is the sequence number of the last message received from the *MSS*. Then the *MH* establishes a new connection by sending a join(*MH*-id, previous *MSS*-id) message to the new *MSS*. Usually, leaving a cell and entering another cell happens simultaneously when an *MH* crosses the boundary between two cells and it is called a *handoff*. Each *MSS* maintains a list of identifiers of *MH*s that are currently supported by the *MSS*. A *MH* can also disconnect itself from the local *MSS* without leaving the cell by sending disconnect(r) message when the *MH* goes into the sleep mode for power conservation. Later, the *MH* can reconnect to any *MSS* by sending a reconnect(*MH*-id, previous *MSS*-id) message to

the *MSS*. If the *MH* is reconnected to a new *MSS*, the new *MSS* informs the previous *MSS* of the reconnection of the *MH* so that the previous *MSS* can perform the proper handoff procedures [4].

Handoff time is an important parameter which affects mobile system performance besides checkpoint state-saving cost and recovery cost [5]. There three categories handoff strategy named *Eager*, *Lazy* and *Movement-based*. *Eager* mobility handoff strategy, which also named *Pessimistic*, always keeps the logging and checkpoint information in the local *MSS* in which the *MH* currently resides [5]. Thus, when the *MH* moves from one *MSS* to another during the execution of a mobile application, all the checkpoint and logging information must be moved to the current *MSS* as well. The advantage of this approach is fast failure recovery. But the *MSS*s visited by the *MH* have to experience high fault-free cost to transfer the recovery information and access the stable storage. Under the *Lazy* strategy, on the other hand, the checkpoint and logging information do not be moved as the *MH* moves [5]. Rather, a forwarding pointer is established from the local *MSS* to the last *MSS* so that when a failure occurs, the checkpoint and logging information of the mobile application can be recovered from all the *MSS*s on the forwarding chain by following the links. The advantage of this approach is little fault-free cost, but the recovery cost can be too high, if the recovery information is dispersed over a wide range of cells. The tradeoff schemes are *Movement-based* handoff strategies, which are *Distance-based* and *Frequency-based* [4]. Under the *Distance-based* scheme, which focuses on the distance between MH_i and the *MSS* carrying latest checkpoint of MH_i, the checkpoint and message logs need to be moved into a *MSS* near MH_i, only when the moving distance of MH_i from a *MSS* carrying the latest checkpoint exceeds a certain threshold. On the other hand, the *Frequency-based* scheme concerns the number of handoffs, since that number indicates the number of sites carrying the message logs and the frequency of communication for collecting the message logs in case of recovery. Hence, in this scheme, MH_i keeps counting the number of handoff and transfers the checkpoint and logs if the number exceeds a certain value. Of course, in both of the above schemes, the recovery cost and the fault-free operation cost is adjustable using the threshold values. Checkpoint and logs are moved to new local *MSS* when fault-free, the *Movement-based* schemes have the disadvantage of *Eager*. Checkpoint and logs are not moved to new local *MSS* until recovery, the *Movement-based* schemes have the disadvantage of *Lazy*. Obviously, how effective these strategies would be depends on various system parameters, including the checkpoint rate, logging message arrival rate, user mobility rate, failure rate, and bandwidth. No one scheme is always better than others under all situations [6].

3 The Recovery Scheme

The proposed recovery scheme is based on independent checkpointing, pessimistic message logging and asynchronous rollback-recovery. An efficient handoff scheme is proposed. Different from *Eager*, *Lazy* and *Movement-based* schemes which move the whole recovery information or do not move any recovery information until recovery, in our strategy, the whole recovery information which include checkpoint and logs are broken into two parts. One part includes only a little part of recovery information with

high-priority. The other part includes the rest of the recovery information with low-priority. The high-priority part of recovery information is treated as that in *Eager* scheme and the low-priority part of recovery information is treated as that in *Lazy* scheme. Appropriate partitions high-priority and low-priority recovery information can satisfy both quick recovery and fault-free cost. When recovering from a fault, the high-priority part can be transferred instantly to the recovering *MH*, the low-priority part can be collected by the local *MSS* simultaneously from other *MSS*s and then be transferred to the recovering *MH* successively. From the view of the recovering *MH*, it seems that all recovery information always resides on the local *MSS*.

3.1 The Data Structure and Denotations

Let $CK_{i,\alpha}$ denotes the αth checkpoint of MH_i; CK_info_i is a record which contains six variables, *CK_sn*, *CK_loc*, *CK_low*, *Logm_seq*, *Send_max*, and *Log_queue*. *CK_sn* denotes the sequence number of the latest checkpoint and the *CK_loc* denotes the identifier of the *MSS* carrying the high-priority of the latest checkpoint; *CK_low* denotes the identifier of *MSS* carrying the low-priority of the latest checkpoint; *Logm_seq* denotes the sequence number of the first message logged after the latest checkpoint; *Send_max* denotes the maximum sequence number of message sent successfully by MH_i to other *MH*s since the latest checkpoint; *Log_queue* is a list established for the local *MSS* to save the identifiers of *MSS*s which have the logs saved after the latest checkpoint; $Msg_{i,\alpha}$ denotes the αth message sent by MH_i; Rcv_seq_i is an integer variable, which denotes the maximum sequence number of messages that have been received and consumed in MH_i. $Logm_{i,\alpha}$ denotes the αth message log.

3.2 The Checkpointing and Logging

Each MH_i takes an initial checkpoint on initialization and sets the corresponding checkpoint sequence number $CK\_info_i.CK\_sn$ to 0. Every *MH* takes checkpoint periodically. When MH_i finishes a new checkpoint, the information about this checkpoint is recorded in CK_info_i. The CK_info_i and the new checkpoint will be sent to its local MSS_p.

Each *MSS* logs the received messages before delivering to *MH*s in its cell. As a message heading for MH_i should be routed through the local MSS_p, using the local MSS_p to log the message into its storage space will not incur extra overhead. MSS_p also logs the messages of the mobility of *MH*s, including the messages of *MH*s to join in, leave from, disconnect from and reconnect to the cell. Upon a user input of the *MH*, a copy of it is firstly forwarded to the local MSS_p for logging in case of its lost. On receipt of the acknowledgment from MSS_p, the *MH* starts to process the input event. When MH_i leaves or disconnects from MSS_q, it sends *Disconnect*(*i*) message to the local MSS_q for logging. MSS_q logs the event on the receipt of it and deals with it. When MH_i joins in a new cell of *MSS*, says MSS_r, it sends *Join*(MSS_q) to MSS_r. And MSS_r will add MSS_p into the $CK\_info_i.Log\_queue$ if MSS_p is not in the $CK\_info_i.Log\_queue$.

The checkpointing and message logging algorithm is described in Fig. 2.

Actions taken when checkpointing Timer of MH_i Expires:
CK (i, ++$CK\_info_i.CK\_sn$);
/*saves the state of MH_i as a new checkpoint of MH_i.*/
$CK\_info_i.Logm\_seq=Rcv\_seq_i$+1;
/*assigns the sequence number of the first message which will be logged after the new checkpoint.*/
Send ($CK_{i,\alpha}$, CK_info_i, MSS_p) ;
/* Sends the new checkpoint and its information to its local MSS. */
Actions taken when MSS_p receives $CK_{i,\alpha}$ and CK_info_i from MH_i:
Store ($CK_{i,\alpha}$, CK_info_i);
/* Saves the new checkpoint and its information in local MSS_p. */
$CK\_info_i.CK\_loc= MSS_p$;
/* identifies the MSS carrying the high-priority of latest checkpoint.*/
$CK\_ low = MSS_p$;
/* identifies the MSS carrying the low-priority of latest checkpoint.*/
Actions taken when MH_i receives a message $Msg_{k,\alpha}$ from local MSS_p:
Consume $Msg_{k,\alpha}$;
/* MH_i deals with the message.*/
Rcv_seq_i ++;
/* MH_i adds the sequence number of message.*/
Actions taken when MSS_p receives a message $Msg_{i,\alpha}$ from MH_i:
Log $Msg_{i,\alpha}$;
$Logm_{i,\alpha}$++;
/* saves the new message into log-space and adds the message number. */
If $MSS_p \notin CK\_info_i.Log\_queue$
Then $CK\_info_i.Log\_queue = CK\_info_i.Log\_queue \bigcup \{MSS_p\}$;
/*add MSS_p to the $CK\_info_i.Log\_queue$. */
Transfer $Msg_{i,\alpha}$;
/* forwards the computational message to goal MH_i.*/
If ($Msg_{i,\alpha} \in$ {join, leave, disconnect, reconnect})
Actions ;
/* takes related actions according to the received messages.*/

Fig. 2. Checkpointing and message logging algorithm

3.3 The Handoff Strategy

The recovery information including checkpoint and massage logs are broken into two parts that one with high-priority and the other with low-priority. The main idea is that low-priority checkpoint information can be sent to the local MSS of recovering MH through high speed wired network at the same time as the high-priority recovery information is being sent to recovering MH through low speed wireless network.

The amount of high-priority part and low-priority part of recovery information depend on the speeds of wired and wireless networks. Set LT_{max} and WT_{min} denote the maximum communication speed of wireless network and minimum communication speed of wired network respectively. Set VP_0 denotes the amount of high-priority

recovery information. For simplicity, we assume that the amount transmitted is the integral multiple of packet size.

When a MH_i recovers from a fault, the VP_0 will be sent to MH_i from the local MSS_p instantly. The transmission time at least is:

$$t_1 = \frac{VP_0}{LT_{\max}} . \tag{1}$$

The amount of low-priority recovery information collected from other *MSS*s simultaneously in time t_1 at least is:

$$VP_1 = t_1 * WT_{\min} = \frac{VP_0 \times WT_{\min}}{LT_{\max}} . \tag{2}$$

Because the speed of wired network is faster than the speed of wireless, so $VP_1 > VP_0$, that is more information can be collected through wired network when an amount of information is sent through wireless network. In turn, more recovery information, VP_2, can be collected through wired network when VP_1 is sent through the wireless network. The amount of recovery information transferred to MSS_p from other *MSS*s is:

$$VP_0 \times \left[\frac{WT_{\min}}{LT_{\max}} + \left(\frac{WT_{\min}}{LT_{\max}} \right)^2 + \cdots + \left(\frac{WT_{\min}}{LT_{\max}} \right)^{n-1} + \left(\frac{WT_{\min}}{LT_{\max}} \right)^n \right] . \tag{3}$$

The effective handoff scheme is described in Fig. 3.

Actions taken when MSS_p receives join (MSS_q) request:
Send (VP_0_*retrieve*, MSS_q)
/*sends request message to MSS_q get high-priority part of $CK_{i,\alpha}$ ***/**
Actions taken when MSS_q receives VP_0_*retrieve* from the new local MSS_p:
Send (VP_0 , MSS_p)
/*sends high-priority part of recovery information to MSS_p. */
Actions taken when MSS_p receives VP_0:
CK_loc=MSS_p;
Store *CK_info*$_i$ and high-priority part of $CK_{i,\alpha}$;
/* identifies the *MSS* carrying the high-priority of latest checkpoint and saves the high-priority of recovery information.*/

Fig. 3. An effective handoff scheme

3.4 Independent Recovery

When recovering from a fault, MH_i sends a recovery request, *RollbackReq*(*i*), to its local MSS_p. If *CK_info*$_i$ is saved in the local MSS_p, the high-priority part of recovery information is sent to MH_i instantly. To collect the rest of checkpoint and logs, the local MSS_p sends *Chkpt_retrieve*(*CK_info*$_i$.*CK_sn*) and *Log_retrieve*(*CK_info*$_i$.*Logm_seq*) request to other *MSS*s according to *CK_low* and *CK_info*$_i$.*Log_queue*. After receiving the request, the other *MSS*s reply with the low-priority checkpoint part and

the logs whose sequence number is not less than $CK\_info_i.Logm\_seq$. After transferring the high-priority part of recovery information to the recovering MH_i, the local MSS_p sends the low-priority checkpoint part to the recovering MH_i successively. After receiving the whole checkpoint, MH_i reloads the checkpoint to restore the system, and then resumes and replays the logs. During recovering, new message sent to MH_i is saved in its local MSS_p, and will be forwarded to MH_i, in turn, after recovering. The message that sequence number less than $CK\_info_i.Logm\_seq$ is discarded to avoid repeat messages.

If there is not CK_info_i in the local MSS_p, it means that a fault has occurred in other cell and then MH_i enters this cell in where it submits the recovery request. MSS_p broadcasts recovering request to all *MSS*s. The previous local MSS_q sends the high-priority part of recovery information to MSS_p. MSS_p executes the recovering process. The asynchronous recovery algorithm is described in Fig. 4.

Actions taken when MH_i occurs a fault:
Send (*RollbackReq*(i), MSS_p);
/*Sends recovery request to the local MSS_p.*/
Actions taken when MSS_p receives *RollbackReq*(i) from MH_i:
If ($MH_i \in$ (*Active_MH_List*$_p$ *or Disconnected_MH_List*$_p$))
Send (VP_0, MH_i);
/*if MSS_p holds latest checkpoint with high-priority, sends it to MH_i. */
Send (*Chkpt_retrieve* (*CK_info*$_i$.*CK_sn*), *CK_low*);
/* sends retrieval message to *CK_low* to get the remnant part of checkpoint.*/
Send (*Log_retrieve*(*CK_info*$_i$.*Logm_seq*), *CK_info*$_i$.*Log_queue*);
/* sends request to MSS_r in the list of *Cpinfo*$_i$.*Log_queue* to get message logs.*/
Else
Broadcast *info_retrieve*(i);
/* If the local MSS_p don't hold the latest CK_info_i, broadcasts the recovery request.*/
Actions taken when MSS_k receives Broadcast *info_retrieve*(i) from MSS_p:
Send (high-priority of $CK_{i,\alpha}$, CK_info_i , MSS_p);
/* sends the high-priority part of recovery information to local MSS_p.*/
Actions taken when MSS_q receives *Chkpt_retrieve*(*CK_sn*) from MSS_p:
Send (the remnant of $CK_{i,\alpha}$, MSS_p);
/*sends the remnant of checkpoint to local MSS_p.*/
Actions taken when MSS_q receives *Log_retrieve* (*Logm_seq*) from MSS_p:
If ($\exists Logm_{i,\alpha}$ in MSS_q log space and $\alpha \geq Logm\_seq$)
Send ($Logm_{i,\alpha}$, MSS_p);
/*sends related message logs to local MSS_p.*/
Actions taken when MH_i receives the full $CK_{i,\alpha}$ from MSS_p:
Restore $CK_{i,\alpha}$;
/*restores the full checkpoint.*/
Resume action;
/*starts the computation process of recovery.*/

Fig. 4. The asynchronous recovery algorithm

3.5 Garbage Collection

MH_i takes new checkpoint $CK_{i,\alpha}$ and sends the checkpoint and CK_info_i to MSS_p. MSS_p sends a message which a new checkpoint has been taken to all *MSS*s which will delete old checkpoint $CK_{i,\alpha\text{-}1}$ and relative information that denoted by old $CK\_info_i.CK\_loc$, $CK\_info_i.CK\_low$, $CK\_info_i.Log\_queue$. Every *MSS* will release the space held by checkpoint $CK_{i,\alpha\text{-}1}$ and implement garbage collection.

4 Correctness of the Algorithm

Theorem 1**.** The asynchronous recovery of MH_i is a consistent recovery.

Proof: *Recoverable*: the latest checkpoint and the messages with the sequence number larger than the $CK\_info_i.logm\_seq$ were saved safely due to the reliable communication, the reliable *MSS*s and the pessimistic message logging. Therefore, on the recovery of MH_i, every message logs and the latest checkpoint can be retrieved. The messages can be replayed according to the sequence number after restoring the latest checkpoint. In other words, MH_i can reconstruct one possible sequence of state intervals as those constructed before the fault due to processes following the piece-wise deterministic execution model. So MH_i is recoverable on fault in the strategy.

Consistent recovery: The lost events which incurs $L(i,f)$ can only be the messages or user inputs that had not been sent successfully to the local *MSS* before a fault. This implies the corresponding messages could not be transferred to their destinations. According to the *definition* 1, the lost events can not incur new dependency relations between *MH*s. Therefore, for any $I(i,\alpha) \in L(i,f)$ there exists no $I(j,\beta)$, such that $I(i,\alpha) \rightarrow I(j,\beta)$. The independent recovery is a consistent recovery as the recovery strategy satisfies the *definition* 3. □

5 Performance Study

The model and parameters in [6] are adopted. *MH*s communicate with *MSS*s through 802.11a wireless network adapter. *MH* moves from one cell to another follows a Poisson process with rate σ=0.01. The message sending rate of a *MH* follows a Poisson process with rate λ=0.1. Each *MH* takes a checkpoint with a fixed interval Tc=1000s, the failure rate of each *MH* follows an exponential distribution with rate δ=0.0001. Increment strategy is adopted for saving a checkpoint and its size is 1MB. The size of a logs entry is 50B. The ratio of wireless network speed to wired network speed is r=0.1. The time required to load a log entry through a wireless channel is T_1=0.016s, and the time required to load a checkpoint through a wireless channel is T_3=0.32s. The time required to execute a log entry is T_2=0.0008s. We assume that when a *MH* moves 5 times, its checkpoint should be moved to the new local *MSS* in *Frequency-based* strategy and when a *MH* moves 10 times, its checkpoint should be moved to the new local *MSS* in *Distance-based* strategy.

Fig. 5 shows the amounts of recovery information needed to be transmitted in every recovery strategies. The *y*-axis indicates the overhead of message transfer incurred by different strategies for *MH*'s recovery, while the *x*-axis denotes the time that a fault

occurs on the *MH*. The overhead of recovery information management under our virtual strategy is always less than those under *Eager*, *Frequency-based* and *Distance-based* strategy, and only a little large than that under *Lazy* strategy, because our compromise strategy only moves little, not the whole, latest checkpoint to the local *MSS* when a *MH* moves from one cell to another.

Fig. 6 shows the amounts of recovery information needed be transmitted by every strategy under different message sending rate after the system has run 500 seconds. The overhead of every handoff strategy becomes increment with the increment of message sending rateλ. The overhead of our handoff strategy almost equals to that under *Lazy* and far less than the other's.

Let $N(t)$ denotes the number of logs saved until *MH* faults. $f_f(t)$ denotes the probability when a fault occurs in time t. Under our and *Eager* strategies, the recovering probability of time T is [6]:

$$F_1(T) = \sum_{M=0}^{+\infty} \int_{MTc}^{(M+1)Tc} \Pr\{ N(t)[T_1 + T_2] + T_3 \le T \} f_f(t) dt \cdot \tag{4}$$

M denotes the checkpoint number experienced by a *MH* in time t, T_1, T_2 and T_3 denote the mean time of loading a log entry through wireless network, executing a log entry and loading whole checkpoint through wireless network respectively. $N(t)$ is a Poisson process with rate λ=0.1, and $f_f(t)$ is an exponential distribution with rate δ=0.0001, we get:

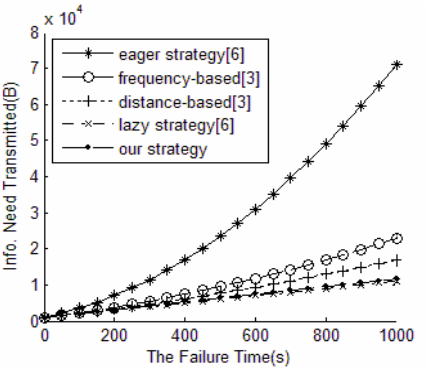


Fig. 5. The amounts of recovery information need transmitted

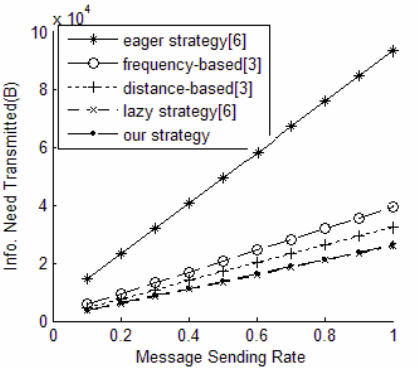


Fig. 6. The amounts of recovery information need transmitted with different rate

$$F_1(T) = \sum_{M=0}^{+\infty} \int_0^{Tc} \sum_{n=0}^{\left\lfloor \frac{T-T_3}{T_1+T_2} \right\rfloor} \frac{e^{-\lambda t'} (\lambda t')^n}{n!} \bullet \delta e^{-\delta t'} \bullet e^{-\delta MTc} dt'$$

$$= \frac{\int_0^{Tc} \sum_{n=0}^{\left\lfloor \frac{T-T_3}{T_1+T_2} \right\rfloor} \frac{e^{-\lambda t} (\lambda t)^n}{n!} \bullet \delta e^{-\delta t} dt}{1 - e^{-\delta Tc}} . \tag{5}$$

Under *Lazy* strategy, the recovering probability of time T is:

$$F_2(T) = \sum_{M=0}^{+\infty} \int_{MTc}^{(M+1)Tc} \Pr\{T_r \le T \mid t_f = t, k_b = k\} \bullet \Pr\{k_b = k \mid t_f = t\} f_f(t) dt \,. \tag{6}$$

Replacement $N(t)$ and $f_f(t)$ by their value:

$$F_2(T) = \sum_{M=0}^{+\infty} \int_0^{Tc} \sum_{k=0}^{+\infty} \sum_{n=0}^{\left\lfloor \frac{T-T_3-rT_3-rkT_3}{rT_1+T_1+T_2} \right\rfloor} \frac{e^{-\lambda t'}(\lambda t')^n}{n!} \bullet \frac{e^{-\sigma t'(\sigma t')^k}}{k!} \bullet \delta e^{-\delta t'} \bullet e^{-\delta MTc} dt'$$

$$= \frac{\int_0^{Tc} \sum_{k=0}^{+\infty} \sum_{n=0}^{\left\lfloor \frac{T-T_3-rT_3-rkT_3}{rT_1+T_1+T_2} \right\rfloor} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \bullet \frac{e^{-\sigma t(\sigma t)^k}}{k!} \bullet \delta e^{-\delta t} dt}{1 - e^{-\delta Tc}} \,. \tag{7}$$

Fig. 7 shows the probabilities of recovering time under various handoff strategies. Our strategy which only has a little overhead large than that under *Eager* is better than that under *Lazy* and has the same recovery probability as *Eager* has. Fig. 8 shows the executing overhead under our asynchronous recovery strategy and coordinated recovery strategy which the number of *MH*s is 100 and only 10 *MH*s need recovery from a fault. As shown in the figure, our strategy is better than coordinated strategy, and is more effective.

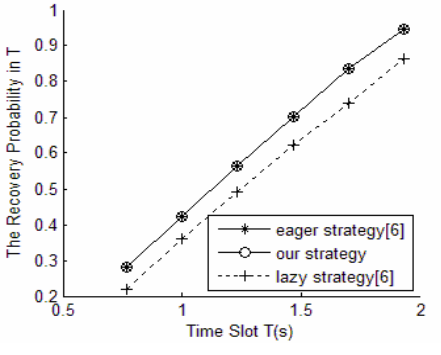


Fig. 7. The fault recovery probability

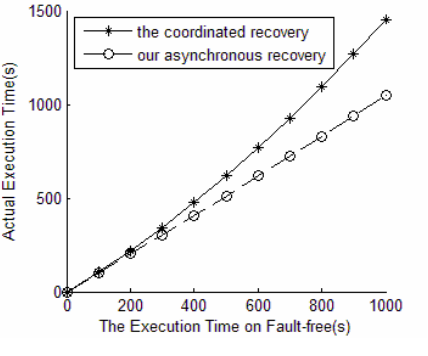


Fig. 8. The actual execution time of the mobile applications

6 Conclusion

The handoff strategy taken when a *MH* moves from one cell to another will affect the executing efficiency and recovering time of checkpoint algorithm. Different from other schemes, in the strategy proposed in the paper, the recovery information is

broken into two parts, which the first part must be transferred instantly to the new cell when a handoff happens and the second part can be transferred simultaneously to the local *MSS* through static network as the first part is transferred to the recovering *MH*. The partition principle is that the first part as little as possible and the second part as large as possible under guaranteeing recovering information to be transmitted to recovering *MH* successively. From the view of recovering *MH*, it seems that all recovery information resides on the local *MSS* all the time. This strategy considers both minimum executing time on fault-free and quickly recovering from a fault. Experiments and analysis show our strategy is better than others.

References

1. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys 34(3), 375–408 (2002)
2. Ching, E.Y., Phipatanasuphorn, V.: A Survey of Checkpoint-Recovery Techniques in Wireless Networks (2002), http://www.cae.wisc.edu/ ece753/papers/Paper_9.pdf
3. Park, T., Woo, N., Yeom, H.Y.: An Efficient Optimistic Message Logging Scheme for Recoverable Mobile Computing Systems. IEEE Transactions on Mobile Computing 1(4), 265–277 (2002)
4. Park, T., Woo, N., Yeom, H.Y.: An Efficient Recovery Scheme for Mobile Computing Environments. In: The 8th International Conference on Parallel and Distributed Systems (ICPADS), Kyongju City, Korea, pp. 53–60 (2001)
5. Pradhan, D.K., Krishna, P., Vaiday, N.H.: Recoverable Mobile Environment: Design and Trade-off Analysis. In: Proc. of the 26th Int'l Symp. on Fault Tolerant Computing System, Sendai, Japan, pp. 16–25 (1996)
6. Chen, I.-R., Gu, B., George, S.E., Cheng, S.-T.: On failure recoverability of client-server applications in mobile wireless environments. IEEE Transactions on Reliability 54(1), 115–122 (2005)

A Lightweight RFID Protocol Using Substring

Hung-Yu Chien and Chen-Wei Huang

Dept. of Information Management, National Chi Nan University, Taiwan, R.O.C.
{hychien,s95213508}@ncnu.edu.tw

Abstract. As low-cost RFIDs with limited resources will dominate most of the RFID market, it is imperative to design lightweight RFID authentication protocols for these low-cost RFIDs. However, most of existing RFID authentication protocols either suffer from some security weaknesses or require costly operations that are not available on low-cost tags. In this paper, we analyze the security vulnerabilities of a lightweight authentication protocol recently proposed by Li et al. [4], and then propose a new lightweight protocol to improve the security.

Keywords: RFID, authentication, low-cost cryptography, tracing, reader, DOS attack.

1 Introduction

A Radio Frequency Identification (RFID) system mainly consists of three components: radio frequency tags, readers, and a backend server/database (or a set of distributed databases) which maintains information on the tagged objects. Generally, the tag consists of a microchip with some data storage and an antenna. A reader queries tags to obtain tag contents though wireless communications.

Recently, the wide deployment of RFID systems in a variety of applications has raised many concerns about the privacy and the security. An RFID tag can be attached to a product, an animal, or a person for the purpose of identification using radio waves. For any possible reasons, an adversary may perform various attacks such as eavesdropping, traffic analysis, spoofing, disabling the service, or disclosing sensitive information of tags, and hence infringes people's privacy and security.

Even though RFID tags with full-fledged capacity are available, to attain great market penetration, RFID tags should be low-cost, which limit the computation power, the storage space, the communication capacity and the gates count. As studied by the previous work like [2], a low-cost RFID tag has approximately 4,000 logic gates. Although there have been many works devoted to design security mechanisms for low-cost RFIDs, most of these works require the tags to be equipped with costly operations such as one-way hashing functions [1, 3, 5], which are still un-available on low-cost tags. Contrary to these works, the schemes [4, 6, 8, 9] do not require the support of hashing functions on tags. However, the schemes [6, 8, 9] have been reported to show some security weaknesses [6, 7]. Recently, Li et al. [4], based on only bitwise XOR ($\oplus$), the Partial ID concept and pseudo random numbers, proposed a lightweight RFID authentication protocol for low-cost RFIDs. Different from most

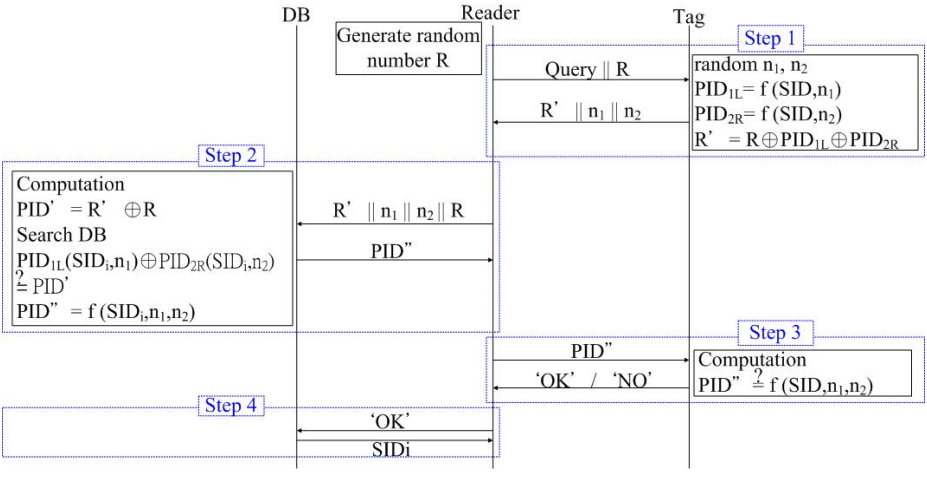
T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 422–431, 2007.

of existing solutions like [1, 3, 5] which used conventional cryptographic primitives (encryptions, hashing, etc), this protocol only used simple operations like XOR and substring. Unfortunately, we find that Li et al.'s scheme has several security weaknesses. In this paper, we shall analyze the security weaknesses of Li et al.'s RFID authentication protocol. To heal the weaknesses while preserving the lightweight feature, we propose a new RFID authentication protocol.

The rest of this paper is organized as follows. Section 2 reviews Li et al. lightweight RFID authentication protocol. Section 3 analyzes the vulnerabilities of Li et al.'s scheme. Section 4 proposes a new RFID authentication protocol that heals the security weaknesses while preserving the lightweight feature for low-cost RFID tags. Section 5 analyzes the security of our proposed protocol. Finally, conclusion remarks and future work are drawn in Section 6.

2 Review of Li et al.'s Scheme

The tags in Li et al.'s RFID authentication protocol [4] use only bitwise XOR ($\oplus$), the partial ID concept and pseudo random numbers. Costly operations such as multiplications and hash functions are eliminated in the design. In Li et al.'s scheme, each tag and the backend server share an l-bit secret information, *SID* (the secure ID). During the authentication the tag generates two random numbers n_1 and n_2 such that $2l \geqq n_1+n_2 \geqq l/2$. The two random numbers are used in the substring function f to extract the partial *ID*s, PID_{1L} and PID_{2R}, where PID_{1L} denotes the left substring of *SID* and PID_{2R} denotes the right substring of *SID*. That is, let $f(SID, i, j)$ denotes the substring of *SID* starting from position i to position j, then $PID_{1L} = f(SID, 1, n_1)$ and $PID_{2R} = f(SID, n_2, l)$. Li et al.'s scheme is depicted in figure 1.



SID: Secure ID *DB*: Backend Database/server
PID: Partial ID $\oplus$: XOR operation
f: substring function

Fig. 1. Li et al.'s scheme

The scheme consists of four stages: the *PID* generating stage, the *SID* searching and tag authentication stage, the reader authentication stage and the result returning stage.

- ***PID generating stage*:** The reader generates a random number R, and then sends it to the tag. Upon receiving the probe from the reader, the tag uses two random numbers n_1, n_2 and the substring function f to compute $PID_{1L}= f(SID, 1, n_1)$, $PID_{2R} = f(SID, n_2, l)$ and $R'= R \oplus PID_{1L} \oplus PID_{2R}$. The tag then responds the data R', n_1 and n_2 to the reader.
- ***SID searching and tag authentication* stage:** The reader sends R', R, n_1 and n_2 to the server. The server computes $PID'= R' \oplus R$, and iteratively picks up one candidate *SID'* from the database to check whether $PID'_{1L} \oplus PID'_{2R} = PID'$, where $PID'_{1L} = f(SID', 1, n_1)$ and $PID'_{2R} = f(SID', n_2, l)$. If a match is found, then the selected *SID*' is the tag's identification; otherwise, it continues the process until a match is found or responds with "failure" if no match could be found in the whole database. If a match is found, it computes $PID"= f(SID', n_1, n_2)$ and then sends it to the reader.
- ***Reader authentication* stage:** The reader sends *PID"* to the tag, which then checks whether $f(SID, n_1, n_2)$ equals *PID"* to authenticate the reader. After the reader is authenticated successfully, the tag sends 'OK' to the reader; otherwise, it responds with "no find" information.
- ***Result returning* stage:** If the reader receives 'OK', and then sends it to the server, which will transmit the *SID* to the reader. Otherwise the reader stops the protocol.

3 Vulnerabilities of Li et al. Scheme

In this section, we remark that Li et al.'s scheme is vulnerable to replay attack and is prone to reveal the secret information *SID*.

3.1 The Replay Attack

An adversary can easily eavesdrop on the communications from a legal tag, modify the data, and then replay the messages to masquerade as the legal tag as follows. The attack consists of two stages- the data deriving stage and the spoofing stage.

- **The data deriving stage:** The adversary records the communication (R, R', n_1, n_2) from a tag (say T_a), and then derives $PID_{1L} \oplus PID_{2R}$ from $R' \oplus R$.
- **The spoofing stage:** In this stage, the adversary uses the derived data $PID_{1L} \oplus PID_{2R}$ to masquerade as the tag T_a as follows.
  1. Upon receiving the probe Query$\| \overline{R}$ from the reader, the adversary computes $R' = PID_{1L} \oplus PID_{2R} \oplus \overline{R}$, and responds with $R' \| n_1 \| n_2$ to the reader.
  2. It is easy to see that the forged data $R' \| n_1 \| n_2$ will be accepted by the server, and the reader will forward the data $PID"$ from the server to the adversary.

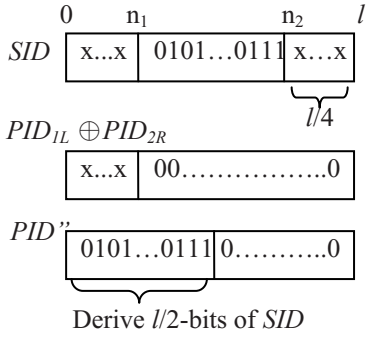
3. The adversary just records the data *PID*" and always responds with "OK" to the reader. We can see that the reader finally accepts this spoofing tag as the genuine tag T_a.

3.2 Disclosing the Secret Value *SID*

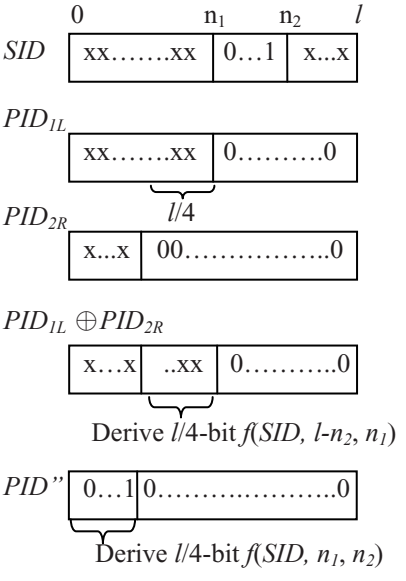
Since an adversary can eavesdrop on the communications and record the data *R'*, *R*, n_1, n_2 and *PID"*, and he can compute $R' \oplus R$ to obtain $PID_{1L} \oplus PID_{2R}$. With n_1, n_2, $R' \oplus R$ and *PID"*, an adversary can derive partial information of *SID*, and can repeatedly run the process many times to fully disclose all the bits of *SID* or derive partial information of *SID* (if most bits of the identification are known, then it is highly possible to guess the rest bits because the identification of a tag- for example, the EPC code- has a pre-defined format). In the following, we describe the single run of our attack process, and examine some cases to point out the vulnerabilities of Li et al.'s scheme.

As the lengths of PID_{1L}, PID_{2R} and *PID"* are unequal to l bits, we assume that 0s are padded to them such that each length of them equals l-bit in the following scenario (we can also assume 1s are padded to these strings, and the same attack still works). Based on the values $R' \oplus R = PID_{1L} \oplus PID_{2R}$ and $PID" = f(SID, n_1, n_2)$, an adversary can derive parts of *SID*. The length of the disclosed part of *SID* depends on the values of n_1 and n_2. With $2l \geqq n_1 + n_2 \geqq l/2$ property, the values of n_1 and n_2 generally have four situations. Firstly, if $n_1 = l - n_2$, an adversary can derive $f(SID, n_1, n_2)$. Secondly, if $n_1 > l - n_2$, an adversary can derive $f(SID, n_1, n_2)$ and $f(SID, l-n_2, n_1)$. Thirdly, if $n_1 < l - n_2$, an adversary can derive $f(SID, n_1, n_2)$ and $f(SID, n_1+n_2, l)$. Finally, if $n_1 = l$, $l - n_2 = 0$, an adversary can obtain all of *SID*. Some example cases are discussed as follows.

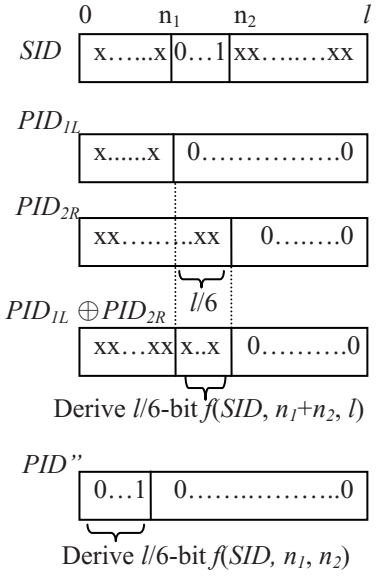
Example 1. Deriving parts of *SID*. Assume $n_1 = l/4$ and $n_2 = 3/4\ l$, an adversary can directly derive the $l/2$-bits $f(SID, n_1, n_2)$ from *PID*".



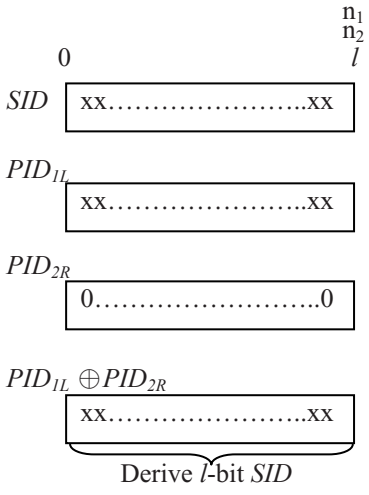
Example 2. Deriving parts of PID_{1L}. Assume $n_1 = l/2$ and $n_2 = 3/4\ l$, an adversary can derive $l/4$-bit $f(SID, l-n_2, n_1)$ from $PID_{1L} \oplus PID_{2R}$, and derive $l/4$-bit $f(SID, n_1, n_2)$ from *PID"* as follows.



Example 3. Deriving parts of PID_{2R}. Assume n_1 =1/3 l and n_2 = l/2, an adversary derives l/6-bit $f(SID, n_1+n_2, l)$ from $PID_{1L} \oplus PID_{2R}$, and derive l/6-bit $f(SID, n_1, n_2)$ from *PID"* as follows.



Example 4. Deriving all the bits of *SID*. Assume $n_1 = l$ and $n_2 = l$, an adversary can obtain all the bits of *SID* as follows.



In a single run of the above attack, an adversary can derive partial information of *SID*, and he can launch the above attack several times to aggregate the partial information of *SID* or even derive all the bits of *SID*. Although it is possible that part of the *SID* can not be directly derived in the above process, one might guess the rest bits, because the identifications of tags are usually with fixed format.

4 A New Lightweight RFID Authentication Protocol

In this section, we propose a new protocol to improve the security while preserving the lightweight property. Our proposed protocol is depicted in figure 2 and described as follows.

We assume that each tag and the database share an l-bit secret key x, $x = x_0x_1.....x_{l-1}x_l$. The reader generates a random number R_1, and the tag generates a random number R_2. Some notations are introduced as follows.

$g(z)$: $g()$ is a random number generator, and z is an input number.

$\tilde{g}$: the random output of $g(z)$.

rotate(p, w): *rotate* denotes the bitwise left rotation operator, and the operand p is rotated w positions.

Left(s): the left half of s.

Right(s): the right half of s.

Step1: The reader generates a random number R_1, and then sends it to the tag. Upon receiving the probe from the reader, the tag generates another random number R_2, computes $\tilde{g} = g(R_1 \oplus R_2 \oplus x)$ and rotates its *SID* to obtain *SID'* = *rotate*(*SID*, $\tilde{g}$). It calculates *R'*=*Left*(*SID'* $\oplus$ $\tilde{g}$), and responds the data *R'* and R_2 to the reader.

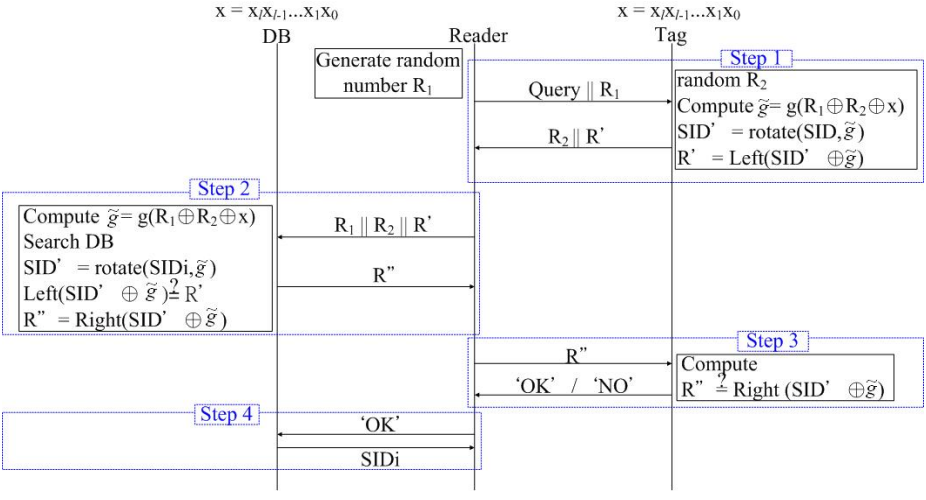


Fig. 2. Our proposed protoco1

Step2: The reader forwards R_1, R_2, and R' to the server. The server iteratively picks up one candidate *SID* from the database, computes $\tilde{g} = g(R_1 \oplus R_2 \oplus x)$ and $SID' = rotate(SID, \tilde{g})$ and checks whether $Left(SID' \oplus \tilde{g}) = R'$. If a match is found, then the selected *SID* is taken as the tag identification; otherwise, it continues the process until a match is found or responds with "failure'" if it cannot find a match in the whole database. If a match is found, it computes $R''=Right(SID' \oplus \tilde{g})$ and then sends it to the reader.

Step3: The reader sends R'' to the tag, which then checks whether $Right(SID' \oplus \tilde{g})$ equals R'' to authenticate the reader. After the reader is authenticated successfully, the tag sends 'OK' to the reader; otherwise, it responds with "no find" information.

Step4: If the reader receives 'OK', and then sends it to the server, which then transmits the *SID* to the reader. Otherwise the reader stops the protocol.

During singulation, if multiple tags respond simultaneously to a query, they will interfere with each other. Therefore, we suggest that an anti-collision algorithm like the binary tree-walking [3] could be used in our proposed protocol to solve the problem of collisions.

5 Analysis

5.1 Security Analysis

We now analyze the security of the proposed scheme as follows.

- **No traceability.** During each authentication instance, an adversary can only observe the values (R_1, R_2, R', R''), where R_1, R_2 are random numbers and

R'/R'' are respectively the left/right half bits of the random string $SID' \oplus \tilde{g}$. No identity-related information can be derived from these values, and these values are distinct and look random to an adversary. So, an adversary cannot trace the tags.

- **Mutual authentication.** The server authenticates the tag by verifying the substring *Left*($SID' \oplus \tilde{g}$), and the tag authenticates the server by verifying the substring *Right*($SID' \oplus \tilde{g}$). Since only the genuine tag and the server who have the secret key x can generate and verify the values, the scheme provides mutual authentication.
- **Replay attack prevention.** An adversary could eavesdrop on the communications between the reader and the tag. However, the substring *Left*($SID' \oplus \tilde{g}$) and the substring *Right*($SID' \oplus \tilde{g}$) should depend on the random challenges R_1, R_2 , and replay messages cannot satisfy the verification either by the reader or by the tag.
- **DOS attack prevention.** In some previous schemes, the technique of varying pseudonyms is used to resist tracing, and these schemes need to synchronize the pseudonyms between the server and the tags; otherwise, they are unable to authenticate each other. In our scheme, there is no requirement of state synchronization. Therefore, it can resist the DOS attack.

In *Table 1*, we show a comparison of the security with previous mentioned schemes [1, 2, 3, 5].

Table 1. Comparison between schemes

Protocol	RHLK[3]	HBIV[1]	SRAC[5]	LCAP[2]	Li et al.[4]	Our scheme
No traceability	x	x	x	o	o	o
Mutual Authentication	o	o	o	o	o	o
Replay attack prevention	x	x	x	o	x	o
DOS attack prevention	x	o	x	x	o	o

5.2 Performance Analysis

It is important to minimize the storage cost, the computational cost and the communication cost of low-cost tags. In *Table 2*, we examine the performance of our scheme in terms of storage space, computational cost and communication cost, and compare it with the previous schemes [1, 2, 3, 5].

- **Storage space:** In our scheme, the tag has to store its tag *ID* of length l and an l-bit secret key. For identifying the tag, the database has also to store related information. Therefore, implementation of our scheme, the tag and the database both only require $2l$ bits of memory, which is suitable to low-cost tags.

- **Computational cost:** In Henrici-Müller's scheme [1], Lee et al.'s scheme [2], Weis et al.'s scheme [3] and Lee-Verbauwhede's scheme [5], the tag has to be equipped with hash functions, which are still un-available on low-cost tags. On the contrary, in our scheme, the tag only needs random number generation, XOR, shifting, and substring function. The computations are very efficient and lightweight.
- **Communication cost:** In our scheme, messages of tag-to-reader communication are R_2 and R' with a total of $1\frac{1}{2}l$ bits and a message of reader-to-tag communication is R'' with $\frac{1}{2}l$ bits. Compared to Henrici-Müller's scheme [1], Weis et al.'s scheme [3] and Lee-Verbauwhede's scheme [5], the communication performance of our scheme is more efficient.

Table 2. Performance analysis

Protocol		RHLK[3]	HBIV[1]	SRAC[5]	LCAP[2]	Li et al.[4]	Our scheme
Storage.	Tag	l	$3l$	l	l	l	$2l$
	Reader	-	-	-	-	-	-
	Database	nl	$10l$	$4l$	$6l$	l	$2l$
Comp.	Tag	$1h$	$3h$	$2h$	$2h$	2(XOR)	3(XOR)
	Reader	$1h$	-	$2h$	-	-	-
	Database	-	$3h$	-	$2h$	2(XOR)	4(XOR)
Comm.	Tag-to-Reader	$2l$	$3l$	l	$1\frac{1}{2}l$	$l+\alpha$	$1\frac{1}{2}l$
	Reader-to-Tag	l	$2l$	$2l$	$\frac{1}{2}l$	β	$\frac{1}{2}l$

Notations of Table: l – size of required memory, h – the cost of a hash function operation, α – between l /2 and $2l$, β – less than l.

6 Conclusion and Future Work

This paper has shown the replay attack and the secret disclosure problem of Li et al.'s scheme. In the attack, an adversary can easily derive partial information of the secret *SID* or even all the bits of the *SID*. We also have proposed a new lightweight RFID authentication protocol, which improves the security, the communication performance and the computational performance. And taking into account that low-cost tags are highly resource-constrained, the tags only need to store tag's *ID* of length l and an l-bit secret key. So it can easily be implemented on those low-cost RFIDs like EPC generation 2 RFID.

In Our proposed protocol, it doesn't offer forward secrecy since the key updating is not fulfilled after the mutual authentication. But, we find that the previous re-keying protocols like [1, 2, 5, 8, 9] all suffer from DOS attacks. Furthermore, designing a protocol that simultaneously ensures forward secrecy and DOS attack resistance is our future work.

Acknowledgments. This research is partially supported by National Science Council with project number NSC95-2221-E-260-050-MY2.

References

1. Henrici, D., Müller, P.: Hash-based Enhancement of Location Privacy for Radio-Frequency Identification Devices using Varying Identifiers. In: PerSec 2004 at IEEE PerCom (2004)
2. Lee, S.M., Hwang, Y.J., Lee, D.H., Lim, J.I.: Efficient Authentication for Low-Cost RFID Systems. In: Gervasi, O., Gavrilova, M., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3483, Springer, Heidelberg (2005)
3. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In: International Conference on Security in Pervasive Computing (March 2003)
4. Li, Y.Z., Cho, Y.B., Um, N.K., Lee, S.H.: Security and Privacy on Authentication Protocol for Low-cost RFID. In: IEEE International Conference on Computational Intelligence and Security, vol. 2, pp. 1101–1104 (November 2006)
5. Lee, Y.K., Verbauwhede, I.: Secure and Low-cost RFID Authentication Protocols. Adaptive Wireless Networks–AWiN (November 2005)
6. Chien, H.Y., Chen, C.H.: Mutual Authentication Protocol for RFID Conforming to EPC Class 1 Generation 2 Standards. Computers Standards and Interfaces 29(2), 254–259 (2007)
7. Lin, C.-L., Chang, G.-G.: Cryptanalysis of EPC Class 1 Generation 2 RFID authentication. In: Information Security Conference 2007, ChiaYi, Taiwan (2007)
8. Duc, D.N., Park, J., Lee, H., Kim, K.: Enhancing Security of EPCglobal Gen-2 RFID Tag against Traceability and Cloning. In: The 2006 Symposium on Cryptography and Information Security (2006)
9. Karthikeyan, S., Nesterenko, M.: RFID security without extensive cryptography. In: Workshop on Security of ad hoc and sensor networks, pp. 63–67. ACM Press, Alexandria, Virginia, USA (2005)

The Reliability of Detection in Wireless Sensor Networks: Modeling and Analyzing

Ming-Tsung Hsu[1], Frank Yeong-Sung Lin[1], Yue-Shan Chang[2], and Tong-Ying Juang[2]

[1] Dept. of Information Management, NTU,
No. 1, Sec. 4, Roosevelt Road, Taipei, 10617 Taiwan
{d94004,yslin}@im.ntu.edu.tw
[2] Dept. of Computer Science and Information Engineering, NTPU,
No. 151, University Rd., San Shia, Taipei, 237 Taiwan
{ysc,juang}@mail.ntpu.edu.tw

Abstract. A Wireless Sensor Network (WSN) composed of tiny sensor nodes may operate in an unfavorable terrain. The coupling of inherent limitations and harsh environments makes WSNs fallible. For this reason, reliability becomes one of the most important issues in WSN research. Some of the early work in the field of detection reliability focuses on collaborative effort. Instead of the collaborative work, the sensing improvements are proposed for detection reliability enhancement. Two types of detection models are constructed based on the scenarios of WSN operations for probability decomposition. The fault probability of detection and the probability of detection reliability in WSNs can then be estimated based on the decomposition of probabilities and empirical data. In analyzing the decomposition of probabilities, sensing improvements are shown to enhance detection reliability. An illustrative example is demonstrated to show how detection reliability can be controlled by different sensing improvements in different application situations.

Keywords: Wireless sensor networks, Detection models, Detection reliability, Fault probability of detection, Sensing improvements.

1 Introduction

Recently, WSNs are developed and used for information collection [1], [7]. Including environmental monitoring, automatic controlling, and target tracking, WSN applications all have a data collection task. A tiny sensor node equipped with multifunctional sensors, a micro-processor, and a radio transceiver is responsible for this task.

The reliability becomes one of the most important issues in WSN research since sensor nodes are usually deployed in unattended and unfavorable environments, which makes each component of sensor nodes fault or crash easily. The techniques and mechanisms for the operations of sensing, processing, and communication are necessarily aware of this essential fact to maximize the reliability of WSNs. In this paper, sensing (detection) reliability is discussed in detail.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 432–443, 2007.

Faults in the sensing system, which is responsible for sensing environmental energies, may be caused by the hardware or software failure; may be produced by environmental noise; may last for a short or long time period; and can make the behavior of sensor nodes inactive or arbitrary. The results of sensing faults, such as the missing detection, false alarm, and unusual reading, can affect the data collection task severely, so they must be effectively overcome.

In most WSN applications, sensor nodes only send detection decisions or reports to a sink or a fusion center for energy conservation. For detection reliability improvement, the collaborative effort of a large number of sensor nodes is proposed previously [1], [6], [11], [13], [20]. Instead of the collaborative work, detection reliability is estimated by the analysis of detection models and enhanced by proposed sensing improvements in this paper.

In a detection-based (event-driven) WSN, there are four possible scenarios of a sensor node: (i) the sensor node misses an interesting event; (ii) the sensor node issues a false alarm; (iii) the sensor node accurately reports an interesting event; and (iv) the sensor node faultily reports an interesting event [11]. These scenarios show that the interesting event and detection error result in detection and there are four types of events (missing detection, false alarm, validity detection, and hidden fault detection) in the detection process.

The sensing system can be enhanced in different ways to increase detection reliability, e.g., increasing sensing capability may reduce the missing detection while increasing error resistance capability can avoid the false alarm. In this paper, four types of sensing improvements (including sensibility, dependability, effectiveness, and resistiveness) are theoretically defined.

The remainder of this paper is organized as follows: The detection models are constructed in Section 2. The fault probability of detection and the probability of detection reliability are also estimated in this section for detection models. Section 3 defines the sensing improvements and shows how to improve detection reliability theoretically. An illustrative example is depicted in Section 4 to demonstrate the effect of sensing improvements. Section 5 briefly reviews the related work of reliability in WSNs. Section 6 draws our conclusions and future work.

2 Detection Models

In this section, we first construct two types of detection models by the scenarios of WSN operations as mentioned previously. The fault probability of detection ($P(F_D)$) and the probability of detection reliability ($P(R)$) are also defined by the scenarios of WSN operations. $P(F_D)$ and $P(R)$ can be estimated based on the decomposition of probabilities or the observation of missing detection (M), false alarm (F), and hidden fault detection (H) in different detection models.

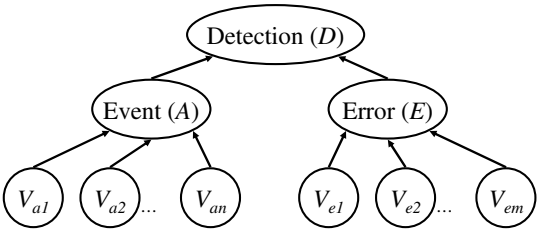
2.1 Model Construction

As introduced in Section 1, the interesting event (A) and detection error (E) lead to detection (D). Since A results from the environmental factors whose

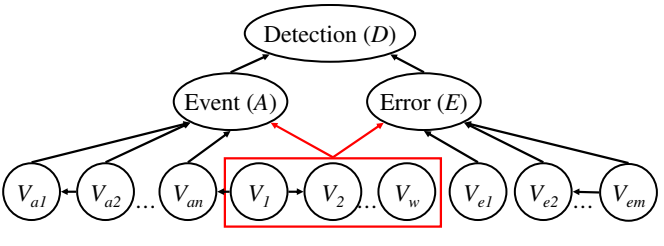
energies are sensed by sensor nodes and E also results from the environmental factors which make sensor nodes dysfunctional, A and E are thus both caused by environmental factors.

In an Independent Detection Model (IDM), A and E are affected by different environmental factors and these factors are mutually independent. Fig. 1(a) shows the structure of IDM. The intrusion detection system is an example of IDM, where infrared sensors sense the heat of objects (the environmental factor of intrusion events) and can be affected by high temperature and moisture (the environmental factors of errors) [1], [2].

The Conditionally Independent Detection Model (CIDM) is that A and E can be both affected by common environmental factors and/or the environmental factors affecting A and E are not mutually independent. Fig. 1(b) shows the structure of CIDM, where A and E are both affected by common environmental factors $(V_1, V_2, \ldots, V_w)$ and the environmental factors affecting A and E are not mutually independent. The forest fire tracking system is an example of CIDM where a fire event is detected and tracked as the temperature and humidity (the environmental factors of fire events) sensed by thermometers and hygrometers are both high, and an error may also be produced as the high temperature seriously affecting thermometers and hygrometers [1], [8].



(a) IDM: A and E lead to D and are affected by different and mutually independent environmental factors. The $(V_{a1}, V_{a2}, \ldots, V_{an})$ and $(V_{e1}, V_{e2}, \ldots, V_{em})$ are the environmental factors affecting A and E, respectively.



(b)CIDM: A and E lead to D and are both affected by common environmental factors and/or the environmental factors affecting A and E are not mutually independent.

Fig. 1. Detection models

2.2 Fault Probability of Detection

As introduced in Section 1, the operation of detection-based WSNs exists four possible events where M and F are widely discussed in the Signal Detection Theory (SDT). The fault probability of detection in SDT is defined as follows [20]:

Definition 1. *The fault probability of detection is defined by*

$$P(F_D) = P(A)P(\bar{D}|A) + P(\bar{A})P(D|\bar{A}) = P(A)P(M) + P(\bar{A})P(F) \ . \tag{1}$$

Based on Bayesian theorem and detection models, $P(M)$ can be decomposed as (2a) and (2b) for CIDM and IDM, respectively.

$$P(M) = \frac{P(\bar{D}|AE)P(AE) + P(\bar{D}|A\bar{E})P(A\bar{E})}{P(A)} \tag{2a}$$

$$P(M) = P(\bar{D}|AE)P(E) + P(\bar{D}|A\bar{E})P(\bar{E}) \ . \tag{2b}$$

In a similar manner, $P(F)$ of CIDM and IDM can also be decomposed based on Bayesian theorem as shown in (3a) and (3b), respectively.

$$P(F) = \frac{P(D|\bar{A}E)P(\bar{A}E) + P(D|\bar{A}\bar{E})P(\bar{A}\bar{E})}{P(\bar{A})} \tag{3a}$$

$$P(F) = P(D|\bar{A}E)P(E) + P(D|\bar{A}\bar{E})P(\bar{E}) \ . \tag{3b}$$

2.3 Probability of Detection Reliability

$P(F_D)$ is focused for detection reliability in most of previous work [6], [11], [13], [17], [20]. For example, in a WSN, $P(A)$ is 0.05 while $P(M)$ and $P(F)$ of senor nodes are 0.05 and 0.008, respectively. By Definition 1, $P(F_D)$ is 0.0101 and detection reliability might be treated as 0.9899.

It must be noted that errors are essential facts in WSNs and therefore, the correct detection $(D|A)$ can be differentiated into the validity detection and hidden fault detection. However, sensor nodes which faultily report interesting events should be considered as faulty. Considering that the hidden fault detection $(DE|A)$ may exist, the probability of reliability is defined as follows:

Definition 2. *The probability of detection reliability is defined by*

$$P(R) = 1 - (P(M) + P(H))P(A) - P(F)P(\bar{A}) \ . \tag{4}$$

The decomposition of $P(H)$ of CIDM and IDM by Bayesian theorem are shown in (5a) and (5b), respectively.

$$P(H) = \frac{P(D|AE)P(AE)}{P(A)} \tag{5a}$$

$$P(H) = P(D|AE)P(E) \ . \tag{5b}$$

2.4 Probability Estimation

To compute the probabilities in the subsection 2.2 and 2.3, $P(A)$, $P(E)$, and probabilities of D given A and E must be known.

$P(A)$ can be computed in (6a) and (6b) for CIDM and IDM, respectively, if all environmental factors affecting A are sensed. For the computing purpose, the value of environmental factors is assumed to be divided into levels.

$$P(A) = \sum_i \cdots \sum_h P(A|V_{a1,i} \cdots V_{w,h}) P(V_{a1,i} \cdots V_{w,h}) \tag{6a}$$

$$P(A) = \sum_i \cdots \sum_l P(A|V_{a1,i} \cdots V_{an,l}) P(V_{a1,i}) \cdots P(V_{an,l}) \tag{6b}$$

where $V_{an,i}$ and $V_{w,h}$ are the environmental factor V_{an} with level i and the environmental factor V_w with level h, respectively.

In a similar manner, $P(E)$ of CIDM and IDM can be computed by (7a) and (7b), respectively.

$$P(E) = \sum_i \cdots \sum_h P(E|V_{e1,i} \cdots V_{w,h}) P(V_{e1,i} \cdots V_{w,h}) \tag{7a}$$

$$P(E) = \sum_i \cdots \sum_l P(E|V_{e1,i} \cdots V_{em,l}) P(V_{e1,i}) \cdots P(V_{em,l}) \tag{7b}$$

where $V_{em,i}$ is the environmental factor V_{em} with level i.

In practical applications, $P(A)$ and $P(E)$ cannot be computed theoretically since some of environmental factors are difficult to sense. Instead of theoretically computing, the t-out-of-n rule [20] can be used to estimate $P(A)$ as sensor nodes are deployed densely. $P(M)$ and $P(F)$ of sensor nodes can also be estimated by counting the number of missing detection and false alarms [9]. $P(H)$ can be estimated as detection reports contain the energy readings of sensors. Since the error rate (including missing detection, false alarm, and unusual reading) of sensor nodes also can be computed by t-out-of-n rule, $P(E)$ can be estimated as the average error rate of all sensor nodes. $P(F_D)$ and $P(R)$ then can be estimated in practical applications as $P(A)$, $P(E)$, $P(M)$, $P(F)$, and $P(H)$ are known.

3 Theoretical Analysis

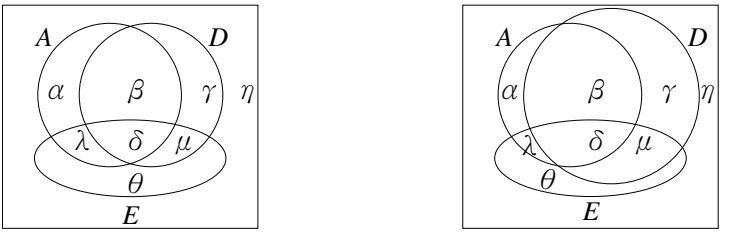
This section theoretically analyzes $P(M)$, $P(F)$, and $P(H)$ by proposed sensing improvements to minimize $P(F_D)$ and to maximize $P(R)$. For simplicity of analysis, Fig. 2, which illustrates the relationship among A, E, and D, is used in this section.

3.1 Sensing Improvements

The obvious method for improving $P(F_D)$ and $P(R)$ is that sensor nodes must be reinforced to resist the environmental interference and uncertainty. Based on the (2a), (2b), (3a), (3b), (5a), and (5b), the sensing improvements of sensor nodes can be classified as follows:

1. The sensibility improvement (S_S): the sensibility is the capability of a sensor node that it can report detection when only event occurs ($D|A\bar{E}$).
2. The dependability improvement (S_D): the dependability is the capability of a sensor node that it will not report detection when both error and event do not occur ($\bar{D}|\bar{A}\bar{E}$).
3. The effectiveness improvement (S_E): the effectiveness is the capability of a sensor node that it can report detection when both event and error occur ($D|AE$).
4. The resistiveness improvement (S_R): the resistiveness is the capability of a sensor node that it will not report detection when only error occurs ($\bar{D}|\bar{A}E$).

The capability of S_S, S_D, S_E, and S_R can be improved by decreasing the area of α, γ, λ, and μ or increasing the area of β, η, δ, and θ. In practical applications, these sensing improvements might be the trade-off and might not be improved simultaneously, e.g., improving S_S (increasing the area of D in Fig. 2(a)) might make S_R degraded (μ is increased and θ is decreased as in Fig. 2(b)).



(a) Relationship among A, E, and D. (b) Increasing S_S makes S_R degraded.

Fig. 2. Relationship among A, E, and D

3.2 Probability Analysis

There are two parts in (2a) and (2b) for $P(M)$: one is the environmental interference ($\bar{D}|AE$) while another is the uncertainty ($\bar{D}|A\bar{E}$). Enhancing S_E can reduce the environmental interference and enhancing S_S can reduce the uncertainty.

Similarly, the reduction of $P(F)$ can be achieved when the S_R and S_D of sensor nodes can be enhanced to reduce the environmental interference ($D|\bar{A}E$) and the uncertainty ($D|\bar{A}\bar{E}$), respectively.

In IDM, $P(E)$ is the multiplier of sensing improvements in $P(M)$ and $P(F)$, which determines the efficiency of sensing improvements. As $P(E)$ is small, the effect of S_S is more than that of S_E for $P(M)$ and the effect of S_D is more than that of S_R for $P(F)$. Unlike IDM, the multiplier of sensing improvements in $P(M)$ and $P(F)$ for CIDM is the joint probabilities of A and E divided by the $P(A)$ or $P(\bar{A})$.

As shown in (1), $P(F_D)$ is the function of $P(M)$, $P(F)$, and $P(A)$, where $P(A)$ is the multiplier of $P(M)$ and $P(F)$. To reduce $P(F_D)$ needs to simultaneously reduce both of $P(M)$ and $P(F)$ in Bayesian detection problem [20]. Therefore, S_S, S_D, S_E, and S_R all affect $P(F_D)$ while $P(A)$ and $P(E)$ are the multiplier of sensing improvements in IDM and the joint probabilities of A and E are the multiplier of sensing improvements in CIDM.

$P(H)$ can be improved in (5a) and (5b) for CIDM and IDM, respectively, as S_E degraded. Then, $P(R)$ can only be affected by S_S, S_D, and S_R since S_E can reduce the environmetal interference in $P(M)$ but can also increase $P(H)$.

S_S, S_E, and S_R can be directly estimated from observation of empirical data, which is the same as discussed in subsection 2.4. S_D can be estimated by (3a) and (3b) for CIDM and IDM, respectively, as $P(F)$, $P(E)$, $P(A)$, the joint probabilities of A and E, and S_R are known.

4 Illustrative Example

Although Section 3 shows how to theoretically enhance $P(F_D)$ and $P(R)$, the impact of sensing improvements on $P(F_D)$ and $P(R)$ still needs to discern clearly. This section illustrates an example of IDM to show the impact of different sensing improvements in different application situations.

4.1 Intrusion Detection System

As mentioned in subsection 2.1, the intrusion detection system is an example of IDM. In an intrusion detection system, the prior probability used for probability computation can be obtained by empirical data or training data as discussed in subsection 2.4 and in [5], [9].

Based on the t-out-of-n rule, the probability of intrusion is observed by counting the times of intruder's arrival divided by the total time intervals in the empirical data, where at most one intrusion will occur in each interval.

$P(M)$ and $P(F)$ can be estimated by the average of missing detection and false alarm probabilities of each sensor node as shown in [9]. $P(H)$ can also be estimated by the observation as detection reports contain the energy readings. $P(E)$ can be estimated as the average error rate of all sensor nodes. The ratio of sensing improvements can be obtained as discussed in subsection 3.2. Table 1 lists these probabilities.

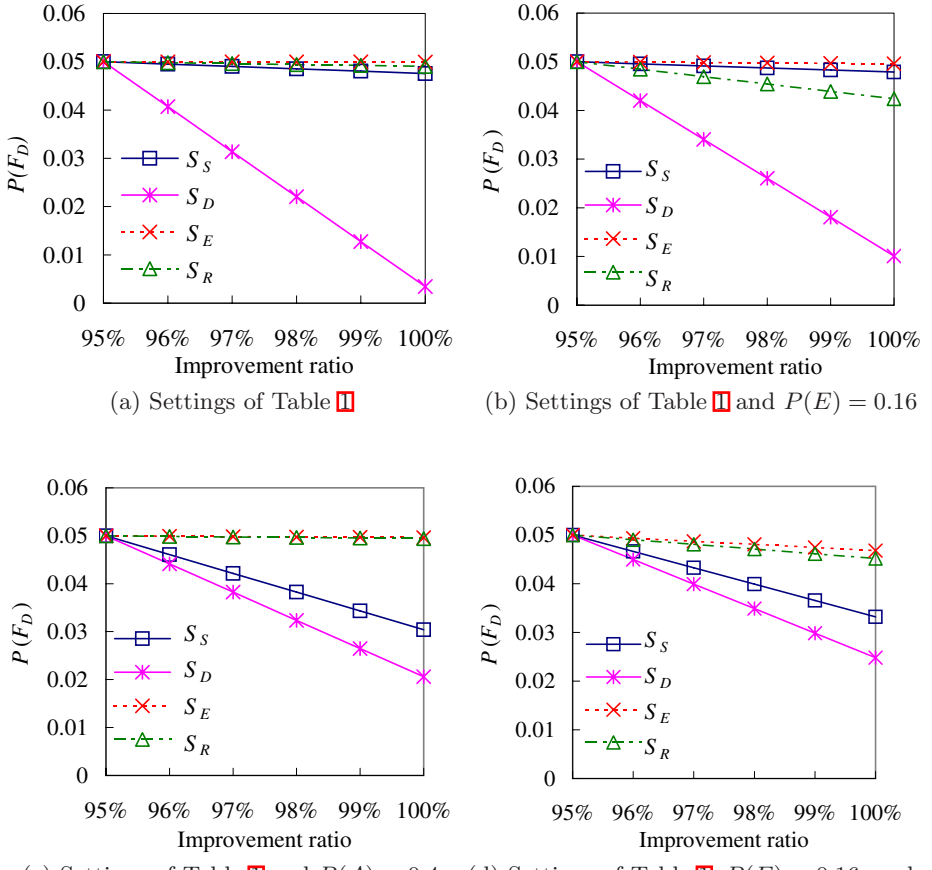
Table 1. The prior probabilities of an intrusion detection system

Event	True	False	Event	True	False	Event	True	False
A	0.050	0.950	E	0.020	0.980	M	0.050	0.950
F	0.050	0.950	H	0.019	0.981	$D\|A\bar{E}$	0.950	0.050
$D\|\bar{A}\bar{E}$	0.050	0.950	$D\|AE$	0.950	0.050	$D\|\bar{A}E$	0.050	0.950

4.2 Impact of Sensing Improvements

By the settings of Table 1, Fig. 3 shows the impact of different sensing improvements on $P(F_D)$. The slope of sensing improvements in figures means the decreasing speed of $P(F_D)$ and is determined by $P(A)$ and $P(E)$ as introduced in subsection 3.2.

In Fig. 3(a), S_S, S_E, and S_R scarcely affect $P(F_D)$, which reflects the small $P(A)$ and $P(E)$ of this example. Fig. 3(b) shows that S_R can reduce $P(F_D)$ as $P(E)$ increased. $P(E)$ can be treated as the weight of S_R for $P(F_D)$ in IDM. S_S can be used to reduce $P(F_D)$ as $P(A)$ increased, which is shown in Fig. 3(c). $P(A)$ can then be treated as the weight of S_S for $P(F_D)$ in IDM. In Fig. 3(d),



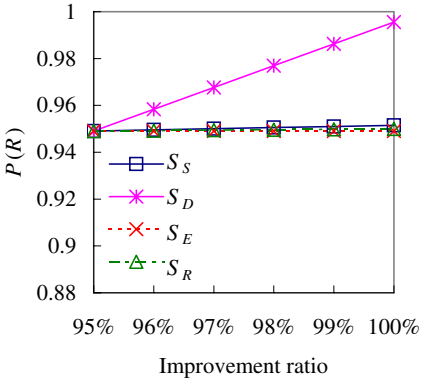
(a) Settings of Table 1

(b) Settings of Table 1 and $P(E) = 0.16$

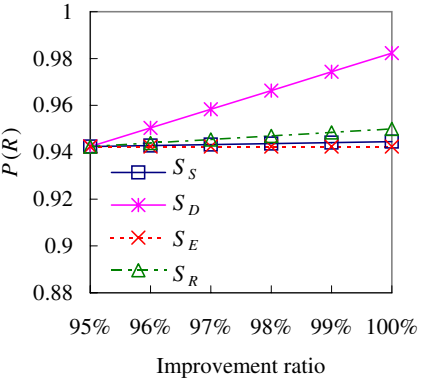
(c) Settings of Table 1 and $P(A) = 0.4$

(d) Settings of Table 1, $P(E) = 0.16$, and $P(A) = 0.4$

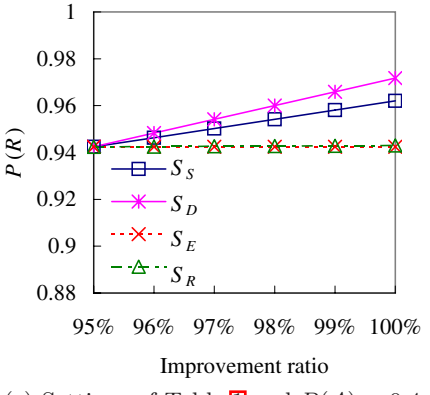
Fig. 3. Impact of sensing improvements on $P(F_D)$



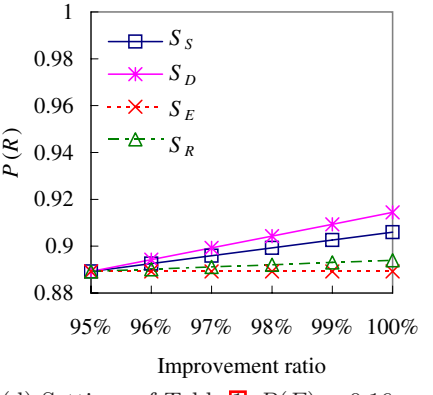
(a) Settings of Table 1



(b) Settings of Table 1 and $P(E) = 0.16$



(c) Settings of Table 1 and $P(A) = 0.4$



(d) Settings of Table 1, $P(E) = 0.16$, and $P(A) = 0.4$

Fig. 4. Impact of sensing improvements on $P(R)$

S_D and S_S can reduce $P(F_D)$ efficiently in IDM as $P(A)$ and $P(E)$ are both increased. The effect of S_R is reduced as $P(A)$ increased in Fig. 3(c) and 3(d).

Fig. 4 shows that the effect of sensing improvements on $P(R)$ based on the settings of Table 1. The results of Fig. 4(a), 4(b), and 4(c) are the same as Fig. 3 while Fig. 4(d) shows that the $P(R)$ will be significantly affected by $P(H)$ as $P(A)$ and $P(E)$ are both large. In Fig. 3 and 4, S_E is shown to be less important in sensing improvements for $P(F_D)$ and $P(R)$ in IDM as $P(A)$ and $P(E)$ are small.

The results of the impact of sensing improvements on $P(F_D)$ and $P(R)$ shown in Fig. 3 and 4, respectively, are the same as that discussed in Section 3. In addition, they can provide the quantitative and illustrative information of sensing improvements.

5 Related Work

There are many approaches proposed to reinforce the reliability of WSNs. Most of these approaches are based on the collaborative work of sensor nodes since WSNs is generally deployed densely [1], [19].

For the purpose of reliable communication, Cerpa and Estrin [3] proposed an adaptive self-configuring routing protocol, named ASCENT, to establish a routing forwarding backbone by using a subset of sensor nodes; Chang, Hsu, Liu, and Juang [4] proposed a dependable geographical routing to dodge the faulty region; Ruiz, Siqueira, Oliveira, Wong, Nogueira, and Loureiro [14] used MANNA to identify the faulty sensor nodes and proposed a management scheme for event-driven sensor networks; and Staddon, Balfanz, and Durfee [15] proposed a tracing scheme in continuous sensor networks to monitor the crashed sensor nodes.

In reliable density control, Huang, Lo, Tseng, and Chen [10] proposed several decentralized protocols that schedule the duty cycle of sensor nodes to prolong the network lifetime while the sensing field is sufficiently covered; and Ye, Zhong, Cheng, Lu, and Zhang [18] proposed an adaptive scheduling approach, named PEAS, to ensure the coverage requirement of target area is fulfilled.

The fault tolerance mechanisms are also based on the collaboration of sensor nodes with the goal of reliable computing and detecting. Sun, Chen, Han, and Gerla [16] proposed a simple distributed technique, named CWV, by using neighbor's result and exploiting redundant information to discern local data dependability for improving reliability. Krishnamachari and Iyengar [11] proposed a scheme which let an individual sensor node use binary decisions of neighbors to correct its own decision to detect the event region for increasing fault tolerant capability. Luo, Dong, and Huang [13] enhanced this work by considering both measurement error and sensor node fault, which minimized the probability of detection error by choosing a proper neighborhood size in fault correction.

The collaboration of sensor nodes may cause the consistency problem when the Byzantine faults exist. Clouqueur, Saluja, and Ramanathan [6] proposed two fusion schemes, value fusion and decision fusion, to solve the Byzantine problem [12] and to accomplish better reliability in data fusion.

6 Conclusions and Future Work

Most WSNs are coupled with inherent limitations and harsh environments, which makes them fallible. The collected data might be flawed especially under the unfavorable conditions. The reinforcement of the reliability must be seriously considered before the deployment of WSNs and during the network operation.

In this paper, we show how to estimate $P(F_D)$ and $P(R)$ in different detection models. $P(R)$, which considers the hidden fault detection, is first proposed in the detection reliability research. We also discuss and analyze the impact of sensing improvements, including S_S, S_D, S_E, and S_R on $P(F_D)$ and $P(R)$. These sensing improvements can be obtained in laboratory experiments before sensor nodes

are deployed and can be corrected in different applications by observed data as disscused in subsection 2.4 and 3.2.

The theoretical analysis illustrates the relationship among sensing improvements, $P(F_D)$, and $P(R)$. The enhancement in both of $P(F_D)$ and $P(R)$ are shown in Section 3. Further, the illustrative example of the intrusion detection system clearly shows how to control and improve $P(F_D)$ and $P(R)$ based on the different sensing improvements in different situations.

This paper shows that we can control detection reliability before the deployment of WSNs by default sensing improvements. $P(F_D)$ and $P(R)$ can then be used to compute the neighborhood size for collaborative work in the critical terrain as discussed in previous research. During the network operation, sensing improvements can be re-measured by practical data and the network protocols can be adapted by the information of $P(F_D)$ and $P(R)$.

The future work of this research will include the adaptive algorithm to rapidly adapt to the environmental interference for minimizing $P(F_D)$ and maximizing $P(R)$ during the network operation.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks 38, 393–422 (2002)
2. Biswas, P.K., Phoha, S.: Self-Organizing Sensor Networks for Integrated Target Surveillance. IEEE Trans. on Computers 55(8) (2006)
3. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. IEEE Trans. on Mobile Computing, Special Issue on Mission-Oriented Sensor Networks 3(3) (2004)
4. Chang, Y.-S., Hsu, M.-T., Liu, H.-H., Juang, T.-Y.: Dependable Geographical Routing on Wireless Sensor Networks. In: LNCS, vol. 4523 (2007)
5. Chang, Y.-S., Juang, T.-Y., Lo, C.-J., Hsu, M.-T., Huang, J.-H.: Fault estimation and fault map construction in Cluster-based Wireless. In: The IEEE International Workshop on Ad Hoc and Ubiquitous Computing (AHUC 2006), Taichung, Taiwan, June 5–7 (2006)
6. Clouqueur, T., Saluja, K.K., Ramanathan, P.: Fault Tolerance in Collaborative Sensor Networks for Target Detection. IEEE Trans. on Computers 53(3), 320–333 (2004)
7. Culler, D., Estrin, D., Srivastava, M.: Overview of Sensor Networks. IEEE Computer, Special Issue in Sensor Networks (August 2004)
8. Doolin, D.M., Sitar, N.: Wireless sensors for wildfire monitoring. In: Proceedings of SPIE Symposium on Smart Structures & Materials/ NDE 2005, March 6–10, San Diego, California (2005)
9. Hsu, M.-T., Lin, F.Y.-S., Chang, Y.-S., Juang, T.-Y.: The Fault Probability Estimation and Decision Reliability Improvement in WSNs. In: Proceedings of the IEEE Region 10 Annual International Conference (TENCON 2007), October 30–November 2 Taipei, Taiwan (2007)
10. Huang, C.-F., Lo, L.-C., Tseng, Y.-C., Chen, W.-T.: Decentralized Energy-Conserving and Coverage-Preserving Protocols for Wireless Sensor Networks. ACM Trans. on Sensor Networks 2(2) (2006)

11. Krishnamachari, B., Iyengar, S.: Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks. IEEE Trans. on Computers 53(3), 241–250 (2004)
12. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. ACM Trans. on Programming Languages and Systems 4(3), 382–401 (1982)
13. Luo, X., Dong, M., Huang, Y.: On Distributed Fault-Tolerant Detection in Wireless Sensor Networks. IEEE Trans. on Computers 55(1) (2006)
14. Ruiz, L., Siqueira, I., Oliveira, L., Wong, H., Nogueira, J., Loureiro, A.: Fault Management in Event-Driven Wireless Sensor Networks. In: ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2004 (MSWIM 2004) (2004)
15. Staddon, J., Balfanz, D., Durfee, G.: Efficient Tracing of Failed Nodes in Sensor Networks. In: First ACM International Workshop on Wireless Sensor Networks and Applications (September 002)
16. Sun, T., Chen, L.-J., Han, C.-C., Gerla, M.: Reliable Sensor Networks for Planet Exploration. In: The 2005 IEEE International Conference On Networking, Sensing and Control (ICNSC 2005) (2005)
17. Varshney, P.: Distributed Detection and Data Fusion. Springer, Heidelberg (1996)
18. Ye, F., Zhong, G., Cheng, J., Lu, S., Zhang, L.: PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks. In: Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003) (2003)
19. Zhang, H., Hou, J.C.: Maintaining Sensing Coverage and Connectivity in Large Sensor Networks. Wireless Ad Hoc and Sensor Networks: An International Journal 1(1–2), 89–123 (2005)
20. Zhang, Q., Varshney, P.K., Wesel, R.D.: Optimal Bi-Level Quantization of i.i.d. Sensor Observations for Binary Hypothesis Testing. IEEE Trans. Information Theory 48(7), 2105–2111 (2002)

Fast and Simple On-Line Sensor Fault Detection Scheme for Wireless Sensor Networks

Jeng-Yang Wu[1], Dyi-Rong Duh[1,*], Tsang-Yi Wang[2], and Li-Yuan Chang[3]

[1] Department of Computer Science and Information Engineering
National Chi Nan University, Puli, Nantou Hsieh 54561, Taiwan
{s94321522,drduh}@ncnu.edu.tw
[2] Institute of Communications Engineering
National Sun Yat-Sen University, Kaohsiung 80424, Taiwan
tcwang@mail.nsysu.edu.tw
[3] Department of Computer Science and Information Engineering
National Cheng Kung University, Tainan 70101, Taiwan
lance@csie.ncku.edu.tw

Abstract. Wireless sensor networks (WSN) are composed of a large number of sensor nodes and usually used to monitor a region of interest. The sensor nodes are very prone to damage due to low-cost design and random deployment. Additionally, faulty nodes may degrade the performance of the distributed hypothesis testing. This work addresses fault isolation in WSN where the fusion center attempts to identify faulty nodes through temporal sequences of received local decisions. Owing to the processor, memory, and power constraints in embedded systems, the employed method should be as simple as possible. Therefore, the primary goal of this investigation is to design a low-complexity sensor fault detection scheme, which can detect most sensor faults by using the majority voting technique. The simulation results show the proposed approach is effective in terms of identifying faulty members in a distributed sensor network.

Keywords: Wireless sensor networks, fusion technique, sensor fault detection, distributed detection; fault-tolerant detection; false testing.

1 Introduction

The problem of distributed decision fusion in wireless sensor networks (WSN) has received much attention because of many important applications [1, 4, 7, 8, 9, 10, 11]. Sensor nodes in WSN are deployed in the region of interest for collecting data. These sensor nodes, which consist of sensing, data processing, communicating, and power components, observe the phenomenon at each measured time step. After processing the observation, each node transmits individual local decision to a fusion center. The fusion center then makes a final decision based on these preliminary local decisions.

* Corresponding author.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 444–455, 2007.

WSN usually consist of a large number of sensor nodes, which are deployed in inaccessible and harsh environments. Furthermore, the sensor nodes are prone to damage as a result of low-cost design and random deployment. Additionally, placing sensor nodes in inaccessible areas makes them irreplaceable. Therefore, the design of distributed detection in WSN needs to be fault tolerant. The types of sensor faults in WSN may range from simple stuck-at faults to random sensor faults, which render prior failure probability models unsuitable for the design of distributed detection in WSN. For this reason, the primary goal of this study is to design an effective fault detection scheme, which can tolerate most sensor faults.

The fusion center may make a wrong decision when the combined effect of the number of faulty nodes and sensor fault types is high. To provide fault-tolerance capability in distributed detection, the detection system can remove the unreliable local decisions transmitted from faulty sensor nodes during the process of final decision making. This work considers the fault detection based on a collaboratively sequential detection scheme. The problem formulation in this study is the fusion center needs to identify faulty nodes at every time step. In the decision fusion process, the data sent form faulty nodes will be discarded for making more dependable final decisions. The considered scenario has many applications such as health monitoring and security surveillance. A deployed sensor network in each of these applications may have to report its decision at every measured time step; for this reason, an appropriate strategy can be immediately selected when an unexpected event occurs.

Some related investigations have addressed several variants of fault detection problems. Fault detection problems by central testing can be found in [2, 3]. The distributed fault detection problem for general nonlinear, non-Gaussian stochastic systems with multiple sensor nodes has been addressed [6]. The work in [9] applies the non-parametric statistics-based technique for identifying the faulty sensor nodes in a sensor network. For information assurance of the data fusion in WSN, a witness based approach has been demonstrated to verify the fusion data [8]. An improved witness-based approach using a direct voting mechanism has also been proposed to verify the fusion data without any cryptographic operation [10].

This work also considers the problem of sensor fault detection as follows. Assume that all sensors will have the same readings and make the same decision if they are fault-free and deployed in an area. The fusion center can identify a faulty node by judging whether its behavior is very different from the others since each node sends its local decision to the fusion center at every time step. Therefore, a sensor fault detection scheme with a record table, which records the history of all local decisions during the monitor process, is proposed. Because of the processor, memory, and power constraints in embedded systems, the employed approach should be as simple as possible. For this reason, the proposed scheme just applies the majority voting technique to differentiate between normal nodes and faulty nodes. Since the employed method is quite simple, applying the proposed sensor fault detection scheme in real applications is feasible.

The remainder of this investigation is organized as follows. Section 2 formally presents the system model and the problem formulation. The details of the proposed sensor fault detection scheme are described in Section 3. Section 4 shows the performance evaluation of the proposed approach by simulation. Finally, conclusions are drawn in Section 5.

2 System Model and Problem Formulation

This section first describes the system model and the problem formulation. In practice, the types of sensor faults in a sensor network are actually very diverse. Three considered sensor fault types in this study are presented finally.

2.1 Network Operation

A two-layer detection system is considered in this work, as illustrated in Fig. 1. The parallel fusion system, which consists of N identical sensor nodes and a fusion center, is used to determine whether an unknown binary hypothesis is H_0 or H_1. The prior probabilities of H_0 and H_1 are assumed to be known. Each member of N sensor nodes is denoted by s_i, where $i = 1, \ldots, N$. Let x_i^t denote the observation of the ith sensor node and u_i^t denote the binary decision of the ith node, where $i = 1, \ldots, N$ and t represents the time index. The observations across sensor nodes are independent and identically distributed condition on phenomenon.

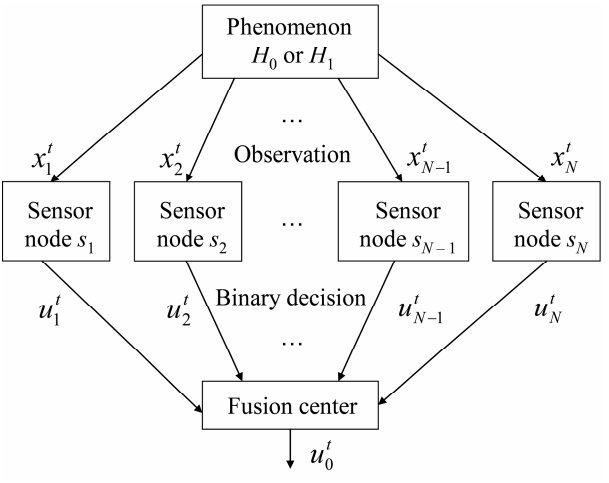


Fig. 1. System model of a parallel fusion network

Assume that an identical local decision rule is employed at each sensor node. Each node independently makes a binary decision based only on its observation. The local decision u_i^t of node s_i is obtained through the local decision rule γ as (1). Each sensor node reports its local decision to the fusion center at each time step. A decision '0' is sent if the sensor node makes a decision in favor of H_0; otherwise, a decision '1' is transmitted.

$$u_i^t = \gamma(x_i^t). \tag{1}$$

A parallel fusion network employing identical local decision rules at each node is asymptotically optimal based on error exponents when the number of nodes becomes very large. The error exponents of the identical local decision rules are equal to that of the non-identical local decision rules obtained by system-wide optimization [5]. For this reason, employing the identical local decision rule at each node is quite suitable for large-scale sensor networks.

Let's consider the fusion center is processing its information at time step t. All preliminary decisions up to time step t from all nodes are available at the fusion center. The fusion center begins to identify faulty members by utilizing the proposed sensor fault detection scheme. In the decision fusion process, the fusion center discards the data of faulty nodes for making a more believable final decision.

2.2 Sensor Fault Types

A sensor network is very likely to contain faulty nodes, because sensor nodes are usually low-cost and deployed randomly. Additionally, the sensor faults may include hardware or software damage resulting in all misbehavior; hence, the types of sensor faults are diverse.

Three types of sensor faults are considered in this work. In one fault type, a faulty sensor node is frozen to transmit a fixed local decision '0' to the fusion center regardless of the real observation. This type of sensor fault is named stuck-at-zero fault. Similarly, a fault type is called stuck-at-one fault when a faulty node always transmits a fixed decision '1'. The last sensor fault type is that a faulty sensor node reports decision '0' or '1' randomly regardless of the present hypothesis and called random fault. The fusion center does not know the sensor fault types in advance. The detection system identifies faulty nodes just according to the behavior of each node.

3 Sensor Fault Detection Scheme

The record table, which records the history of all local decisions during the monitor process, is introduced in this section. The record table will faithfully present the behavior of each node in a network. In fact, the record table plays a key role in the proposed fault detection scheme. The details of the proposed approach are described subsequently.

3.1 Record Table

This investigation considers that sensor nodes sequentially transmit local decisions to the fusion center. A sensor node can be reasonably assumed to be faulty when its behavior is very different from the majority of nodes. Recording the history of local decisions transmitted from all nodes is a method to represent the behavior of sensor nodes, since the fusion center receives each node's decision at every time step. Therefore, a record table is designed to record the behavior of each node at each time step. Let R_i^t denote the ratio of decisions '1' to all decisions which have been transmitted to the fusion center by the ith node at time step t as

$$R_i^t = \frac{1}{t}\sum_{j=1}^{t} u_i^j \tag{2}$$

As shown in (2), each R_i^t is independent of all other nodes and realistically shows the condition of local decisions transmitted by node s_i. For example, if a sensor node has ever sent three decisions '1' to the fusion center at the first five time steps, its rate of decision '1' is recorded as 3/5. For instance, a node has never transmitted decision '1' to the fusion center at the first seven time steps and then its rate of decision '1' is recorded as 0/7. The value of R_i^t is just between 0 and 1 obviously.

3.2 Proposed Scheme

The fusion center can identify faulty nodes through comparing each node's behavior. The rates of decision '1' of normal sensor nodes are similar since they have the same density function. A sensor node has the highest probability to be faulty when its rate is very different from the other nodes. For making a distinction between normal nodes and faulty nodes, the proposed scheme divides the entire rate value into q equal regions. Let p_i denote the range of each rate region as (3).

$$\text{range of } p_i = [\frac{i-1}{q}, \frac{i}{q}), \tag{3}$$

where $i = 1, \ldots, q$. Exceptionally, the 1.0 rate value is included in the last rate region.

The R_i^t of each node can be corresponded to a rate region. At each time step, the number of nodes in each rate region is initialized to 0 first. After updating the record table, the R_i^t of each node is placed to the corresponding rate region. A rate region owning the maximum number of nodes can be easily found. The fusion center then identifies faulty nodes by using the majority voting technique. However, all normal nodes do not always exactly locate in the same rate region. Several normal nodes possibly locate in the neighbor regions. For this reason, the proposed scheme marks every three continuous rate regions to form a rate group except the first group and the last one. Let g_i denote the range of each rate group as (4).

$$\begin{aligned} g_1 &= p_1 + p_2, \\ g_i &= p_{i-1} + p_i + p_{i+1}, \\ g_q &= p_{q-1} + p_q, \end{aligned} \tag{4}$$

where $i = 2, \ldots, q-1$. Rate groups are presently used to replace rate regions for lowering the probability of erroneous judgment. Similarly, the fusion center scans all rate groups for discovering a group which possesses the maximum count of nodes at each time step. A node is determined to be faulty by the fusion center if its rate does not locate in the group having the maximum number of sensor nodes.

The counts of nodes in different groups are sometimes equal. If two groups, g_a and g_b, have the same number of nodes, the fusion center will select g_a when rate region p_a has a larger count of nodes; if p_a and p_b also own the identical number of nodes, the fusion center will randomly select one rate group. Finally, the flow chart of the proposed sensor fault detection scheme at time step t is illustrated in Fig. 2.

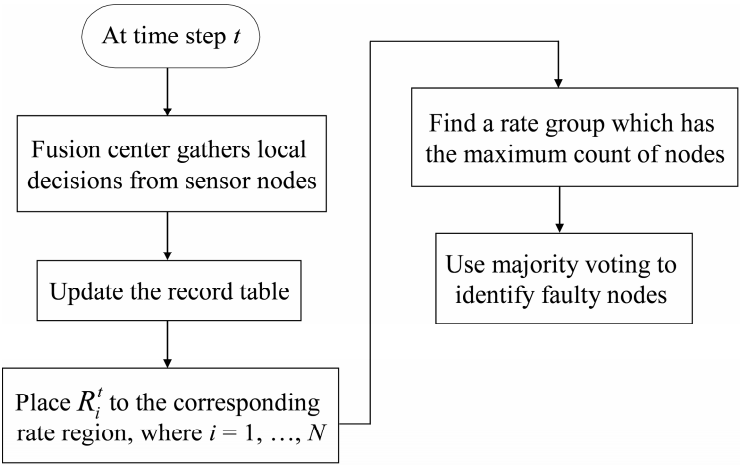


Fig. 2. Flow chart of the proposed scheme

4 Simulation Results

The error rate of fault detection in this study is described first in this section. The performance of the proposed sensor fault detection scheme is then evaluated. The sensor fault types and the actual number of faulty nodes are unknown in advance in these simulations.

4.1 Error Rate of Fault Detection

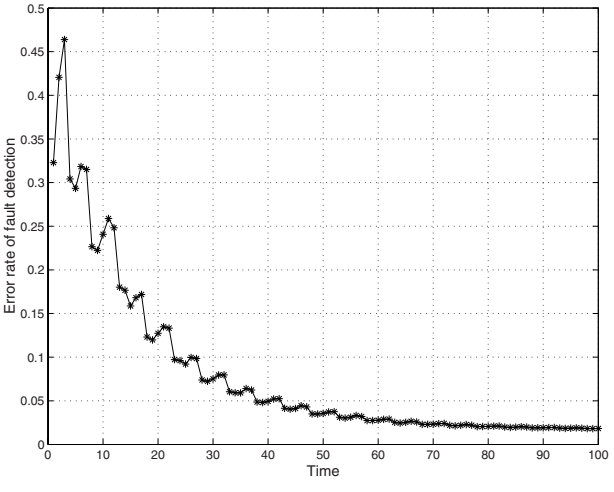
This investigation decides the error rate of fault detection through comparing the results detected by the proposed scheme with the real conditions. For instance, the proposed approach identifies two faulty sensor nodes at time step t, but all nodes are actually normal. The error rate at this time step is indicated as $2/N$. For example, there are three faulty nodes in fact, but the proposed scheme only detects two of them. The error rate in this condition is then indicated as $1/N$. Restated, the error rate of fault detection is the rate of difference between the detected result and the reality.

4.2 Simulation Setup

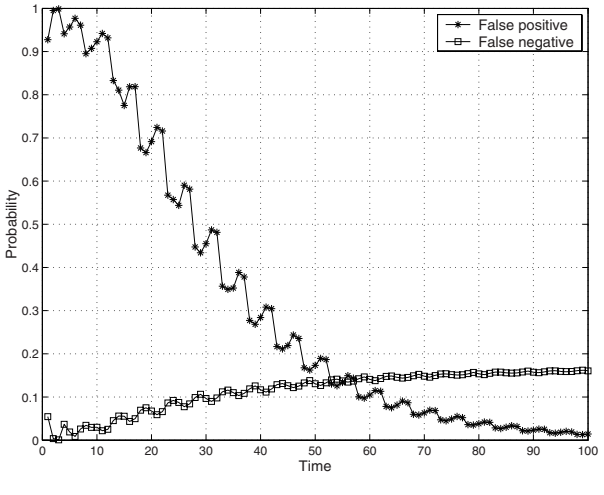
The detection of known signals in Gaussian noise is considered. This study lets the conditional densities at the sensor nodes be Gaussian with unit variance. Under H_0 and H_1, the mean at all the sensor nodes is assumed to be zero and m respectively. Accordingly, the signal-to-noise ratio (SNR) can be defined as $20 \log_{10} m$. Additionally, each sensor node makes the local decision based on the bisection threshold $m/2$. The number of deployed sensor nodes N is set to 10 in all simulated conditions. In the following simulations, the number of rate partitions q is set to 10. Each simulated scenario is iterated 10,000,000 times to obtain the simulated performance. The true hypothesis and faulty sensor nodes are randomly chosen at the beginning of every iteration step.

4.3 Results and Analysis

In practice, a sensor network probably contains several types of faults at the same time and the fault types are unknown in advance. For convenience, a scenario, which simultaneously contains three kinds of fault sensor types including stuck-at-zero fault, stuck-at-one fault, and random fault, is investigated. The fault type of individual faulty node is randomly decided at the beginning of every iteration step.

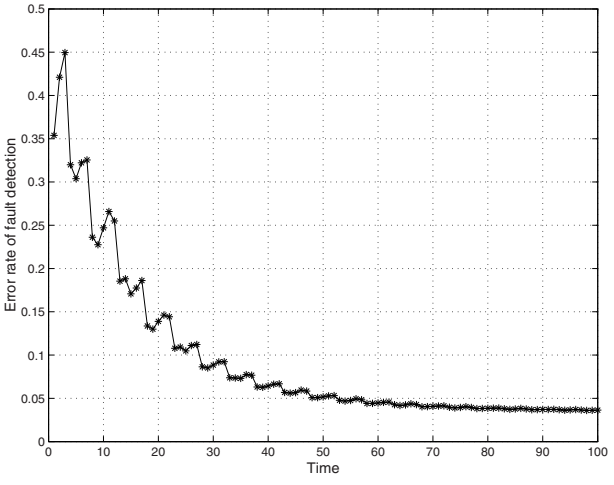


(a)

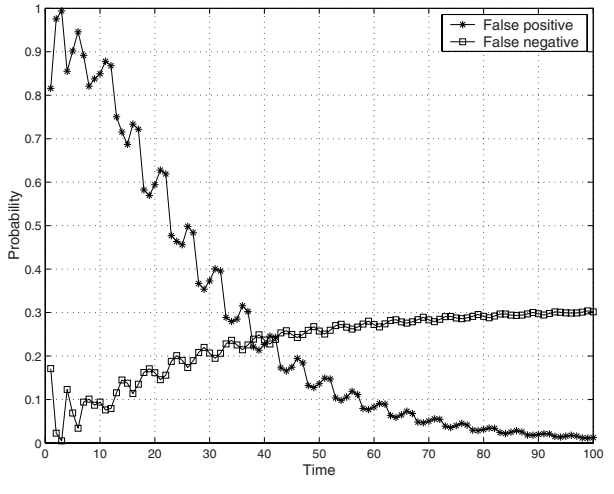


(b)

Fig. 3. (a) Error rate of fault detection and (b) probabilities of false positive and false negative in a network with one faulty node at 0 dB SNR



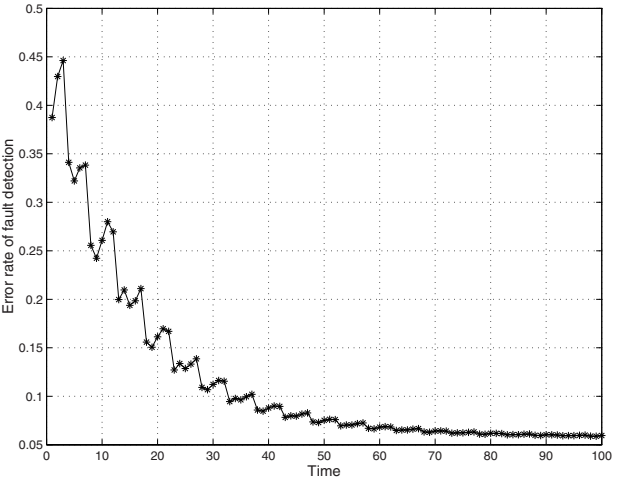
(a)



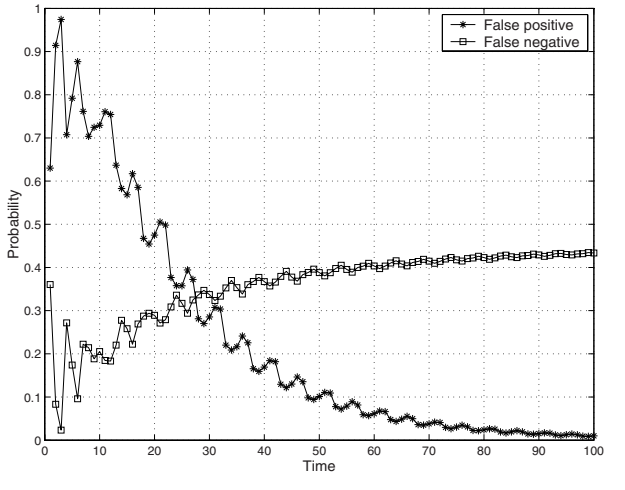
(b)

Fig. 4. (a) Error rate of fault detection and (b) probabilities of false positive and false negative in a network with two faulty nodes at 0 dB SNR

The performance of fault detection at 0 dB SNR in a situation with one faulty node is shown in Fig. 3(a). The error rate of fault detection rises in the first three time steps. The reason is no rate group can be formed in these three time steps because every possible value of rate is distinct from each other. Therefore, the fusion center identifies faulty nodes through comparing the number of nodes in each rate region. After time step three, several rate groups are formed and the probability of erroneous judgment is lowered.



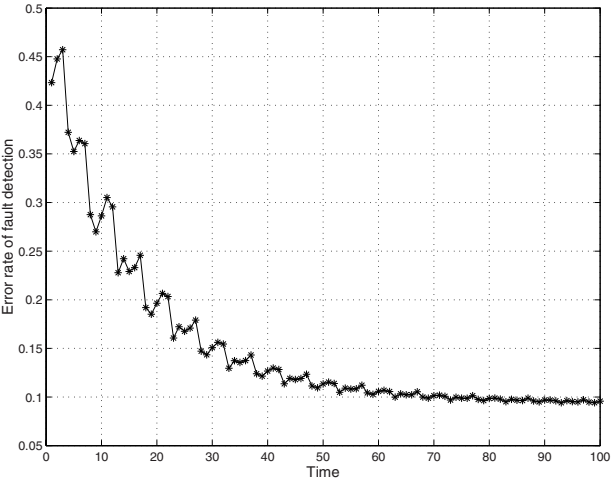
(a)



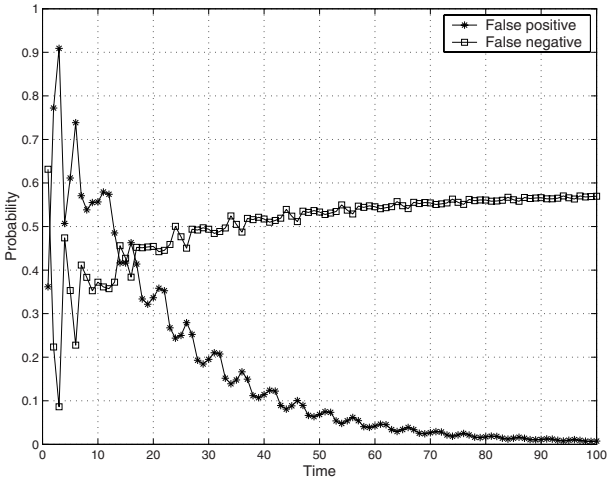
(b)

Fig. 5. (a) Error rate of fault detection and (b) probabilities of false positive and false negative in a network with three faulty nodes at 0 dB SNR

Fig. 3(b) shows the probabilities of reported false positive (fault-free node was indicated as faulty) and false negative (a node with fault was not detected) when one faulty node occur in a network. In the first several time steps, the rates of sensor nodes are unstable. Therefore, the corresponding rate regions of normal nodes are mutable. Several normal nodes are probably judged to be faulty at the moment. When time increases, the behavior of a normal node is gradually similar to other normal ones. Most normal sensor nodes will locate in a rate group which possesses the



(a)



(b)

Fig. 6. (a) Error rate of fault detection and (b) probabilities of false positive and false negative in a network with four faulty nodes at 0 dB SNR

maximum count of nodes. Therefore, the probability of false positive will gradually lower. On the other hand, the probability of false negative rises. The reason is that the behavior of random fault is not significantly different from that of a normal node in fact. When the rates of all nodes are more and more stable, random faults might be contained by the rate group having maximum number of nodes. Consequently, the probability of false negative rises owing to the erroneous judgment.

Figs. 4, 5 and 6 present the same evaluation information for the cases where two, three and four deployed sensor nodes were faulty, respectively. The descending error rate of fault detection shows the major part of faulty nodes can be identified by the fusion center apparently. For this reason, the proposed approach has the capability to assist the distributed detection system in making more dependable decisions by isolating most sensor faults. The reason for the trends of false positive and false negative in Fig. 6 is similar to the aforementioned statement. The rates of normal nodes become more and more stable and similar when time increases. The probability of false positive will gradually decline. Contrarily, the probability of failing in detecting random faults increases when the number of faulty nodes is large.

5 Conclusions

Faulty sensor nodes in WSN always report unreliable information. For this reason, the fusion center may make wrong decisions according to inaccurate local decisions. This study investigates the problem of detecting faulty sensor nodes in distributed sensor networks at every time step.

A sensor node whose behavior is very unusual may be faulty. In order to show the behavior of each node in a network, this work designs a record table for recording the history of all local decisions during the monitor process. Additionally, designing a simple fault detection scheme for WSN is necessary owing to the computing capability constraint. For this reason, this investigation applies the majority voting technique to identify faulty members. The simulation results show that the proposed scheme with a record table is effective in terms of fault detection. Most importantly, the proposed sensor fault detection scheme does not need complex operations. Therefore, the precious energy resource in WSN could be conserved. The number of faulty nodes in this investigation is fixed during the whole monitor process. This work will be continuously improved for dealing with the increasable number of faulty nodes in a network as the further work.

References

1. Aldosari, S.A., Moura, J.M.F.: Detection in decentralized Sensor Networks. In: Proc. of the 2004 IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing, vol. 2, pp. 277–280 (2004)
2. Basseville, M.: Detecting Changes in Signals and Systems-A Survey. Automatica 24(3), 309–326 (1988)
3. Basseville, M., Nikiforov, I.: Detection of Abrupt Changes-Theory and Applications. Prentice-Hall, Englewood Cliffs, NJ (1993)
4. Chamberland, J.F., Veeravalli, V.V.P.: Asymptotic Results for Decentralized Detection in Power Constrained Wireless Sensor Networks. IEEE Journal of Selected Areas in Communications 22(6), 1007–1015 (2004)
5. Chen, P.N., Papamarcou, A.: New Asymtotic Results in Parallel Distributed Detection. IEEE Transactions on Information Theory 39(6), 1847–1863 (1993)

6. Cheng, Q., Varshney, P.K., Michels, J., Belcastro, C.M.: Distributed Fault Detection via Particle Filtering and Decision Fusion. In: Proc. of the 8th Int'l. Conf. on Information Fusion, vol. 2, pp. 1239–1246 (2005)
7. D'Costa, A., Sayeed, A.M.: Data versus Decision Fusion for Distributed Classification in Sensor Networks. In: Proc. of the 2003 IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing, vol. 1, pp. 585–590 (2003)
8. Du, W., Deng, J., Han, Y.S., Varshney, P.K.: A Witness-Based Approach for Data Fusion Assurance in Wireless Sensor Networks. In: Proc. of GLOBECOM 2003, vol. 3, pp. 1435–1439 (2003)
9. Koushanfar, F., Potkonjak, M., Sangiovanni-Vincentelli, A.: On-Line Fault Detection of Sensor Measurement. In: Proc. of IEEE Sensors, vol. 2, pp. 974–979 (2003)
10. Pai, H.T., Han, Y.S.: Power-Efficient Direct-Voting Assurance for Data Fusion in Wireless Sensor Networks. IEEE Trans. on Computers (accepted)
11. Varshney, P.K.: Distributed Detection and Data Fusion. Springer, New York (1997)

An Activity-Centered Wearable Computing Infrastructure for Intelligent Environment Applications

Dipak Surie and Thomas Pederson

Department of Computing Science, Umeå University,
S-901 87 Umeå, Sweden
{dipak,top}@cs.umu.se

Abstract. There are many research efforts that focus on converting everyday environments into intelligent and computationally active environments that support and enhance the abilities of its occupants in executing their activities. Such environments must have the ability to recognize the activities performed by its occupant, maintain a real-time model of the environment, address the occupant's privacy and personalization issues, and provide interaction capabilities in a way the occupant would with other people. In this paper we present an activity-centered wearable computing infrastructure for designing intelligent environment applications based on the occupant's usage and storage of everyday objects in that environment. Four components namely object manager, situative space monitor, activity recognizer and egocentric interaction manager are introduced and described in detail. A prototypical intelligent home environment capable of supporting 15 Activities of Daily Living with an activity recognition precision of 92% is presented as a "proof of concept" in a virtual reality (VR) simulated home environment.

1 Introduction

There are many research efforts that focus on converting everyday environments into intelligent and computationally active environments that support and enhance the abilities of its occupants in executing their activities [1], [2], [3]. Such environments must have the ability to detect its current state, the current state of its occupant and the activity performed by the occupant for providing computational support that are typically different to the office activities currently supported by desktop computing. Examples of applications that are developed for intelligent environments include providing support for activities of daily living at homes [4], for surgical activities in operation theatres, for mechanical activities in car workshops, etc. A major problem in building such applications has been the lack of uniform support for modelling and detecting: 1) the occupant's activities and actions; 2) the occupant's interactive status in terms of situative and multi-modal access to input and output devices; and 3) the occupant's environmental status in terms of available resources and their states. Many efforts in developing intelligent environments are heavily influenced by the underlying technology resulting in a lack of generality in developing such environments. We believe that starting out from a perspective centred around how occupants literally

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 456–465, 2007.

perceive the environment, based on the occupant's usage and storage of everyday objects in that environment and based on the weight that current cognitive science give to activity-driving factors could offer a valuable complement. In particular, it could offer a conceptual design platform robust enough to survive and handle generations of changes in the field of sensor technology.

Intelligent environments are public and private spaces that allow computers to participate in everyday activities that have previously never involved computation and allow occupants to interact with computers in a way they would with other people: via gesture, voice, context, etc. [5]. We follow similar design goals in developing an activity-centered wearable computing infrastructure for an intelligent home environment application capable of providing support for patients suffering from mild-dementia in performing their Activities of Daily Living (ADL) [4]. Four components namely *object manager*, *situative space monitor*, *activity recognizer* and *egocentric interaction manager* are introduced and described in detail.

2 Challenges Involved in Designing Intelligent Environments

Designing intelligent environments involves many challenges of which we address the following four fundamental challenges.

2.1 Activity Recognition

Intelligent environments that provide support for the occupant's activities should understand the concept of an activity and provide provisions for handling it [6]. Activity recognition in everyday environments is difficult due to the number, variety and variations in activities performed by the occupants[1]. For many intelligent environment applications, knowing the inhabitant's activity information alone is not sufficient [4]. This introduces a necessity to recognize activities and actions related to those activities with fine granularity; to recognize those activities during the initial phase of an activity before the activity reaches an irreversible state; and to determine the end of an activity [7], [8]. According to Activity Theory [9], human activities have an objective and are mediated through tools. We consider the objects present in the occupant's environment as tools for accomplishing his/her activities and recognize the activities based on the occupant's usage of those objects. For more information about the two activity recognition systems that we have built based on the occupant's usage of everyday objects, we refer to [7], [8].

2.2 Situative and Multi-modal Interaction Design

Intelligent environments that provide support to the occupant's activities should consider the occupant's current interactive status before requesting for explicit input or for providing explicit output. Two issues that need to be addressed are [10]: 1) to select the appropriate moment to grab the occupant's attention and interact with him/her since the occupant does not explicitly dedicate all his/her attention, and 2) to

[1] We are interested in activities performed by an individual instead of group activities performed by many occupants in an intelligent environment.

select the appropriate interactive device to interact with the occupant considering the number and variety of interactive devices present in such environments. The first issue is addressed by considering the action the occupant is currently performing within an activity and based on the status of an activity (*initiated*, *interrupted* or *completed*). For more information about how activities are modelled, we refer to [8]. The second issue is addressed by considering interactive devices around the occupant's body and based on the occupant's perceptual capabilities (explicit output devices within the *observable space* and explicit input devices within the *manipulable space*). For more information about observable space and manipulable space which is part of the egocentric interaction model, we refer to [10].

2.3 Maintaining a Model of the Environment

Intelligent environments should maintain a real-time model of its environment in a generalized manner required by all its applications. However there is no common way to acquire and maintain a model of the environment. This is a problem addressed within the research area of context-aware computing [11], [12], and [13]. We do not maintain a model of the environment based on absolute location; instead we model the environment in a symbolic manner in terms of the objects that are present in the environment and their relationship to other objects (*containment property* discussed in [14]) [15]. An important issue that needs to be addressed is to keep track of the objects that enter or leave the intelligent environment in an ad-hoc manner. We classify objects present in an intelligent environment into four types: 1) *simple objects* - that do not change their internal state like *knife*, *fork*, *coffee cup*, etc; 2) *complex objects* - that do change their internal state like *microwave oven*, *stove*, *oven*, *tap*, etc; 3) *container objects* - that contain other objects like *fridge*, *freezer*, *cupboard*, *dining table*, etc; and 4) interactive devices - that are used for obtaining explicit input and for providing explicit output like *wall mounted display*, *wrist-worn display, audio headset, speakers*, etc. By considering a generalized approach to maintaining a model of the environment, the intelligent environment applications can directly make use of the model without knowing the details of how such information is sensed.

2.4 Privacy and Personalization

Intelligent environments should be designed to control the flow of information about its occupants and to personalize the environment based on the occupant's presence, behavior and intentions [16], [17]. However, maintaining a complex model of each person within an intelligent environment is a difficult task to handle, since such models get outdated with time and needs to be updated everytime the occupant enters an intelligent environment. Considering the issues of privacy and personalization, we propose a wearable computing infrastructure for developing and executing intelligent environment applications [10]. However, we are aware of the limitations of a purely wearable computing infrastructure in managing environmental resources among several occupants and in controlling the localized resources in the environment [18]. We intend to address such issues in the future.

3 An Activity-Centered Wearable Computing Infrastructure

Based on the four fundamental challenges described earlier, we present an activity-centered wearable computing infrastructure for developing and executing intelligent environment applications. Refer to Figure 1.

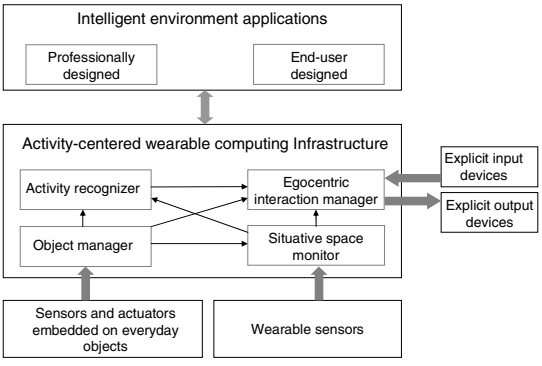


Fig. 1. An activity-centered wearable computing infrastructure for intelligent environment applications, adapted from [10]

Sensors embedded on everyday objects[2]. All everyday objects that are present in the occupant's environment are embedded with passive RFID tags [21]. This includes simple objects, complex objects, container objects and interactive devices. Simple, but multiple state change sensors like on-off switches, temperature sensors, rotation sensors, pressure sensors, etc. are embedded on complex everyday objects to sense their internal states and state changes. RFID readers are embedded on container objects to sense the objects present within the containers and when an object enters or leaves a container [8]. The instrumented sensors communicate with the object manager using ZigBee communication protocol [22].

Wearable sensors. A wearable RFID reader is worn on the inhabitant's chest with three antennas. Two antennas are worn on either wrist to sense the objects grabbed/released events while the third antenna is worn on the chest to sense the objects within the inhabitant's manipulable space [21]. Objects present within the inhabitant's observable space is currently not considered for sensing using real world sensors. However, observable space information channel is used in our VR simulation [7].

Object manager maintains a real-time model of the environment by capturing, storing and managing information about the objects present in the occupant's environment. This component is responsible for the activation or deactivation of the objects that enter or leave the intelligent environment respectively. It stores the activated objects' identity, type, and possible internal states. An object's current

[2] At present, actuators are not embedded on everyday objects and we have left it for future work.

internal state and current external state in terms of its container object can be queried. For container objects, it stores the identities of the set of objects the container contains. It also stores the event that is generated when an object enters or leaves a container object [8].

- *<Activate, {objectID, Type, "simple", "complex", "container", "interactive", set_of_internal_states}>*
- *<Deactivate, objectID>*
- *<objectID, {current_internal_state}>*
- *<objectID, {containerID}>*[3]
- *<containerID, {set_of_objectIDs}>*
- *<objectID, containerID, {has_entered, has_left}>*

Situative space monitor maintains a real-time model of the environment by capturing and storing information about the objects present in the environment from the occupant's egocentric perspective based on what the occupant can see and not see, touch and not touch at any given moment in time [7], [10]. This component stores the set of objects in the occupant's hands, the set of objects within the occupant's manipulable space and the set of objects within the occupant's observable space. It can be queried to check if an object is in the occupant's hands or within the occupant's manipulable space or within the occupant's observable space. It also stores the event that is generated when an object is grabbed or released by the occupant [7].

- *<OM, {set_of_objects}>*
- *<MS, {set_of_objects}>*
- *<OS, {set_of_objects}>*
- *<objectID, OM, {is_OM}>*
- *<objectID, MS, {within_MS}>*
- *<objectID, OS, {within_OS}>*
- *<objectID, {is_grabbed, is_released}>*

Egocentric interaction manager is responsible for the situative and multi-modal selection of input and output devices considering the occupant's situation, current activity, personal preferences, and based on the application. It obtains explicit input for the applications and presents explicit output from the applications to the occupant by communicating with the interactive devices that are either worn (using Bluetooth communication) or present in the occupant's environment (using WLAN) [10]. This component is responsible for the activation or deactivation of the devices that are available or not available for interaction respectively. It stores the activated devices' identity, type, access level and modality. It can be queried to obtain the set of available input devices, the set of available input devices within the occupant's manipulable space, the set of available output devices, and the set of available output devices within the occupant's observable space.

- *<Activate, {deviceID, Type, "input", "output", AccessLevel, "private", "public", Modality, "voice", "visual", "gesture", "tactile"}>*[4]
- *<Deactivate, deviceID>*

[3] *containerID* is valid only for container objects and is the same as its *objectID*.
[4] *deviceID* is valid only for interactive devices and is the same as its *objectID*.

- *<InputDevices, {set_of_available_input_devices}>*
- *<InputDevices, MS, {set_of_input_devices_within_MS}>*
- *<OutputDevices, {set_of_available_output_devices}>*
- *<OutputDevices, OS, {set_of_output_devices_within_OS}>*

Activity recognizer is responsible for modeling, recognizing and storing the occupant's activities and actions performed within the intelligent environment. For more information about activity recognition based on object manipulation, manipulable space and observable space information channels, we refer to [7]. For more information about activity recognition based on intra manipulation and extra manipulation information channels, we refer to [8]. This component is responsible for the activation or deactivation of the activities that are included or excluded for recognition respectively. It can be queried for current activity (recognition probability included), current action (recognition probability included), an activity's current status and also an action's current status.

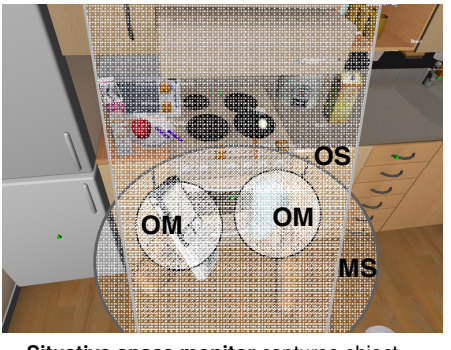
- *<Activate, {activityID, set_of_actionIDs, set_of_mandatory_events}>*
- *<Deactivate, activityID>*
- *<CurrentActivity, {activityID, activityPR, set_of_actionIDs, set_of_mandatory_events, Status, "initiated", "interrupted", "completed"}>*
- *<CurrentAction, {actionID, actionPR, activityID}>*
- *<activityID, {activityPR, set_of_actionIDs, set_of_mandatory_events, Status, "initiated", "interrupted", "completed"}>*
- *<actionID, {actionPR, activityID}>*

The object manager, situative space monitor, activity recognizer, egocentric interaction manager and the intelligent environment applications write data into a common black board and exchanges information among themselves using Information and Content Exchange (ICE) Protocol.

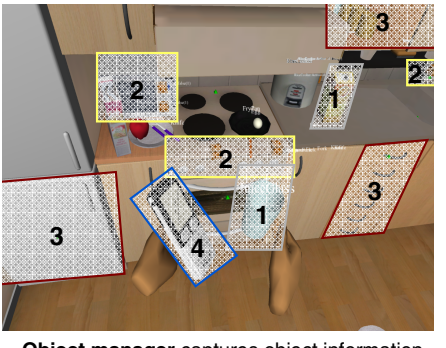
4 Experimental Set-Up and Evaluation

VR was used as a test-bed [7] in order to speed up the design process of our activity-centered wearable computing infrastructure and to allow us to compensate for the limitations with the currently available sensing technologies (especially for sensing the objects present within the container objects and for sensing the occupant's observable space). A VR model, developed using the Colosseum3D real-time physics platform [19] is used to simulate a physical home environment with wearable sensors and sensors embedded on selected everyday objects (discussed in section 3). Fig. 2 shows a snapshot of our VR simulated intelligent home environment. We have experimented with 78 different objects (128 objects in total) which include 56 different simple objects, 7 different complex objects, 11 different container objects and 4 different interactive devices (*wall-mounted display*, *wrist-worn display, audio headset,* and *environmental speakers*). We represent objects by their object identity (*objectID*) and not by their individual identity. For instance, both *fork_1* and *fork_2* are represented as *fork*. Hence all *fork*s have the same *objectID*.

The experiments were performed by 5 subjects (none of them are affiliated to the system development team)[5] in a virtual reality simulated intelligent home environment[6]. 15 activities of daily living were included based on the AMPS framework [20]. The activities were performed 10 times as part of various scenarios. A scenario comprises of a few related activities performed in some sequence. We have used 7 scenarios with some activities common for several scenarios like for instance, the activity of *preparing coffee* which is common to both the *lunch scenario* and the *coffee-break scenario*. All the subjects were allowed to perform the activities in their own way (often in many different ways) within the intelligent home. We had conducted ethnographical studies in 5 households to compare how ADLs are performed in the "real-world" and in the "VR-world". For more information, we refer to [23].



Situative space monitor captures object information within the occupant's **OS** (observable space), **MS** (manipulable space) and **OM** (object manipulation)



Object manager captures object information available in the occupant's environment: **1)** simple object, **2)** complex object, **3)** container object, and **4)** interactive device

Fig. 2. VR simulated intelligent home environment

ADL support for mild-dementia patients. The proposed activity-centered wearable computing infrastructure was evaluated by building a professionally designed ADL support application. To learn about our user group, we have conducted over 10 brainstorming sessions and 4 interview sessions with 6 occupational therapists (from HjälpmedelsCentrum Norr and Geriatric Medicine Department at the University Hospital of Northern Norrland in Sweden).

The data generated based on the activities performed by the 5 subjects were used to train 5 different wearable computing systems, each one personalized for the particular subject. By applying leave-one-out cross-validation (LOOCV) scheme for each subject without mixing the data, we obtain a recognition precision of 92% at the

[5] Dementia patients were not used as subjects in the VR environment. Based on our initial results, we are currently working on a hardware prototype which will be evaluated by the mild-dementia patients.

[6] The subjects were initially taught how to perform activities in a virtual reality environment and then given a time period to practice in this environment. Only when the subjects were comfortable with the environment, they were allowed to perform the activities.

activity-level (recall is 99%) and 81% at the action-level (recall is 94%) among 15 ADLs. For more information, refer to [8]. However by mixing the data of all the 5 subjects and applying the LOOCV scheme, we obtain a low recognition precision of 59% at the activity-level (recall is 92%) among the 15 ADLs. One important requirement for our application is that the infrastructure should recognize activities and action with high precision and recall values. Hence by considering a personalized approach for modelling and recognizing the occupant's activities, we were able to obtain high precision and recall values.

The support application operates in the occupant's background, constantly keeping track of the activities performed by the occupant. For all the activities initiated by the occupant, the application checks if the occupant has completed the following: 1) Mandatory Events – examples include *forgetting to turn off the stove* after preparing vegetables, *forgetting to place the knife and the fork on the dining table* during the activity of preparing the table for lunch, etc., 2) Mandatory Actions – examples include *forgetting to clean-up the dining table* after having lunch, etc. If the occupant forgets to complete the mandatory events and actions as part of an activity (determined using a sliding window of 2 actions "before" and 2 actions "after" from the system detected current activity state), the application intervenes and attempts to establish communication with the occupant through the egocentric interaction manager.

The egocentric interaction manager was qualitatively evaluated by the 5 subjects. The egocentric interaction manager attempted to grab the occupant's attention by playing a beep sound on his/her Bluetooth headset. Incase the Bluetooth headset is currently not available the system plays the sound in the environmental speakers (if the *AccessLevel* is *"public"*). Since mild-dementia patients comprehend better when the events and actions are represented by pictures with less information, we have used pictures with some bright colours that inform about the missing events and actions. The egocentric interaction manager attempts to present this information on a wall-mounted screen present in the occupant's kitchen if the occupant is present in the kitchen environment and if the screen is within or near to his/her observable space. Otherwise this information is presented on the occupant's wrist-worn display. We have also experimented with Voice User Interface (VUI) designed using Dragon Naturally Speaking voice recognition software for obtaining explicit input using voice. Mobile phone keys were used as a complementary option to VUI for obtaining explicit input from the occupant. Even though the egocentric interaction manager at present is based on simple rules, the subjects were positive on the idea of presenting information on different devices based on the occupant's situation. 2 subjects were not completely positive with the idea of having to always wear the Bluetooth headset and the wrist-worn display for explicitly interacting with the system. In the future, we intend to import more intelligence into the egocentric interaction manager and also experiment with other multimodal interaction approaches that are based on gesture and tactility.

5 Conclusions

In this paper we have presented an activity-centered wearable computing infrastructure for developing intelligent environment applications. Four fundamental challenges were outlined for designing intelligent environments and the components

developed to address such challenges were described and evaluated in a VR simulated home environment. By considering a personalized approach, the activity recognizer has shown as recognition precision of 92% at the activity-level and 81% at the action-level among 15 ADLs. The egocentric interaction manager has shown initial promise with the idea of interacting with the occupant based on his/her situation within an intelligent environment. An ADL support application capable of providing support for missing mandatory events and actions within activities was developed based on the infrastructure proposed for intelligent environments.

Acknowledgements

We would like to thank Anders Backman, Björn Sondell, Daniel Sjölie, Fabien Lagriffoul, Gösta Bucht, Kenneth Bodin, Lars-Erik Janlert, and Marcus Maxhall from Umeå University, Sweden. This work is partially funded by the EC Target 1 structural fund program for Northern Norrland, Sweden.

References

1. Brumitt, B., Meyers, B., Krumm, J., Kern, A., Shafer, S.: EasyLiving: Technologies for Intelligent Environments. In: Proceedings of the Intl. Conf. on Handheld and Ubiquitous Computing, pp. 12–27 (2000)
2. Kidd, C.D., Orr, R.J., Abowd, G.D., Atkeson, C.G., Essa, I.A., MacIntyre, B., Mynatt, E., Starner, T.E., Newstetter, W.: The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In: Streitz, N.A., Hartkopf, V. (eds.) CoBuild 1999. LNCS, vol. 1670, pp. 191–198. Springer, Heidelberg (1999)
3. Mozer, M.C.: The Neural Network House: An Environment that Adapts to its Inhabitants. In: Coen, M. (ed.) Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments, Menlo Park, CA, pp. 110–114. AAAI Press, Stanford (1998)
4. Backman, A., Bodin, K., Bucht, G., Janlert, L.-E., Maxhall, M., Pederson, T., Sjölie, D., Sondell, B., Surie, D.: easyADL – Wearable Support System for Independent Life despite Dementia. In: Workshop on Designing Technology for People with Cognitive Impairments, CHI2006, pp. 22–23 (April 2006)
5. Coen, M.: Design Principles for Intelligent Environments. In: Proceedings of the 15th national/10th conference on Artificial intelligence/Innovative applications of artificial intelligence (March 23-25, 1998) ISBN:0-262-51098-7
6. Christensen, H.B., Bardram, J.: Supporting Human Activities - Exploring Activity-Centered Computing. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, pp. 107–116. Springer, Heidelberg (2002)
7. Surie, D., Pederson, T., Lagriffoul, F., Janlert, L.-E., Sjölie, D.: Activity Recognition using an Egocentric Perspective of Everyday Objects. In: UIC 2007. Proceedings of IFIP 2007 International Conference on Ubiquitous Intelligence and Computing. LNCS, vol. 4611, pp. 246–257. Springer, Heidelberg (2007)
8. Surie, D., Lagriffoul, F., Pederson, T., Sjölie, D.: Activity Recognition based on Intra and Extra Manipulation of Everyday Objects. In: UCS2007. Proceedings of the 4th International Symposium on Ubiquitous Computing Systems. LNCS, Springer, Heidelberg (to appear, 2007)

9. Nardi, B. (ed.): Context and Consciousness: Activity Theory and Human-Computer Interaction. MIT Press, Cambridge (1995)
10. Pederson, T., Surie, D.: Towards an Activity-Aware Wearable Computing Platform based on an Egocentric Interaction Model. In: UCS 2007. Proceedings of the 4th International Symposium on Ubiquitous Computing Systems. LNCS, Springer, Heidelberg (to appear, 2007)
11. Dey, A.: Providing Architectural Support for Building Context-Aware Applications, PhD thesis, College of Computing, Georgia Institute of Technology (December 2000)
12. Dey, A.: Understanding and Using Context. In: Personal and Ubiquitous Computing, vol. 5(1), pp. 4–7. Springer, Heidelberg (2001) ISSN:1617-4909
13. Schmidt, A.: Ubiquitous Computing – Computing in Context, Ph.D. thesis in computer science, Lancaster University (November 2002)
14. Pederson, T.: From Conceptual Links to Causal Relations — Physical-Virtual Artefacts in Mixed-Reality Space. PhD thesis, Dept. of Computing Science, Umeå university, report UMINF-03.14, ISBN 91-7305-556-5 (2003)
15. Hightower, J., Borriello, G.: A Survey and Taxonomy of Location Sensing Systems for Ubiquitous Computing. In: UW CSE 01-08-03, University of Washington, Seattle, WA (August 2001)
16. Langheinrich, M.: Privacy Invasions in Ubiquitous Computing. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, Springer, Heidelberg (2002)
17. Chatfield, C., Carmichael, D., Hexel, R., Kay, J., Kummerfeld, B.: Personalisation in Intelligent Environments: Managing the Information Flow. In: Proceedings of the 19th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: citizens online: considerations for today and the future. ACM International Conference Proceeding Series, vol. 122, pp. 1–10 (2005)
18. Rhodes, B., Minar, N., Weaver, J.: Wearable Computing Meets Ubiquitous Computing. The Proceedings of The Third International Symposium on Wearable Computers (ISWC 1999), San Francisco, CA, 141-149 (18-19 October,1999)
19. Backman, A.: Colosseum3D – Authoring Framework for Virtual Environments. In: Proceedings of EUROGRAPHICS Workshop IPT & EGVE Workshop, pp. 225–226 (2005)
20. AMPS. as on 4nd (September 2007), http://www.ampsintl.com/
21. Finkenzeller, K.: RFID Handbook, 2nd edn. John Wiley and Sons, Chichester (2003)
22. Gutierrez, J.A., Callaway, E.H., Barrett, R.: IEEE: 802.15.4 Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensor Networks, IEEE ISBN-10: 0738135577, ISBN-13: 978-0738135571 (November 30, 2003)
23. Bhatt, R.: Comparing the Performance of ADLs in Virtual and Real Life Environments. Dept. of Computing Science, Umeå University, report UMINF-06.40 (2006)

Finding and Extracting Data Records from Web Pages*

Manuel Álvarez, Alberto Pan**, Juan Raposo, Fernando Bellas, and Fidel Cacheda

Department of Information and Communications Technologies
University of A Coruña, Campus de Elviña s/n. 15071. A Coruña, Spain
{mad,apan,jrs,fbellas,fidel}@udc.es

Abstract. Many HTML pages are generated by software programs by querying some underlying databases and then filling in a template with the data. In these situations the metainformation about the data structure is lost, so automated software programs cannot process these data in such powerful manners as information from databases. We propose a set of novel techniques for detecting structured records in a web page and extracting the data values that constitute them. Our method needs only an input page. It starts by identifying the data region of interest in the page. Then it is partitioned into records by using a clustering method that groups similar subtrees in the DOM tree of the page. Finally, the attributes of the data records are extracted by using a method based on multiple string alignment. We have tested our techniques with a high number of real web sources, obtaining high precision and recall values.

1 Introduction

In today's Web, there are many sites providing access to structured data contained in an underlying database. Typically, these sources, known as "semi-structured" web sources, provide some kind of HTML form that allows issuing queries against the database, and they return the query results embedded in HTML pages conforming to a certain fixed template. For instance, Fig. 1 shows a page containing a list of data records, representing the information about books in an Internet shop.

Allowing software programs to access these structured data is useful for a variety of purposes. For instance, it allows data integration applications to access web information in a manner similar to a database. It also allows information gathering applications to store the retrieved information maintaining its structure and, therefore, allowing more sophisticated processing.

Several approaches have been reported in the literature for building and maintaining "wrappers" for semi-structured web sources ([2][9][11][12][13]; [7] provides a brief survey). Although wrappers have been successfully used for many web data extraction and automation tasks, this approach has the inherent limitation that the target data sources must be known in advance. This is not possible in all cases.

* This research was partially supported by the Spanish Ministry of Education and Science under project TSI2005-07730.

** Alberto Pan's work was partially supported by the "Ramón y Cajal" programme of the Spanish Ministry of Education and Science.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 466–478, 2007.

Consider, for instance, the case of "focused crawling" applications [3], which automatically crawl the web looking for topic-specific information.

Several automatic methods for web data extraction have been also proposed in the literature [1][4][5][14], but they present several limitations. First, [1][5] require multiple pages generated using the same template as input. This can be inconvenient because a sufficient number of pages need to be collected. Second, the proposed methods make some assumptions about the pages containing structured data which do not always hold. For instance, [14] assumes the visual space between two data records in a page is always greater than any gap inside a data record (we will provide more detail about these issues in the related work section).

In this paper, we present a new method to automatically detecting a list of structured records in a web page and extract the data values that constitute them. Our method requires only one page containing a list of data records as input. In addition, it can deal with pages that do not verify the assumptions required by other previous approaches. We have also validated our method in a high number of real websites, obtaining very good effectiveness.

The rest of the paper is organized as follows. Section 2 describes some basic observations and properties our approach relies on. Sections 3-5 describe the proposed techniques and constitute the core of the paper. Section 3 describes the method to detect the data region in the page containing the target list of records. Section 4 explains how we segment the data region into data records. Section 5 describes how we extract the values of each individual attribute from the data records. Section 6 describes our experiments with real web pages. Section 7 discusses related work.

Fig. 1. Example HTML page containing a list of data records

2 Basic Observations and Properties

We are interested in detecting and extracting lists of structured data records embedded in HTML pages. We assume the pages containing such lists are generated according to the page creation model described in [1]. This model formally captures the basic observations that the data records in a list are shown contiguously in the page and are formatted in a consistent manner: that is, the occurrences of each attribute in several records are formatted in the same way and they always occur in the same relative position with respect to the remaining attributes. For instance, Fig. 2 shows an excerpt of the HTML code of the page in Fig. 1. As it can be seen, it verifies the aforementioned observations.

HTML pages can also be represented as DOM trees as shown in Fig. 3. The representation as a DOM tree of the pages verifying the above observations has the following properties:

- ***Property 1*:** Each record in the DOM tree is disposed in a set of consecutive sibling subtrees. Additionally, although it cannot be derived strictly from the above observations, it is heuristically found that a data record comprises a certain number of *complete* subtrees. For instance, in Fig. 3 the first two subtrees form the first record, and the following three subtrees form the second record.
- ***Property 2:*** The occurrences of each attribute in several records have the same path from the root in the DOM tree. For instance, in Fig. 3 it can be seen how all the instances of the attribute *title* have the same path in the DOM tree, and the same applies to the remaining attributes.

3 Finding the Dominant List of Records in a Page

In this section, we describe how we locate the data region of the page containing the main list of records in the page.

From the property 1 of the previous section, we know finding the data region is equivalent to finding the common parent node of the sibling subtrees forming the data records. The subtree having as root that node will be the target data region. For instance, in our example of Fig. 3 the parent node we should discover is n_1.

```
<html><body>
 <div> ... </div>
 <div> ... </div>
 <div>
  <table> ... </table>
  <table>
   <tr><td><table>
    <tr><td>
        <span><a>Head First Java. 2nd Edition</a></span>
        <br>by <span>Kathy Sierra and Bert Bates</span>
        <br><span>Paperback</span> - Feb 9. 2005</td>
      <td><img></td></tr></table></td></tr>
  <tr><td>Buy new: <span>29.67€</span>
      Price used: <span>20.00€</span></td></tr>
```

```
<tr><td><table>
    <tr><td>
        <span><a>Java Persistence with Hibernate</a></span>
        <br>by <span>Christian Bauer and Gavin King</span>
        <br><span>Paperback</span> - Nov 24. 2006</td>
      <td><img></td></tr></table></td></tr>
  <tr><td>Buy new: <span>37.79€</span></td></tr>
  <tr><td>Other editions: <span>e-Books & Docs</span>
      </td></tr>
  ...
 </table>
 ...
</div>
</body></html>
```

Fig. 2. HTML source code for page in Fig. 1

Our method for finding the region containing the dominant list of records in a page p consists of the following steps:

1. Let us consider N, the set composed by all the nodes in the DOM tree of p. To each node $n_i \in N$, we will assign a score called s_i. Initially, $\forall_{i=1..|N|} s_i = 0$.
2. Compute T, the set of all the text nodes in N.
3. Divide T into subsets $p_1, p_2,\ldots,p_m$, in a way such that all the text nodes with the same path from the root in the DOM tree are contained in the same p_i. To compute the paths from the root, we ignore tag attributes.
4. For each pair of text nodes belonging to the same group, compute n_j as their deepest common ancestor in the DOM tree, and add 1 to s_j (the score of n_j).
5. Let n_{max} be the node having a higher score. Choose the DOM subtree having n_{max} as root of the desired data region.

Now, we provide the justification for this algorithm. First, by definition, the target data region contains a list of records and each data record is composed of a series of attributes. By property 2, we know all the occurrences of the same attribute have the same path from the root. Therefore, the subtree containing the dominant list in the page will typically contain more texts with the same path from the root than other regions. In addition, given two text nodes with the same path in the DOM tree, the following situations may occur:

1. By property 1, if the text nodes are occurrences of texts in different records (e.g. two values of the same attribute in different records), then their deepest common ancestor in the DOM tree will be the root node of the data region containing all the records. Therefore, when considering that pair in step 4, the score of the correct node is increased. For instance, in Fig. 3 the deepest common ancestor of d_1 and d_3 is n_1, the root of the subtree containing the whole data region.
2. If the text nodes are occurrences from different attributes in the same record, then in some cases, their deepest common ancestor could be a deeper node than the one we are searching for and the score of an incorrect node would be increased. For instance, in the Fig. 3 the deepest common ancestor of d_1 and d_2 is n_2.

By property 2, we can infer that there will usually be more occurrences of the case 1 and, therefore, the algorithm will output the right node. Now, we explain the reason for this. Let us consider the pair of text nodes (t_{11},t_{12}) corresponding with the occurrences of two attributes in a record. (t_{11}, t_{12}) is a pair in the case 2. But, by property 2, for each record r_i in which both attributes appear, we will have pairs (t_{11}, t_{i1}), (t_{11}, t_{i2}), (t_{12}, t_{i1}), (t_{12}, t_{i2}), which are in case 1. Therefore, in the absence of optional fields, it can be easily proved that there will be more pairs in the case 1. When optional fields exist, it is still very probable.

This method tends to find the list in the page with the largest number of records and the largest number of attributes in each record. When the pages we want to extract data from have been obtained by executing a query on a web form, we are typically interested in extracting the data records that constitute the answer to the query, even if it is not the larger list (this may happen if the query has few results). If the executed query is known, this information can be used to refine the above method. The idea is very simple: in the step 2 of the algorithm, instead of using all the text

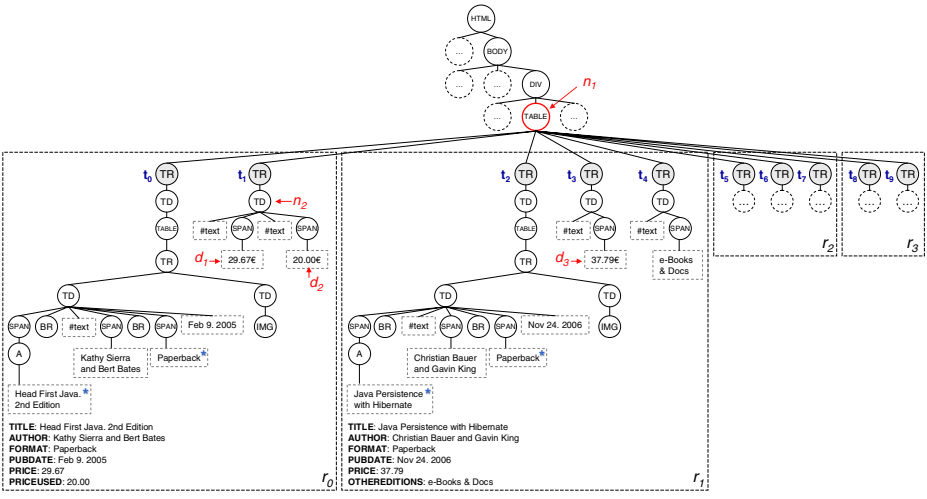


Fig. 3. DOM tree for HTML page in Fig. 1

nodes in the DOM tree, we will use only those text nodes containing text values used in the query with operators whose semantic be *equals* or *contains*. For instance, let us assume the page in Fig. 3 was obtained by issuing a query we could write as *(title contains 'java') AND (format equals 'paperback')*. Then, the only text nodes considered in step 2 would be the ones marked with an '*' in Fig. 3.

4 Dividing the List into Records

Now we proceed to describe our techniques for segmenting the data region in fragments, each one containing at most one data record.

Our method can be divided into the following steps:

- Generate a set of candidate record lists. Each candidate record list will propose a particular division of the data region into records.
- Choose the best candidate record list. The method we use is based on computing an auto-similarity measure between the records in the candidate record lists. We choose the record division lending to records with the higher similarity.

Sections 4.2 and 4.3 describe in detail each one of the two steps. Both tasks need a way to estimate the similarity between two sequences of consecutive sibling subtrees in the DOM tree of a page. The method we use for this is described in section 4.1.

4.1 Edit-Distance Similarity Measure

To compute "similarity" measures we use techniques based in string edit-distance algorithms. More precisely, to compute the *edit-distance similarity* between two sequences of consecutive sibling subtrees named r_i and r_j in the DOM tree of a page, we perform the following steps:

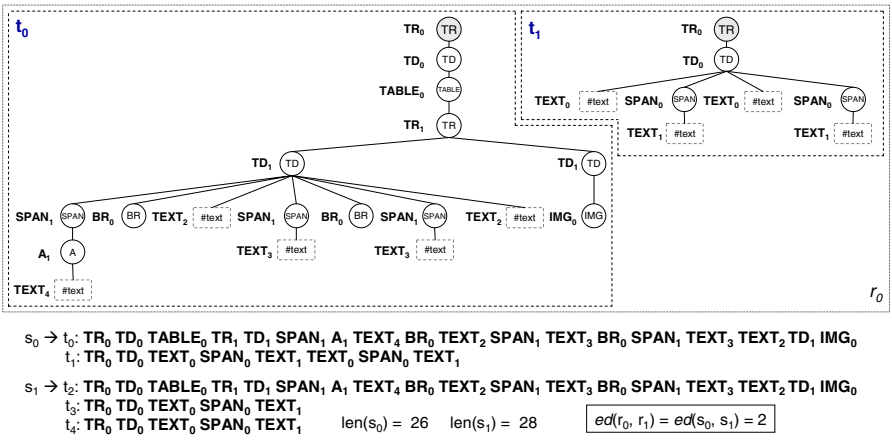


Fig. 4. Strings obtained for the records r_0 and r_1 in Fig. 3

1. We represent r_i and r_j as strings (we will term them s_i and s_j). It is done as follows:
   a. We substitute every text node by a special tag called *text*.
   b. We traverse each subtree in depth first order and, for each node, we generate a character in the string. A different character will be assigned to each tag having a different path from the root in the DOM tree. Fig. 4 shows the strings s_0 and s_1 obtained for the records r_0 and r_1 in Fig. 3.
2. We compute the *edit-distance similarity* between r_i and r_j , denoted as *es* (r_i, r_j), as the string edit distance between s_i and s_j (*ed*(r_i,r_j)) calculated using a variant of the Levenshtein algorithm [8], which does not allow substitution operations (only insertions and deletions are permitted). To obtain a similarity score between 0 and 1, we normalize the result using the equation (1). In our example from Fig. 4, the similarity between r_0 and r_1 is *1- (2 / (26+28)) = 0.96.*

$$es(r_i, r_j) = 1 - ed(s_i, s_j) / len(s_i) + len(s_j) \quad (1)$$

4.2 Generating the Candidate Record Lists

In this section, we describe how we generate a set of candidate record lists inside the data region previously chosen. Each candidate record list will propose a particular division of the data region into records.

By property 1, every record is composed of one or several consecutive sibling subtrees, which are direct descendants of the root node of the data region. We could leverage on this property to generate a candidate record list for each possible division of the subtrees verifying it. Nevertheless, the number of possible combinations would be too high: if the number of subtrees is n, the possible number of divisions verifying property 1 is 2^{n-1}(notice that different records in the same list may be composed of a different number of subtrees, as for instance r_0 and r_1 in Fig. 3). In some sources, n can be low, but in others it may reach values in the hundreds (e.g. a source showing 25 data records, with an average of 4 subtrees for each record). Therefore, this

exhaustive approach is not feasible. The remaining of this section explains how we overcome these difficulties.

Our method has two stages: clustering the subtrees according to their similarity and using the groups to generate the candidate record divisions.

Grouping the subtrees. For grouping the subtrees according to their similarity, we use a clustering-based process we describe in the following lines:

1. Let us consider the set $\{t_1,\ldots,t_n\}$ of all the subtrees which are direct children of the node chosen as root of the data region. Each t_i can be represented as a string using the method described in section 4.1. We will term these strings as $s_1,\ldots,s_n$.
2. Compute the *similarity matrix*. This is a *nxn* matrix where the *(i,j)* position (denoted m_{ij}) is obtained as $es(t_i, t_j)$, the *edit-distance similarity* between t_i and t_j.
3. We define the *column similarity* between t_i and t_j, denoted $cs(t_i,t_j)$, as the inverse of the average absolute error between the columns corresponding to t_i and t_j in the similarity matrix (2). Therefore, to consider two subtrees as similar, the column similarity measure requires their columns in the similarity matrix to be very similar. This means two subtrees must have roughly the same *edit-distance similarity* with respect to the rest of subtrees in the set to be considered as similar. We have found *column similarity* to be more robust for estimating similarity between t_i and t_j in the clustering process than directly using $es(t_i, t_j)$.
4. Now, we apply bottom-up clustering [3] to group the subtrees. The basic idea behind this kind of clustering is to start with one cluster for each element and successively combine them into groups within which inter-element similarity is high, collapsing down to as many groups as desired.

$$cs(t_i,t_j)=1-\sum_{k=1..n}\left|m_{ik}-m_{jk}\right|/n \qquad (2) \qquad s(\Phi)=2/|\Phi|\left(|\Phi|-1\right)\sum_{t_i,t_j\in\Phi} cs(t_i,t_j) \qquad (3)$$

Fig. 5 shows the pseudo-code for the bottom-up clustering algorithm. Inter-element similarity of a set Φ is estimated using the *auto-similarity* measure ($s(\Phi)$), and it is computed as specified in (3).

1. *Let each subtree t be in a singleton group {t}*
2. *Let G be the set of groups*
3. *Let Ω_g be the group-similarity threshold and*
   Ω_e be the element-similarity threshold
4. *While |G| > 1 do*
   - 4.1 *choose $\Gamma,\Delta\in G$, a pair of groups which maximize the auto-similarity measure $s(\Gamma\cup\Delta)$ (see equation 3). The set $\Gamma\cup\Delta$ must verify:*
     - a) $s(\Gamma\cup\Delta)>\Omega_g$
     - b) $\forall i\in\Gamma\cup\Delta,\ j\in\Gamma\cup\Delta, cs(i,j)>\Omega_e$
   - 4.2 *if no pair verifies the above conditions, then stop*
   - 4.3 *remove Γ and Δ from G*
   - 4.4 *let $\Phi=\Gamma\cup\Delta$*
   - 4.5 *insert Φ into G*
5. *End while*

Fig. 5. Pseudo-code for bottom-up clustering

We use *column similarity* as the similarity measure between t_i and t_j. To allow a new group to be formed, it must verify two thresholds:

- The global auto-similarity of the group must reach the *auto-similarity threshold* Ω_g. In our current implementation, we set this threshold to 0.9.
- The column similarity between every pair of elements from the group must reach the *pairwise-similarity threshold* Ω_e. This threshold is used to avoid creating groups that, although showing high overall auto-similarity, contain some dissimilar elements. In our current implementation, we set this threshold to 0.9.

Generating the candidate record divisions. For generating the candidate record divisions, we assign an identifier to each of the generated clusters. Then, we build a sequence by listing in order the subtrees in the data region, representing each subtree with the identifier of the cluster it belongs to. For instance, in our example of Fig. 3, the algorithm generates three clusters, leading to the string $c_0c_1c_0c_2c_2c_0c_1c_2c_0c_1$.

The data region may contain, either at the beginning or at the end, some subtrees that are not part of the data. For instance, these subtrees may contain information about the number of results or web forms to navigate to other result intervals. These subtrees will typically be alone in a cluster, since there are not other similar subtrees in the data region. Therefore, we pre-process the string from the beginning and from the end, removing tokens until we find the first cluster identifier that appears more than once in the sequence. In some cases, this pre-processing is not enough and some additional subtrees will still be included in the sequence. Nevertheless, they will typically be removed from the output as a side-effect of the final stage (see section 5).

Once the pre-processing step is finished, we proceed to generate the candidate record divisions. By property 1, we know each record is formed by a list of consecutive subtrees (i.e. characters in the string). From our page model, we know records are encoded consistently. Therefore, the string will tend to be formed by a repetitive sequence of cluster identifiers, each sequence corresponding to a data record. The sequences for two records may be slightly different. Nevertheless, we will assume they always either start or end with a subtree belonging to the same cluster (i.e. all the data records always either start or end in the same way). This is based on the following heuristic observations:

- In many sources, records are visually delimited in an unambiguous manner to improve clarity. This delimiter is present before or after every record.
- When there is not an explicit delimiter between data records, the first data fields appearing in a record are usually *key* fields appearing in every record.

Based on the former observations, we will generate the following candidate lists:

- For each cluster c_i, $i=1..k$, we will generate two candidate divisions: one assuming every record starts with c_i and another assuming every record ends with c_i. For instance, Fig. 6 shows the candidate divisions obtained for the example of Fig. 3.
- In addition, we will add a candidate record division considering each record is formed by exactly one subtree.

This reduces the number of candidate divisions from 2^{n-1}, where n is the number of subtrees, to $1+2k$, where k is the number of generated clusters, turning feasible to evaluate each candidate list to choose the best one.

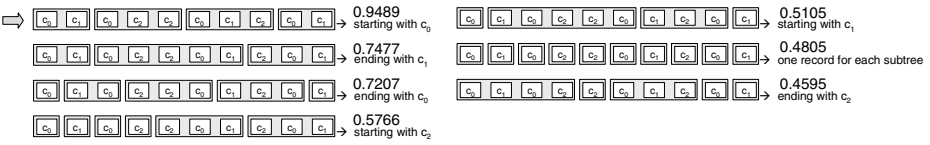


Fig. 6. Candidate record divisions obtained for example page from Fig. 3

4.3 Choosing the Best Candidate Record List

To choose the correct candidate record list, we rely on the observation that the records in a list tend to be similar to each other. Therefore, we will choose the candidate list showing the highest auto-similarity.

Given a candidate list composed of the records $<r_1, \ldots, r_n>$, we compute its auto-similarity as the weighted average of the *edit-distance similarities* between each pair of records of the list. The contribution of each pair to the average is weighted by the length of the compared registers. See equation 4.

$$\sum_{i=1..n,\, j=1..n, i\neq j} es(r_i, r_j)\ \left(len(r_i)+len(r_j)\right)\ \ /\ \ \sum_{i=1..n,\, j=1..n, i\neq j} len(r_i)+len(r_j) \tag{4}$$

For instance, in Fig. 6, the first candidate record division is chosen.

5 Extracting the Attributes of the Data Records

In this section, we describe our techniques for extracting the values of the attributes of the data records identified in the previous section.

The basic idea consists in transforming each record from the list into a string using the method described in section 4.1, and then using *string alignmen*t techniques to identify the attributes in each record. An alignment between two strings matches the characters in one string with the characters in the other one, in such a way that the edit-distance between the two strings is minimized. There may be more than one optimal alignment between two strings. In that case, we choose any of them.

For instance, Fig. 7a shows an excerpt of the alignment between the strings representing the records in our example. Each aligned text token roughly corresponds with an attribute of the record. Notice that to obtain the actual value for an attribute we may need to remove common prefixes/suffixes found in every occurrence of an attribute. For instance, in our example, to obtain the value of the *price* attribute we would detect and remove the common suffix "€". In addition, those aligned text nodes having the same value in all the records (e.g. "*Buy new:*", "*Price used:*") will be considered "labels" instead of attribute values and will not appear in the output.

To achieve our goals, it is not enough to align two records: we need to align all of them. Nevertheless, optimal multiple string alignment algorithms have a complexity of $O(n^k)$. Therefore, we need to use an approximation algorithm. Several methods have been proposed for this task [10][6]. We use a variation of the *center star* approximation algorithm [6], which is also similar to a variation used in [14] (although they use tree alignment). The algorithm works as follows:

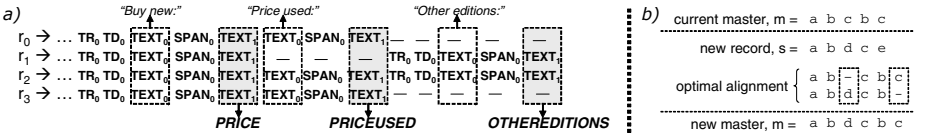


Fig. 7. (a) Alignment between records of Fig. 1 (b) example of alignment with the master

1. The longest string is chosen as the "master string", *m*.
2. Initially, *S*, the set of "still not aligned strings" contains all the strings but *m*.
3. For every *s* ∈ *S*, align *s* with *m*. If there is only one optimal alignment between *s* and *m* and the alignment matches any null position in *m* with a character from *s*, then the character is added to *m* replacing the null position (an example is shown in Fig. 7b). Then, *s* is removed from *S*.
4. Repeat step 3 until *S* is empty or the master string *m* does not change.

6 Experience

This section describes the empirical evaluation of our techniques. During the development phase, we used a set of 20 pages from 20 different web sources. The tests performed with these pages were used to adjust the algorithm and to choose suitable values for the used thresholds.

For the experimental tests, we chose 200 new websites in different application domains (book and music shops, patent information, publicly financed R&D projects, movies information, etc). We performed one query in each website and collected the first page containing the list of results. Some queries returned only 2-3 results while others returned hundreds of results. The collection of pages is available online[*].

While collecting the pages for our experiments, we found three data sources where our page creation model is not correct. Our model assumes that all the attributes of a data record are shown contiguously in the page. In those sources, the assumption does not hold and, therefore, our system would fail. We did not consider those sources in our experiments. In the related work section, we will further discuss this issue.

We measured the results at three stages of the process: after choosing the data region containing the dominant list of data records, after choosing the best candidate record division and after extracting the structured data contained in the page. Table 1 shows the results obtained in the empirical evaluation.

In the first stage, we use the information about the executed query, as explained at the end of section 3. As it can be seen, the data region is correctly detected in all pages but two. In those cases, the answer to the query returned few results and there was a larger list on a sidebar of the page containing items related to the query.

In the second stage, we classify the results in two categories: *correct* when the chosen record division is correct, and *incorrect* when the chosen record division contains some incorrect records (not necessarily all). For instance, two different records may be concatenated as one or one record may appear segmented into two.

[*] http://www.tic.udc.es/~mad/resources/projects/dataextraction/testcollection_0507.htm

As it can be seen, the chosen record division is correct in the 93.50% of the cases. It is important to notice that, even in incorrect divisions, there will usually be many correct records. Therefore, stage 3 may still work fine for them. The main reason for the failures at this stage is that, in a few sources, the auto-similarity measure described in section 4.3 fails to detect the correct record division, although it is between the candidates. This happens because, in these sources, some data records are quite dissimilar to each other. For instance, in one case where we have two consecutive data records that are much shorter than the remaining, and the system chooses a candidate division that groups these two records into one.

In stage 3, we use the standard metrics *recall* and *precision*. These are the most important metrics in what refers to web data extraction applications because they measure the system performance at the end of the whole process. As it can be seen, the obtained results are very high, reaching respectively to 98.55% and 97.39%. Most of the failures come from the errors propagated from the previous stage.

7 Related Work

Wrapper generation techniques for web data extraction have been an active research field for years. Many approaches have been proposed [2][9][11][12][13]. [7] provides a brief survey.

All the wrapper generation approaches require some kind of human intervention to create and configure the wrapper. When the sources are not known in advance, such as in focused crawling applications, this approach is not feasible.

Several works have addressed the problem of performing web data extraction tasks without requiring human input. IEPAD [4] uses the Patricia tree [6] and string alignment techniques to search for repetitive patterns in the HTML tag string of a page. The method used by IEPAD is very probable to generate incorrect patterns along with the correct ones, so human post-processing of the output is required.

RoadRunner [5] receives as input multiple pages conforming to the same template and uses them to induce a union-free regular expression (UFRE) which can be used to extract the data from the pages conforming to the template. The basic idea consists in performing an iterative process where the system takes the first page as initial UFRE and then, for each subsequent page, tests if it can be generated using the current template. If not, the template is modified to represent also the new page. The proposed method cannot deal with disjunctions in the input schema and it requires receiving as input multiple pages conforming to the same template.

Table 1. Results obtained in the empirical evaluation

Stage 1	**# Correct**	**# Incorrect**	**% Correct**
	198	*2*	*99.00*
Stage 2	**# Correct**	**# Incorrect**	**% Correct**
	187	*13*	*93.50*
			Precision
Stage 3	**# Records to Extract**	*3557*	*98.55*
	# Extracted Records	*3515*	**Recall**
	# Correct Extracted Records	*3464*	*97.39*

As well as RoadRunner, ExAlg receives as input multiple pages conforming to the same template and uses them to induce the template and derive a set of data extraction rules. ExAlg makes some assumptions about web pages which, according to the own experiments of the authors, do not hold in a significant number of cases: for instance, it is assumed that the template assign a relatively large number of tokens to each type constructor. It is also assumed that a substantial subset of the data fields to be extracted have a unique path from the root in the DOM tree of the pages. It also requires receiving as input multiple pages.

[14] presents DEPTA, a method that uses the visual layout of information in the page and tree edit-distance techniques to detect lists of records in a page and to extract the structured data records that form it. As well as in our method, DEPTA requires as input one single page containing a list of structured data records. They also use the observation that, in the DOM tree of a page, each record in a list is composed of a set of consecutive sibling subtrees. Nevertheless, they make two additional assumptions: 1) that exactly the same number of sub-trees must form all records, and 2) that the visual gap between two data records in a list is bigger than the gap between any two data values from the same record. Those assumptions do not hold in all web sources. For instance, neither of the two assumptions holds in our example page of Fig. 3. In addition, the method used by DEPTA to detect data regions is considerably more expensive than ours.

A limitation of our approach arises in the pages where the attributes constituting a data record are not contiguous in the page. Those cases do not conform to our page creation model and, therefore, our current method is unable to deal with them. Although DEPTA assumes a page creation model similar to the one we use, after detecting a list of records, they try to identify these cases and transform them in "conventional" ones before continuing the process. These heuristics could be adapted to work with our approach.

References

1. Arasu, A., Garcia-Molina, H.: Extracting Structured Data from Web Pages. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data (2003)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proc. of Very Large DataBases (VLDB) (2001)
3. Chakrabarti, S.: Mining the Web: Discovering Knowledge from Hypertext Data. Morgan Kaufmann Publishers, San Francisco (2003)
4. Chang, C., Lui, S.: IEPAD: Information extraction based on pattern discovery. In: Proc. of 2001 Int. World Wide Web Conf., pp. 681–688 (2001)
5. Crescenzi, V., Mecca, G., Merialdo, P.: ROADRUNNER: Towards automatic data extraction from large web sites. In: Proc. of the 2001 Int. VLDB Conf., pp. 109–118 (2001)
6. Gonnet, G.H., Baeza-Yates, R.A., Snider, T.: New Indices for Text: Pat trees and Pat Arrays. Information Retrieval: Data Structures and Algorithms. Prentice Hall, Englewood Cliffs (1992)
7. Laender, A.H.F., Ribeiro-Neto, B.A., Soares da Silva, A., Teixeira, J.S.: A Brief Survey of Web Data Extraction Tools. ACM SIGMOD Record 31(2), 84–93 (2002)

8. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10, 707–710 (1966)
9. Muslea, I., Minton, S., Knoblock, C.: Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems, 93–114 (2001)
10. Notredame, C.: Recent Progresses in Multiple Sequence Alignment: A Survey. Technical report, Information Genetique et (2002)
11. Pan, A., et al.: Semi-Automatic Wrapper Generation for Commercial Web Sources. In: Proc. of IFIP WG8.1 Conf. on Engineering Inf. Systems in the Internet Context (EISIC) (2002)
12. Raposo, J., Pan, A., Álvarez, M., Hidalgo, J.: Automatically Maintaining Wrappers for Web Sources. Data & Knowledge Engineering 61(2), 331–358 (2007)
13. Zhai, Y., Liu, B.: Extracting Web Data Using Instance-Based Learning. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 318–331. Springer, Heidelberg (2005)
14. Zhai, Y., Liu, B.: Structured Data Extraction from the Web Based on Partial Tree Alignment. IEEE Trans. Knowl. Data Eng. 18(12), 1614–1628 (2006)

Towards Transparent Personal Content Storage in Multi-service Access Networks

Koert Vlaeminck, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester

Ghent University, Dept. of Information Technology - IBBT - IMEC
Gaston Crommenlaan 8 bus 201, B-9050 Gent, Belgium
Tel.: +32 (0)9 331 49 42, Fax: +32 (0)9 331 48 99
koert.vlaeminck@intec.ugent.be

Abstract. Anytime, anywhere and anyhow access to personalized services requires the complete decoupling of devices for accessing the service and the supporting personal data storage. When deploying such transparent personalized services, an important question that needs answering is where to install the storage servers. In this respect, this paper considers the deployment of a personal content storage service in multi-service access networks. The storage server placement problem is formulated as a binary integer linear programming (BILP) problem and a heuristic storage server placement algorithm (SSPA) is presented and evaluated. First it is assumed that servers do not fail. Consequently, the problem formulation is extended to include replication and striping and both BILP and heuristic methods are modified to cope with the additional constraints. The extended SSPA heuristic is used to analyze several resilience and striping scenarios. It is shown that the SSPA produces close to optimal results and is very efficient for optimizing server placement in personal content storage deployments.

Keywords: Distributed Storage, Server Placement, Resilience, Access Network, Personal Content.

1 Introduction

In a converged media world consumers increasingly call the shots. No longer a captive, mass media audience, today's consumer is unique, demanding, and engaged. He wants anytime, anywhere and anyhow access to a personalized experience, generates his own content, mixes it, and shares it on a growing number of social networks [1]. Transparent anytime, anywhere access to such a personalized media experience requires the complete decoupling of devices for accessing the data and the actual data storage. In pursuing transparency, one important question that needs answering is where to install the storage servers. A major opportunity of emerging, converged, multi-service IP access and aggregation networks [2] is providing consumers with transparent storage, enabling anytime, anywhere access to a personal data collection, consisting of both *acquired* and *created* content.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 479–492, 2007.

Storage of acquired content (e.g. a digital movie and music collection) has many similarities with content distribution networks (CDNs), where content is replicated to regional servers at the edge of the network in order to tackle the performance issues of a classical central server based approach [3]. Acquired content is typically read-only and relatively popular, making it an ideal fit for content distribution networks.

Storage of created content (e.g. personalized play lists or even remixes, digital photo albums and home videos) is entirely different. Because of its read/write nature, delay is much more important and replication or caching close to the consumer a must. Furthermore, created personal content is typically a lot less popular—although the ability to share created content with (a limited amount of) other users is still desirable—and requires an ever increasing storage capacity. Where CDNs are designed to distribute a limited amount of very popular content, a (created) personal content storage service stores a huge amount of relatively unpopular content. CDN-based systems assign storage capacity per *item*, while a personal content storage service assigns capacity per *user*.

This paper presents and evaluates an algorithm for determining the best locations for deploying a storage service supporting created content, minimizing the deployment cost while guaranteeing customer satisfaction. Both access and aggregation nodes are candidate storage server locations. Servers have limited storage- and read/write capacity and can only serve a limited number of simultaneous users. Furthermore, the access and aggregation network only has limited bandwidth assigned to the storage service. Within this restricted environment, consumers expect a guaranteed minimum throughput and maximum delay for accessing their content.

After a discussion of related work in Section 2, the remainder of this paper is structured as follows: First, Sections 4 to 6 assume that system crashes and disasters are nonexistent. In that case, no redundancy is required and each data item is stored only once. These sections extend earlier work on dimensioning *read-mostly* data storage in the access and aggregation network to support *read-write* content [4]. Both a Binary Integer Linear Programming (BILP) [5] and a heuristic solution are presented. The heuristic Storage Server Placement Algorithm (SSPA) is evaluated by comparing its placement decisions to the optimal solution, acquired from a direct implementation of the BILP formulation. After that, Section 7 studies the more realistic situation where servers do fail. Both BILP solution and SSPA heuristic are extended to include replication and striping and some simulation results, illustrating the requirements for striping and resilience in terms of number of server locations and total accumulated bandwidth on all network links, are presented. Finally Section 8 summarizes the main results of this paper.

2 Related Work

Many distributed filesystems exist today, ranging from client-server systems such as NFS [6], AFS [7] and Coda [8] over cluster filesystems (e.g. Lustre [9],

GPFS [10] and the Google File System [11]) to global scale peer-to-peer filesystems (e.g. OceanStore [12], FARSITE [13] and Pangaea [14]).

None of the above filesystems were designed for large-scale deployment in a service aware access and aggregation network environment. However, two of the peer-to-peer filesystems seem good candidates for this purpose: OceanStore and Pangaea. One of OceanStore's design goals is support for nomadic data using a *promiscuous caching* policy, allowing data to be cached anytime, anywhere. OceanStore uses introspection for replica management, optimizing the number and location of replicas based on observation of the access requests. Pangaea, on the other hand, uses a *pervasive replication* mechanism that replicates data wherever and whenever it is accessed.

Guaranteeing fast access to a distributed filesystem, requires replication on nodes close to the user: Pangaea, with its pervasive replication mechanism, supports this by design. Another interesting replication mechanism is *fluid replication*, which creates a replica on a nearby node when the connection quality between a client and its current server becomes poor [15]. Whichever replication mechanism is chosen, a minimum set of replicas should be maintained at all times in order to ensure reliability.

Before deciding on how and where data is replicated, an important question that needs answering is where to install the storage servers. This paper investigates how servers can be deployed in the access and aggregation network, supporting fast and reliable personal content storage, minimizing cost, while guaranteeing user satisfaction. In literature, server placement was already discussed in a web server environment [16] and for the placement of regional servers for content distribution networks [3]. However, as discussed in the introduction, personal content storage, supporting created content, has different requirements.

3 Use Cases

Throughout this paper, the two typical access and aggregation network topologies were selected as use cases: a mesh of trees topology, used in DSL deployment, and a ring of rings topology, used in cable Internet access deployment, both consisting of 250 access nodes. In both topologies, all nodes are assumed to be service-aware and thus candidate locations for installing storage servers.

The mesh of trees topology is depicted in Fig. 1(a). The 250 access nodes are at the leaf nodes of five trees of depth two, aggregated per ten at depth two and per five at depth one. The available bandwidth between the access nodes and the first aggregation node is 600 Mbps. Those aggregation nodes are connected to the root of a tree, each by a 1.2 Gbps link. The five trees are interconnected by a full mesh of 3 Gbps links. The ring of rings topology, is depicted in Fig. 1(b). It consists of five unidirectional six-node secondary rings of 2 Gbps. One node connects the secondary ring to a primary ring of 10 Gbps (also unidirectional). The other five secondary ring nodes each connect to ten access nodes trough a shared medium of 2 Gbps, resulting in a total of 250 access nodes.

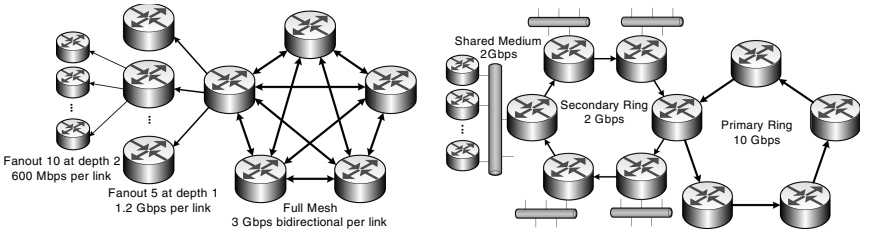


Fig. 1. Topology (a): mesh of trees access and aggregation network topology, used for DSL deployment. Topology (b): ring of rings access and aggregation network topology, used for cable Internet access deployment.

4 Model Description

This paper studies the deployment of a personal content storage service, supporting created content, in the access and aggregation network. The goal is to minimize the number of server locations, while guaranteeing fast access from any access node. Before discussing the storage server placement algorithm in the next section, this section introduces some definitions and formalizes the problem description by presenting it as a Binary Integer Linear Programming (BILP) problem [5].

Define $N = \{N_1, \ldots, N_n\}$ as the set of all nodes and $A = \{A_1, \ldots, A_m\}$ as the set of access nodes ($A \subset N$). Furthermore, define $E = \{E_1, ..., E_o\}$ as the set of all edges. An edge $E_l \in E$ represent a simplex link in the network. Delay over E_l is defined as c_l and its bandwidth as b_l. Finally define $U = \{U_1, \ldots, U_p\}$ as the set of all users.

Server limitations are defined as follows: a server location has storage capacity S, total read speed (download capacity) D, total write speed (upload capacity) D' and supports at most V simultaneous users.

Now the service requirements can be defined: a user U_u at access node A_j ($U_u@A_j$) requires a certain amount of storage capacity S_{ju}, with a downstream (read) bandwidth D_{ju} and an upstream (write) bandwidth D'_{ju}. The total required storage capacity, downstream bandwidth, upstream bandwidth from access node A_j is $S_j = \sum_{U_u@A_j} S_{ju}$, $D_j = \sum_{U_u@A_j} D_{ju}$, $D'_j = \sum_{U_u@A_j} D'_{ju}$ respectively. The maximum delay for accessing the storage service is defined as d.

Finally, before describing the BILP problem, some variables need to be defined: s_i is a binary variable indicating whether servers are installed at node N_i or not, while s_{iju} is a binary variable representing whether user U_u at access node A_j is served by a server at node N_i. Defining P_{ij} as the set of all downstream paths from node N_i to access node A_j, then p_{ijk} is a continuous variable representing the flow on the k^{th} path P_{ijk} from N_i to A_j and $p_{ij} = \sum_{P_{ijk} \in P_{ij}} p_{ijk}$ is the total flow from N_i to A_j. Analogously, defining P'_{ij} as the set of all upstream paths from access node A_j to node N_i, then p'_{ijk} is a continuous variable

representing the flow on the k^{th} path P'_{ijk} from A_j to N_i and $p'_{ij} = \sum_{P'_{ijk} \in P'_{ij}} p'_{ijk}$ is the total flow from A_j to N_i. The BILP problem can now be described as follows:

Minimize:

$$z = \sum_{N_i \in N} s_i \tag{1}$$

subject to:

$$p_{ij} = \sum_{U_u @ A_j} s_{iju}.D_{ju}/b, p'_{ij} = \sum_{U_u @ A_j} s_{iju}.D'_{ju}/b, \forall N_i \in N, \forall A_j \in A \tag{2}$$

$$p_{ij} \leq s_i.D_j/b, p'_{ij} \leq s_i.D'_j/b, \forall N_i \in N, \forall A_j \in A \tag{3}$$

$$\sum_{N_i \in N} s_{iju} \geq R, \forall A_j \in A, \forall U_u \in U \tag{4}$$

$$\sum_{N_i \in N} p_{ij} = R.D_j/b, \sum_{N_i \in N} p'_{ij} = R.D'_j/b, \forall A_j \in A \tag{5}$$

$$\sum_{\forall P_{ijk} \ni E_l} p_{ijk} + \sum_{\forall P'_{ijk} \ni E_l} p'_{ijk} \leq b_l, \forall E_l \in E \tag{6}$$

$$\sum_{E_l \in P_{ijk}} c_l \leq d, \sum_{E_l \in P'_{ijk}} c_l \leq d, \forall N_i \in N, \forall A_j \in A, \forall P_{ijk} \in P_{ij}, \forall P'_{ijk} \in P'_{ij} \tag{7}$$

$$\sum_{A_j \in A} p_{ij} \leq D, \sum_{A_j \in A} p'_{ij} \leq D', \forall N_i \in N \tag{8}$$

$$\sum_{A_j \in A} \sum_{U_u @ A_j} s_{iju}.S_{ju}/b \leq S, \forall N_i \in N \tag{9}$$

$$\sum_{A_j \in A} \sum_{U_u @ A_j} s_{iju} \leq V, \forall N_i \in N \tag{10}$$

For now, assume $R = b = 1$, as these parameters are only required for including striping and resilience in the BILP formulation (cf. Section 7). The constraints in (2) guarantee the (down- and upstream) bandwidth requirements between node N_i and access node A_j are met for each user U_u at A_j that is served by a server at N_i. Constraints (3) ensure the total flow from a node N_i to an access node A_j does not exceed the total (down- and upstream) bandwidth demand of A_j. Furthermore, together with the constraints in (2) and the fact that we're dealing with a minimization problem, the constraints in (3) implicitly guarantee that $s_i = 1$ if and only if one of the $s_{iju} = 1$. Constraints in (4) and (5) ensure that each user U_u is assigned at least one server and that total

(down- and upstream) bandwidth demand for each access node A_j is met, respectively. Link bandwidth constraints are described in (6) and delay constraints in (7). Finally, the last three sets of constraints represent the server capabilities: read and write performance of a server in (8), storage capacity in (9) and a maximum number of simultaneous users in (10).

5 Storage Server Placement Algorithm (SSPA)

As BILP solutions to a server placement problem tend to scale poorly, a heuristic algorithm was designed to solve the storage server placement problem, formalized by the BILP model in Section 4. The algorithm consists of two phases. In the first phase, servers are installed one by one at the best candidate locations, until all users are served, while respecting the maximum delay constraint, link bandwidth constraints and server constraints. The quality Q_i of a candidate location N_i is determined using local information, such as the number of access nodes that can be reached from that location, without exceeding the maximum delay d, the average delay to those access nodes and the total available bandwidth on paths with delay $\leq d$ to those access nodes. Once a server location is added, it is used to its maximum capacity (respecting server and network constraints), before a new location is selected. In the second phase, the allocation of servers to users is redistributed in such a way that, where possible, each user is served by the server location closest to his access node. During this redistribution, only locations selected during the first phase are considered. Server locations that no longer serve any users after phase two are removed.

The operation of the first phase is illustrated in Fig. 2. In the figure, Q_i is defined as the number of access nodes in range from N_i. When two candidate nodes N_i have equal quality Q_i, one is selected at random. Once a server location N_i is selected, all access nodes A_j in range from N_i are connected to a helper node T, using helper edges with delay 0 and link bandwidth D_j, the total (remaining) required downstream bandwidth from A_j. That way, a successive shortest path algorithm (from N_i to T) can be used to assign as much users U_u as possible to the newly installed server location N_i. This successive shortest path algorithm continues until N_i has no capacity left, all demand from the access nodes in range is served or no more paths with delay $\leq d$ are available. By using the remaining downstream bandwidth requirements as link capacity for helper edges connecting an access node to T, total capacity assigned to the access node is guaranteed not to exceed total demand, as storage capacity and down and upstream bandwidth are assigned to a single server location on a per user basis. This process is repeated until all users U_u on all access nodes A_j are served.

In the example of Fig. 2, each user requires a server location at maximum two hops from his access node. In the first iteration, N_3 and N_4 both have four access nodes in range. Assume N_4 is selected as the first server location. The access nodes in range from N_4, i.e. A_1, A_2, A_3 and A_4, are connected to a helper node T after which a successive shortest path algorithm is executed to assign users connected through these access nodes to N_4.

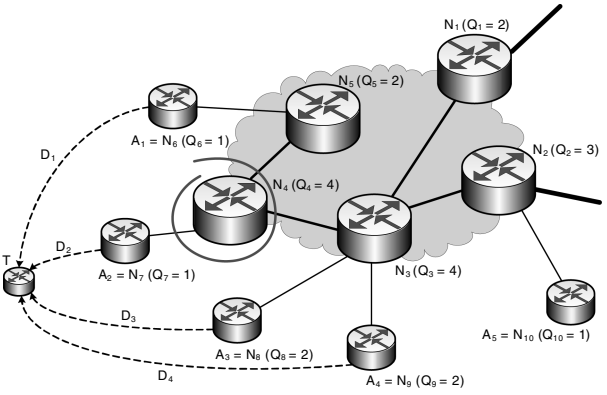


Fig. 2. Operation of the first phase of SSPA. Q_i is defined as the number of access nodes in range from N_i and each user requires a server location at maximum two hops from his access node.

In the second phase, the allocation of servers to users is redistributed in such a way that, where possible, each user is served by the server location closest to his access node. Server locations that become obsolete after this redistribution are removed. The second phase considers server locations added during phase one in reverse order. As phase one only adds a new server location when previously added locations can no longer handle additional user demand, the later added locations are the most unlikely to be expendable. Phase two only adds a server location (out of the server locations selected during phase one) when previously added server locations can't take over all user demand it served after phase one. Phase two only considers server locations on the path from U_u's phase one server location to the access node connecting U_u. That way, phase two is guaranteed to respect link bandwidth constraints.

6 Implementation and Evaluation

The SSPA heuristic was implemented using the Telecom Research Software library (TRS) [17], a Java network library, which was extended to support modeling of personal content storage. For evaluating the heuristic, its placement results were compared to those of a direct implementation of the BILP formulation through CPLEX, a branch and bound solver [18]. In order to keep the ILP calculations feasible, unlimited server capacity was assumed and only downstream bandwidth demand was considered, removing constraints (8), (9) and (10) and all p' related expressions from the BILP problem formulation in Section 4. Furthermore, by only enumerating paths P_{ij} with length $\leq d$, constraint (7) can also be removed and a further speed-up can be achieved.

The number of storage server locations, required for providing a fast storage service, was computed for varying total demand per access node (equally divided between 1000 users connected through each access node) and maximum delay.

	BILP / SSPA		
max # hops	**1**	**2**	**3**
0 - 60 Mbps	25 / 25	5 / 5	1 / 1
61 - 120 Mbps	25 / 25	5 / 5	2 / 2
121 - 600 Mbps	25 / 25	25 / 25	25 / 25
> 600 Mbps	250 / 250	250 / 250	250 / 250

Fig. 3. Simulation results for the mesh of trees topology, depicted in Fig. 1(a). The table gives number of server locations for a varying demand per access node and increasing maximum delay.

For the evaluation, maximum delay for accessing a server is expressed as the number of hops from the access node and the candidate node quality, Q_i, is defined as the number of access nodes in range from node N_i. In order to allow SSPA to break free from suboptimal solutions, a random node is selected from the n best candidate nodes (with n a configurable upper bound) and the best placement after multiple (i.e. ten) runs is used. For instance, in the ring of rings topology of Fig. 1(b), n should be at least 6, to allow SSPA to choose a non-primary ring node as first server location.

Results for the mesh of trees topology are summarized in the table of Fig. 3. For this rather simple topology, the SSPA heuristic produces optimal results. Results from the simulations on the ring of rings topology are depicted in Fig. 4. Here, the SSPA heuristic adds slightly more server locations than the optimal result. Results are still close to optimal, however. Furthermore, each SSPA iteration takes less than a minute on modern PC hardware[1], as opposed to hours to days—depending on the parameters—for the BILP implementation to produce a single result.

7 Striping and Resilience

High availability is an important feature of a network storage system. In a world where harddisks are failure prone and recordable optical media only have limited archival lifespan [19], high availability and guaranteed data retention are major selling points for a high speed network storage service. In order to protect data from node failures, two alternative approaches are considered: whole-file replication and erasure code (or block) replication [20]. In case of whole-file replication, a user's data is replicated on $R > 1$ server locations. The user's data is available, as long as at least one of the R replicas is online. As a drawback, whole-file replication requires R times the storage capacity of storing each data item once. Furthermore, a user's bandwidth requirements must be guaranteed to each of the R replica locations in order to guarantee service quality in case of node failure.

In case of block replication, a data item is split into $b > 1$ blocks. Erasure coding is then applied to these b data fragments, producing $R > b$ fragments of

[1] The simulations were executed on an AMD64 3000+ system with 512 MB of memory, running Linux and the Sun J2SE 5.0 Runtime Environment.

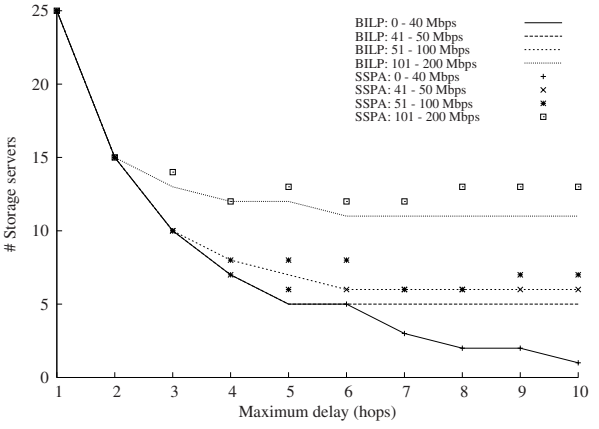


Fig. 4. Simulation results for the ring of rings topology, depicted in Fig. 1(b). Lines represent the optimal BILP results, while dots represent the SSPA results.

the same size as before [21]. The essential property of erasure coding is that any b of the R coded fragments are sufficient to reconstruct the original data item. Each of the R coded fragments are stored at a different server location. A data item is available as long as at least b of the R data fragments are online. Block replication only requires $\frac{R}{b}$ times the storage capacity of storing each data item once. Furthermore, each of the R replica server locations only has to guarantee $\frac{1}{b}$ of a user's bandwidth requirements, at the cost of greater complexity. Note that whole-file replication can be represented as a special case of block replication, where $b = 1$.

Lin, Chiu and Lee [20] showed that the benefit of erasure code (block) replication depends on the node availability, relative to the storage overhead S ($S = R$ for whole-file replication and $S = \frac{R}{b}$ for block replication). Defining μ as the node availability, the file availability is given by:

$$A = \sum_{i=b}^{R} \binom{R}{i} \mu^i (1-\mu)^{R-i} \tag{11}$$

A thorough analysis of the file availability, using different scenarios for both replication schemes, shows that whole-file replication might actually perform better for the same storage overhead S when node availability is low [20].

Striping can be defined as a special case of block replication, where $R = b$. A file is split into b blocks, each block stored at a different server location. Striping provides no resilience—once one of the b server locations goes offline, the file can no longer retrieved—but allows to better distribute the load of the storage system. With the above definitions of $b \geq 1$ and $R \geq b$, the BILP problem description from Section 4 now includes striping and resilience.

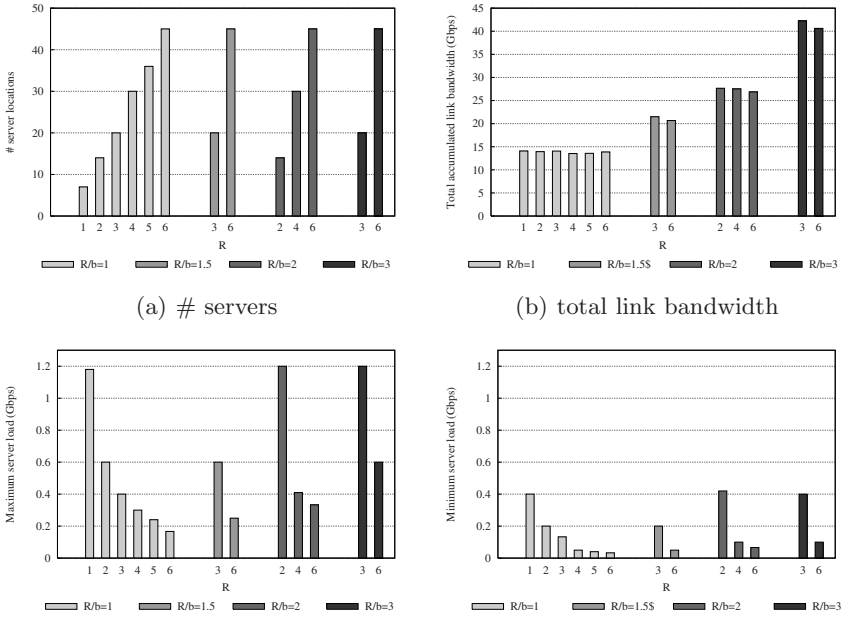
For solving the storage server placement problem, supporting striping and resilience, the SSPA heuristic from Section 5 is extended. Phase one operates

roughly the same, but only assigns $\frac{1}{b}$ of a user's storage and bandwidth requirements to a single server location. A user U_u is only removed from the set of users U when he has R server locations assigned. Furthermore, the link bandwidth of the helper edges E_{A_jT} is initialized at $\frac{D_j}{b}$, as a single server location provides at most $\frac{1}{b}$ of the total required bandwidth from access node A_j. In phase two, an additional check should be made whether a closer server location on the path, from the current server location on that path to an access node, does not already serve (some of) the users on that access node, before relocating these users to that closer server location.

For illustrating the effect of replication and striping on the storage server placement problem, the ring of rings topology from Fig. 1(b) was used. In order to keep the simulations interpretable, read operations are assumed to be dominating the demand. First, server locations are assumed to have unlimited capacity, in order to analyze network limitations and SSPA's load distribution. After that, the number of simultaneous users per server location is limited by installing a maximum read capacity. The quality metric Q_i for selecting the best candidate server location N_i was constructed in such a way that the nodes N_i with the highest number of access nodes in range and the lowest average delay to those access nodes are the most likely to be chosen. Again, for avoiding local optima, a location is randomly selected from the n best candidates ($n > 6$, if multiple nodes have equal quality, they are all considered). The figures show the best result after five runs. Again, the hop-count was used as a delay measure. Simulation results include the total number of server locations, the total accumulated link bandwidth (the sum of the bandwidth usage on all links in the network), and bandwidth requirements at the least and the most loaded server location.

Assuming unlimited server capacity, simulations were run for varying service requirements: maximum delay, read bandwidth per access node, number of replica locations (R) and number of blocks (b) per data item. Simulations results show that for a fixed storage overhead $\frac{R}{b}$, the total accumulated link bandwidth (hence the bandwidth cost for deploying the service) is constant for increasing R, when the network has sufficient bandwidth available (small fluctuations are explained by the heuristic nature of SSPA). Furthermore, as one would expect, for fixed $\frac{R}{b}$ the total number of server locations increases and the maximum server load decreases for increasing R. This is illustrated in Fig. 5 for a maximum delay of 4 hops and a read bandwidth of 20 Mbps per access node.

As SSPA seeks to minimize the number of server locations, it will tend to maximize the usage of each server location. As a side-effect, load is not evenly distributed among the available server locations. This is clearly visible in Fig. 5(c) and 5(d): e.g. in the case where each data item is only stored at one server location ($\frac{R}{b} = 1$ and $R = 1$), the maximum loaded server location provides almost 25% of the 5 Gbps total required bandwidth, while the least loaded server location only provides 400 Mbps. Work is in progress to design a third phase for SSPA, providing better balancing of the load over the available server locations, using a modified minimum cost flow algorithm—from the server locations



(a) # servers (b) total link bandwidth

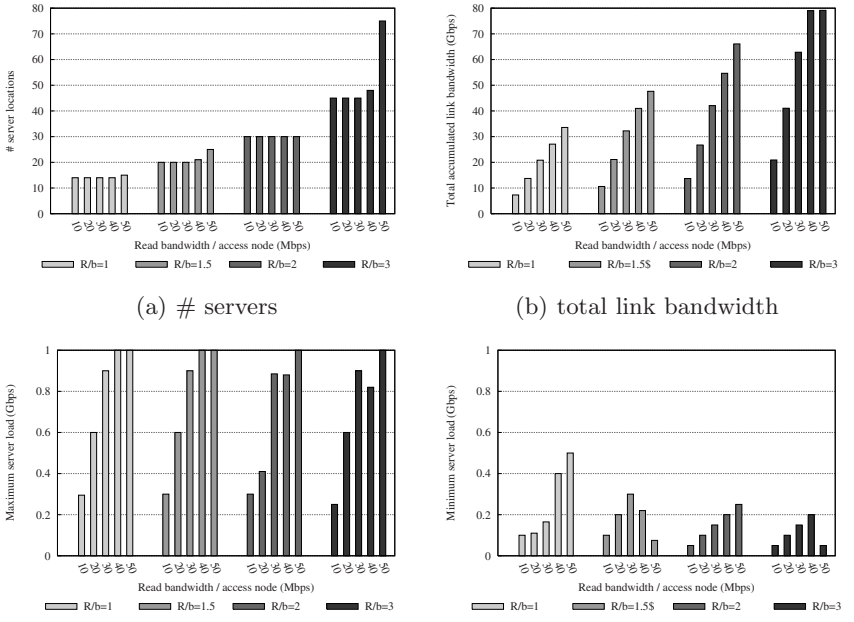
(c) maximum server load (d) minimum server load

Fig. 5. Number of server locations, total link bandwidth, maximum and minimum server location read bandwidth for varying $\frac{R}{b}$ and increasing R. Service requirements are a maximum delay of 4 hops and a read bandwidth of 20 Mbps per access node. Server capacity is assumed unlimited.

remaining after phase two to the access nodes—with dynamic costs for avoiding saturated links and heavily loaded locations, i.e. costs $\sim \frac{1}{1-load(\%)}$ [3].

For the remainder of the simulations, server location read capacity is restricted to 1 Gbps[2]. Furthermore, the maximum delay is set to 4 hops and the number of fragments per data item is set to $b = 2$. Fig. 6 summarizes simulation results for varying storage overhead $\frac{R}{b}$ and increasing read bandwidth demand per access node. From these simulations it is clear that SSPA effectively tries to use servers at each location to their maximum capabilities before adding a new server location. Total accumulated link bandwidth increases linearly with increasing demand per access node and increasing storage overhead. Furthermore, doubling the storage overhead roughly doubles the number of server locations. Both observations only hold as long as the access and aggregation network has sufficient bandwidth available. Note that for a read bandwidth demand of 50 Mbps per access node and a storage overhead $\frac{R}{b} = 3$, which implies 6 server locations per data item for $b = 2$, SSPA is forced to add multiple storage server locations at the access nodes, as the secondary ring bandwidth becomes the limiting factor.

[2] Note that in Fig. 5, the maximum server location read bandwidth is 1.2 Gbps.



(a) # servers (b) total link bandwidth

(c) maximum server load (d) minimum server load

Fig. 6. Number of server locations, total link bandwidth, maximum and minimum server location bandwidth for varying $\frac{R}{b}$ and increasing read bandwidth requirements per access node. A maximum delay of 4 hops is required. The number of data fragments per file $b = 2$ and server locations have a maximum read capacity of 1 Gbps.

8 Conclusion

The deployment of transparent personalized services requires the complete decoupling of devices for accessing the service and the supporting personal data storage. This paper studied the deployment of a personal content storage service, supporting created content, in the access and aggregation network, minimizing the number of server locations, while guaranteeing fast access from any access node. Both network restrictions—link bandwidth and maximum delay—and server restrictions—read and write performance, storage capacity and a maximum number of simultaneous users—are taken into account.

First it was assumed that servers do not fail and storing each data item once is sufficient. This problem was first formulated as a Binary Integer Linear Programming (BILP) problem. Next, a heuristic Storage Server Placement Algorithm (SSPA) was presented, installing servers one by one at the best candidate locations, until all users are served. Two typical access and aggregation network topologies were used for evaluating the algorithm: a mesh of trees, used in DSL deployment, and a ring of rings, used for cable Internet access deployment. Simulations, comparing SSPA's placement results to the optimal placement, achieved

using a CPLEX branch and bound implementation of the BILP formulation, showed the heuristic algorithm produces close to optimal results.

As high availability is an important feature of online personal content storage systems, while real-world servers do fail, both the BILP formulation and the SSPA heuristic were extended to support resilience and striping. The extended storage server placement problem was used to simulate some resilience and striping scenarios in the aforementioned ring of rings topology. As one would expect, simulation results showed that doubling the storage overhead roughly doubles the number of required server locations and the total accumulated link bandwidth, as long as the access and aggregation network has sufficient bandwidth available. For all use cases presented in this paper, simulation took less than a minute per parameter combination on modern PC hardware, showing that SSPA is very efficient for optimizing server placement in personal content storage deployments. The presented SSPA simulator was already used for evaluating a business case for deploying a digital media library service in the access and aggregation network of the Belgian DSL operator Belgacom [22].

Acknowledgment

This work is partially funded by the IBBT GBO project PeCMan.

References

1. Baya, V., Gauntt, J.: The Rise of Lifestyle Media: Achieving Success in the Digital Convergence Era. Technical report, PriceWaterHouseCoopers (2006)
2. Stevens, T., Vlaeminck, K., Van de Meerssche, W., De Turck, F., Dhoedt, B., Demeester, P.: Deployment of Service-Aware Access Networks through IPv6. In: 8th Int. Conf. on Telecommunications (2005)
3. Wauters, T., Coppens, J., Turck, F.D., Dhoedt, B., Demeester, P.: Replica Placement in Ring Based Content Delivery Networks. Journal of Computer Communications 29(16), 3313–3326 (2006)
4. Vlaeminck, K., De Turck, F., Dhoedt, B., Demeester, P.: Deploying Digital Media Libraries in Multi-Service Access Networks. In: 8th IEEE Int. Symposium on Multimedia, pp. 105–112 (2006)
5. Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.: Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. INFORMS Journal on Computing 12(1), 2–23 (2000)
6. Callaghan, B., Pawlowski, B., Staubach, P.: RFC1813: NFS version 3 protocol specification (June 1995)
7. Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J.: Scale and Performance in a Distributed File System. ACM Trans. Comput. Syst. 6(1), 51–81 (1988)
8. Mummert, L.B., Ebling, M.R., Satyanarayanan, M.: Exploiting Weak Connectivity for Mobile File Access. In: 15th ACM Symposium on Operating Systems Principles, pp. 143–155. ACM Press, New York (1995)
9. Cluster File System Inc.: Lustre: A Scalable, High-Performance File System. Technical report (November 2002)

10. Jones, T., Koniges, A., Yates, R.K.: Performance of the IBM General Parallel File System. In: IEEE Int. Parallel & Distributed Processing Symposium, May 2000, pp. 673–683 (2000)
11. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: 19th ACM Symposium on Operating Systems Principles, pp. 29–43. ACM Press, New York (2003)
12. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., Zhao, B.: OceanStore: an Architecture for Global-Scale Persistent Storage. SIGARCH Comput. Archit. News 28(5), 190–201 (2000)
13. Bolosky, W.J., Douceur, J.R., Ely, D., Theimer, M.: Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. SIGMETRICS Perform. Eval. Rev. 28(1), 34–43 (2000)
14. Saito, Y., Karamanolis, C., Karlsson, M., Mahalingam, M.: Taming Aggressive Replication in the Pangaea Wide-Area File System. SIGOPS Oper. Syst. Rev. 36(SI), 15–30 (2002)
15. Noble, B., Fleis, B., Kim, M.: A Case for Fluid Replication. In: Network Storage Symposium, Seattle, WA, USA (October 1999)
16. Qiu, L., Padmanabhan, V.N., Voelker, G.M.: On the Placement of Web Server Replicas. In: INFOCOM, pp. 1587–1596 (2001)
17. Casier, K., Verbrugge, S., Coller, D., Pickavet, M., Demeester, P.: Using Aspect-oriented Programming for Event-handling in a Telecom Research Softwave Library. In: 8th Int. Conf. on Software Reuse (2004)
18. ILOG: CPLEX. http://www.ilog.com/products/cplex/
19. Vitale, T.: Digital Imaging in Conservation: File Storage. AIC News 31(1) (2006)
20. Lin, W.K., Chiu, D.M., Lee, Y.B.: Erasue Code Replication Revisisted. In: 4th Int. Conf. on Peer-to-Peer Computing, pp. 90–97 (August 2004)
21. Pless, V.: Introduction to the Theory of Error-Correcting Codes, 3rd edn. Wiley, Chichester (1998)
22. Van Ooteghem, J., Vlaeminck, K., Colle, D., Pickavet, M., De Turck, F., Dhoedt, B., Demeester, P.: Business Case for Deploying Digital Media Libraries in Multi-Service Access Networks. In: 2007 Int. Conf. on Business and Information, Tokyo, Japan (July 2007)

Extraction and Classification of User Behavior

Matheus L. dos Santos[1], Rodrigo F. de Mello[1], and Laurence T. Yang[2]

[1] University of São Paulo, Institute of Mathematics and Computer Science
São Carlos, SP, Brazil
{matheusl,mello}@icmc.usp.br
[2] St. Francis Xavier University, Antigonish, NS, Canada
lyang@stfx.ca

Abstract. The multimedia document generator, iClass system, has been used by professors from the Institutes of Chemistry, Mathematics and Computer Science from the University of São Paulo aiming at helping the multimedia content production and availability. Data from user interactions, available in iClass system, have motivated this work which aims at studying the user behavior under different circumstances. The behavior extracted makes possible the analysis of different patterns of the same user, among groups, and distinct users. Those pattern differences should help to understand user evolution in iClass system under diverse situations. The data are grouped by a neural network and afterwards Markov Chains are built to represent their behaviors in different time moments. The detected user or group behavior variations are related to classify profiles and comprehend them in different situations.

1 Introduction

The iClass system, developed by the Intermídia Laboratory at University of São Paulo, captures information from conventional environments (such as classrooms), allowing the production of multimedia documents which, afterwards, are delivered through the Web [1]. This system has been adopted to help professors during classes.

During the classes, students make experiments, works and tests using iClass system. The iClass works as a *whiteboard*, generating databases containing user interaction information. Such information is stored in XML[1] documents, which stores the interaction instant and the number of points defined by users. Words and pictures are composed in groups of points, named strokes.

The availability of such information motivates the study of user behaviors under different circumstances (experiments, tests, classes, etc). The extracted behavior makes possible the analysis of different user patterns, among groups and distinct users. Such pattern differences help to understand user evolution in the iClass system under diverse situations.

This behavior study is conducted applying statistics techniques (like Markov Chains [2, 3]), classification by neural networks (SOM [4, 5, 6], ART [7]) and

[1] Extensible Markup Language – http://www.w3.org/XML.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 493–506, 2007.

detection of changes in pattern behavior by measuring energy variation (entropy [8] [9] [10] [11]). The statistics techniques help to detect differences in the interaction intervals. Users at low interaction intervals should demonstrate more familiarity with the tool or with the subject covered in classes. Neural networks assist the behavior classification of user interactions, resulting in groups (or *clusters*), which represent interaction states such as: low, medium and high interaction. The variation between such states help to understand user behaviors.

This paper proposes a model to identify and classify user behavior patterns to allow comparisons in different moments and understand their interactions in computational systems.

This paper is organized as follows: section 2 presents related work; The methodology used to classify user behavior is presented in section 3; section 4 compares two users interacting in the iClass system; Conclusions and considerations are presented in section 5; The plans for future works are described in section 6.

2 Related Work

Works considering methods of user behavior classification are found in literature. The search for information under the human behavior is a concern of diverse research areas such as [12] [13] [14] [15] [16] [17].

Brosso [12] considers an user authentication system in computer networks, which uses behavior analysis and face recognition to obtain the confidence degree to identify a system user. By analyzing the user behavior, the author uses the concept of Context-aware Computing which is based in the study of applications that adapt to locality and changes that occur to people and objects during time [18]. Besides context-aware computing, this work also adopts the five semantic dimensions defined by Abowd [19] [20], which are used to specify and model context information. Those dimensions characterize the relevance of information (*Who*, *Where*, *What*, *When*, *Why*). In this way, to analyze user behavior, matrix is defined which contains information based in context-aware computing. Such information allows to understand user actions, locality, the period of interaction, if some behavior is an habit (*why*) and the confidence restrictions effected by user.

Godoy and Amandi [13] consider a technique to generate user profiles by observing their interaction characteristics on the Web. This technique is inserted in the algorithm *Web Document Conceptual Clustering* [21], which allows to acquire profiles without having any previous knowledge about user interests. The profile of user interests is organized in an hierarchical tree, where at the highest level are the general interests and at the lowest, the most particular ones. Those interests can be any information accessed by the user such as: sports, works, news and games. The relevance degree of the user interests is measured by means of term frequency, according to the document access rate.

Lee *et al.* [14] consider a new load balancing policy for Web servers named PRORD (*Proactive Request Distribution*). PRORD preloads the most accessed

web pages, based on the probability of future accesses, for this, the system analyzes information about web server caches. Using such preloading scheme, the web server anticipates pages with high access probability, decreasing the response time and improving the efficiency.

Macedo *et al.* [15] propose a system, named WebMemex, which recommends information to users by analyzing the navigation description of known users. The WebMemex in conjunction with a proxy web server, which provides user history requests to WebMemex. In this way, WebMemex captures information such as IP addresses, user IDs, user active time in system and the URL accessed. Such information is stored in a relational database for future analysis.

Pepyne *et al.* [16] propose a method to classify user profiles using queue theory and logistic regression. This work explores system application in computer network security. The objective is to identify profiles of specialized user groups, such as bank tellers, which execute periodic computer tasks, from where it is possible to detect frauds using anomaly analysis of user behaviors.

Schuler and Perez [17] apply data mining techniques to discover user profiles in telecommunication systems. Two techniques of data mining are used: decision trees and neural networks. The rules generated by the decision tree represent the general profile of default users. Having the defined tree, historical data can be compared to any user to verify his/her relation to the determined class. Authors have concluded that decision trees represent user pattern/profile behavior, but present a great number of sub-groups becoming the comprehension impracticable.

As previously studied, the majority of works uses classification methods that, in some way, depends on the system or application goal. Such methods take into consideration the semantics of the analyzed data, and therefore they cannot easily be applied to other systems. Considering such problem, the work presented in this paper is motivated to develop an extraction and classification model to detect user behavior in any system or application.

3 The Model

This paper proposes a model to identify and classify user behavior patterns to allow comparisons in different moments and understand their interactions in computational systems. The model proposed in this paper is based on the study and evaluation of user interaction datasets; classification using artificial neural networks (ANNs); representation of user profiles using Markov chains; measure the energy variation to represent user behaviors; analyze user behavior patterns, comparing to other users and groups.

The first step of this work created probability distribution functions (PDFs) to represent the user interactions. Consider the table 1 as an example of user interaction which contains the data available in iClass. The data represent user interactions to the system. Each interaction has a timestamp and the quantity of points (geometric forms or writing) made by an user.

Table 1. Example of user interaction in the iClass system

Time	Points	Object
12:03	1500	geometric shapes
12:05	200	geometric shapes
12:07	400	writing
12:08	200	writing
12:10	400	writing
12:11	200	writing
12:14	900	geometric shapes
12:17	300	writing
12:19	400	geometric shapes
12:20	50	writing
12:22	400	writing
12:24	400	geometric shapes
12:25	200	writing
12:26	200	writing
12:28	1200	writing
12:31	900	geometric shapes
12:35	1600	geometric shapes
12:38	900	writing
12:39	200	writing
12:40	400	geometric shapes

Table 2. Representation of user interaction in time intervals

Time Intervals	Points
12:00 – 12:03	1500
12:03 – 12:05	200
12:05 – 12:07	400
12:07 – 12:08	200
12:08 – 12:10	400
12:10 – 12:11	200
12:11 – 12:14	900
12:14 – 12:17	300
12:17 – 12:19	400
12:19 – 12:20	50
12:20 – 12:22	400
12:22 – 12:24	400
12:24 – 12:25	200
12:25 – 12:26	200
12:26 – 12:28	1200
12:28 – 12:31	900
12:31 – 12:35	1600
12:35 – 12:38	900
12:38 – 12:39	200
12:39 – 12:40	400

Table 3. Example of the probability distribution on the user interaction

Interval	Points	Interval	Points
00 – 01	500	20 – 21	200
01 – 02	500	21 – 22	200
02 – 03	500	22 – 23	200
03 – 04	100	23 – 24	200
04 – 05	100	24 – 25	200
05 – 06	200	25 – 26	200
06 – 07	200	26 – 27	600
07 – 08	200	27 – 28	600
08 – 09	200	28 – 29	300
09 – 10	200	29 – 30	300
10 – 11	200	30 – 31	300
11 – 12	300	31 – 32	400
12 – 13	300	32 – 33	400
13 – 14	300	33 – 34	400
14 – 15	100	34 – 35	400
15 – 16	100	35 – 36	300
16 – 17	100	36 – 37	300
17 – 18	200	37 – 38	300
18 – 19	200	38 – 39	200
19 – 20	50	39 – 40	100

In order to understand how the PDF should be useful, a better representation of the table 1 is presented in the table 2. For this last table, the number of points in time intervals can be observed. For instance: in the interval of 12 : 00 to 12 : 03 the user has produced 2000 points, this is, 2000 points in 3 minutes.

The data from the table 2 can be distributed in constant time intervals to simplify the PDF generation. The table 3 presents interaction data distributed in discrete time intervals of 1 minute, where the number of points p for interval is given by $p = \frac{np}{i}$ being np the quantity of points in the interval i. To better illustrate the construction of the table 3, consider the time interval between 12 : 03 and 12 : 05 (table 2), which has 2010 points in 2 minutes, in this case the time intervals must be divided in two periods of 1 minute: one from 12 : 03 to 12 : 04 and another from 12 : 04 to 12 : 05, with the number of points in each new same-sized interval to 1005 ($\frac{2010}{2} = 1005$) points per minute.

The PDF from the table 3 is presented in the figure 1. In this case, there is a large user interaction variation to the system.

In the second stage, the data densities of the PDF were classified by an ANN. Such classification was made submitting input vectors (table 3) in the form $\frac{np}{i}$ where np is the number of points and i is the time interval, for example: $[\frac{28}{1m}, \frac{10}{0.5m}, ..., \frac{50}{3m}]$. In this stage the ANN has created groups (*clusters*) in accordance with the inputs. When a pattern does not fit in an existing group, a new one is created to receive it. The figure 2 represents the interaction sequence of the ANN, illustrating the creation of groups.

In the third stage, having the groups, Markov Chains are built to represent user profiles. The figure 3 presents an example of a Markov Chain, where the circles represent states (groups) and the arcs indicate transitions in accordance with their respective probabilities.

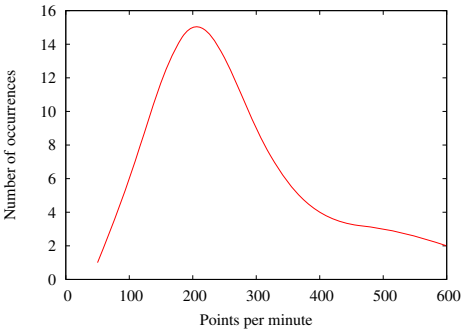


Fig. 1. Graphical representation of the probability distribution of user interaction

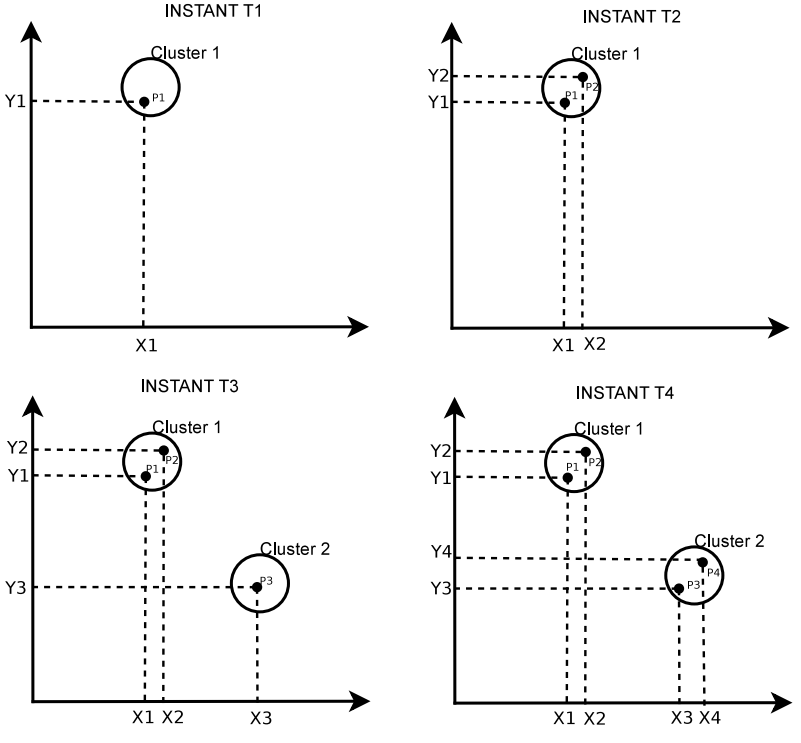


Fig. 2. ANN classifying patterns

As previously observed, the ANN creates a new group when a pattern does not correspond to a group. The creation of a new group indicates the occurrence of an unexpected system pattern. Groups, hardly ever visited, also depict unexpected behavior. To detect such new features [22], the system energy metric (entropy) is used.

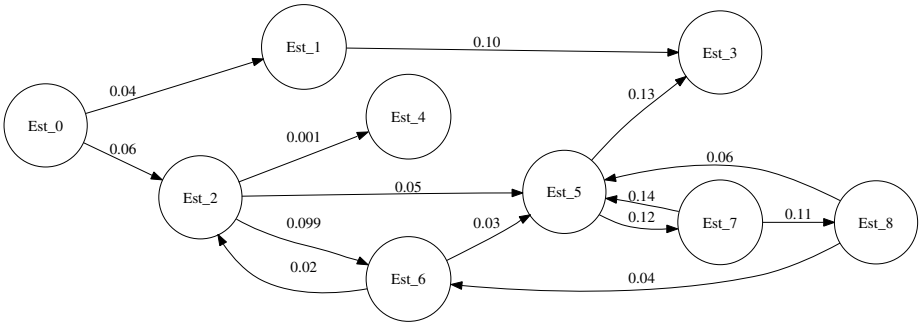


Fig. 3. Example of a Markov Chain representing a user profile in a time instant

The figure 4 illustrates as the energy behaves at every instant of a Markov Chain. At the instant t_0, as the probabilities to change from a state are equal, there is almost no behavior variation, therefore the energy E_0 is low. At the instant t_1, a system variation is caused by the creation of an unexpected state Est_2, which increases the variation in the probabilities of state change and, consequently, the energy goes up to $E_1 = 0.92637354$ (the energy variation is equal to $\Delta E = 0.23322636$).

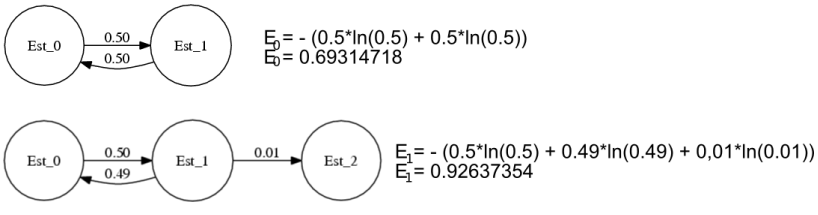


Fig. 4. Example of energy measurements in the Markov Chain

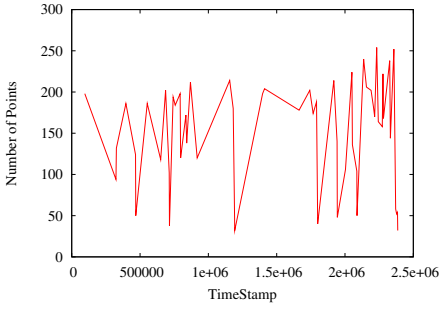
In the fourth stage user behaviors are analyzed. The evaluation of behavior differences are carried out comparing the group labeling obtained by the ANN, joining to the Markov chain of the user at the instant t_0 and t_1. The labeling is represented by vectors which summarize the characteristics of the classified patterns in a certain group of the ANN. Those vectors are in the form $r = [r_0, r_1, ..., r_{n-1}]$, where n is the number of input data and r is the relevance of the input. To allow the comparisons among groups, it is necessary to normalize vectors $r \in [0, 1]$ dividing each element for the summing up of the elements $(\frac{r_{n-1}}{r_0+r_1+...+r_{n-1}})$.

The behavior of an user is compared in each instant, this is, the instant t_0 is compared (using the labeling vectors) to t_1, t_2, and so forth, to detect behavior changes in user interactions. Different users can also have their behavior compared. That comparison is usually made considering the behavior at the same time instant t_k.

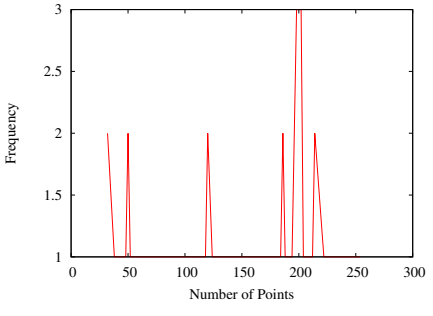
4 Experiments and Results

Experiments were conducted to evaluate the proposed model. In iClass, the information about the user interaction is stored, in a XML file named "session.xml". In this file exists several *tags* which contain information about user interaction

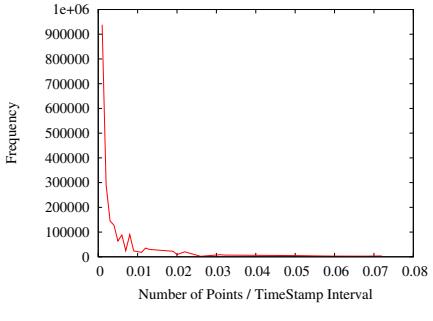
Fig. 5. User playing the Sudoku game in the iClass system



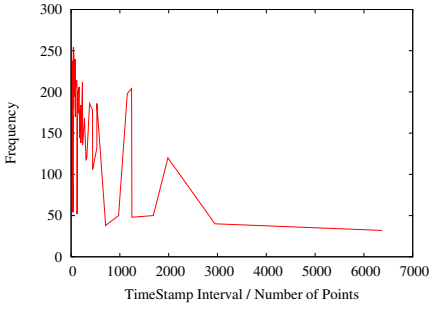
(a) Number of points in the time interval



(b) Frequency of the number of points



(c) Frequency of the number of points per second



(d) Frequency of the elapsed time per point

Fig. 6. Example of data distributions on the user interactions in the Sudoku game

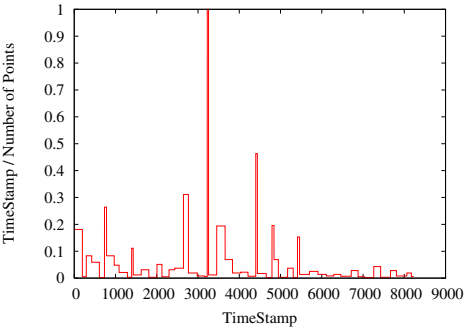
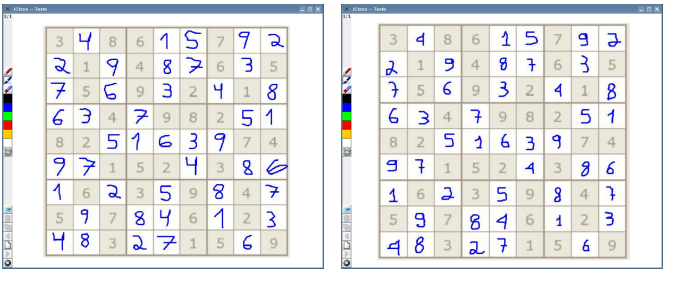
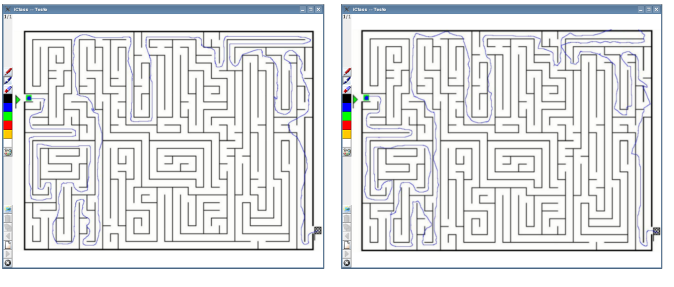


Fig. 7. Distribution representing the elapsed time per point throughout the user interaction



(a) User 1 - Sudoku (b) User 2 - Sudoku

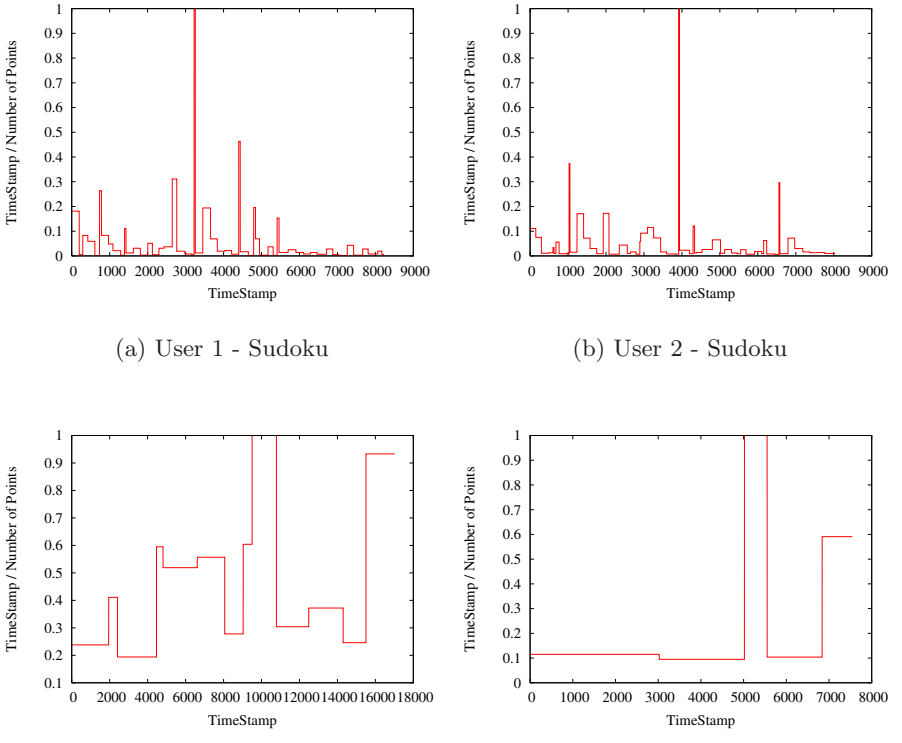


(c) User 1 - Maze (d) User 2 - Maze

Fig. 8. Two users interacting to the iClass System

such as: user name, page resolution, pen color, *timestamp* of *stroke*, number of points per *stroke* and others.

A parser using the Java language was developed to extract information about user interactions from the file "session.xml". From these information (number of points and *timestamps* of each *stroke*), probability distribution functions were generated to analyze data.



(a) User 1 - Sudoku

(b) User 2 - Sudoku

(c) User 1 - Maze

(d) User 2 - Maze

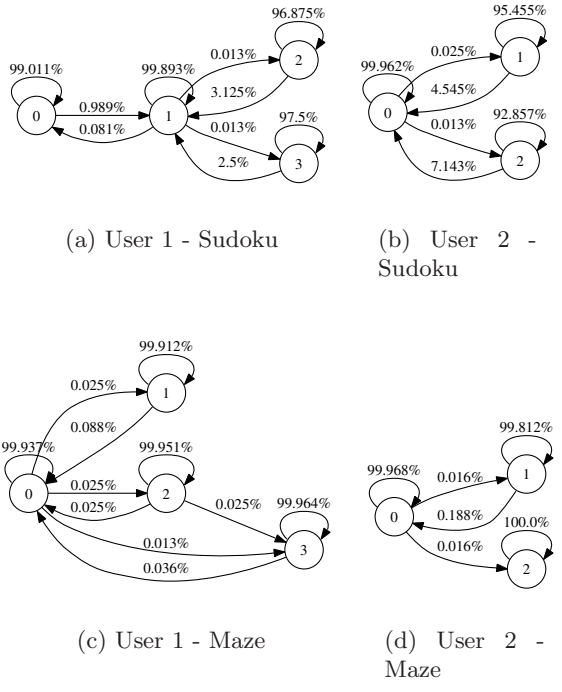
Fig. 9. Interaction data probability distribution of users in the iClass system

Initially, user interactions generated during sessions of the Sudoku[2] game were captured (figure 5). The game took place in the iClass system. From such data, some probability distributions were created to characterize and understand user behavior (figure 6).

The figure 6(a) presents a direct data distribution, this is, the data extracted from the file "session.xml" without modifications. It is difficult to identify some characteristics by using such figure, for instance, the elapsed time in each interaction (user knowledge about the process) and the time interval between two interactions (thinking time). In the same way, the data presented in the figures 6(b), 6(c) and 6(d) do not directly represent user behavior, therefore the data in these charts are grouped according frequencies, and in this way, information as the thinking time and the user knowledge are mixed, making difficult their identification.

The found solution was to represent the distribution demonstrated in the figure 6(d) in a time discrete manner, in this way, the data on the elapsed time

[2] http://en.wikipedia.org/wiki/Sodoku



(a) User 1 - Sudoku

(b) User 2 - Sudoku

(c) User 1 - Maze

(d) User 2 - Maze

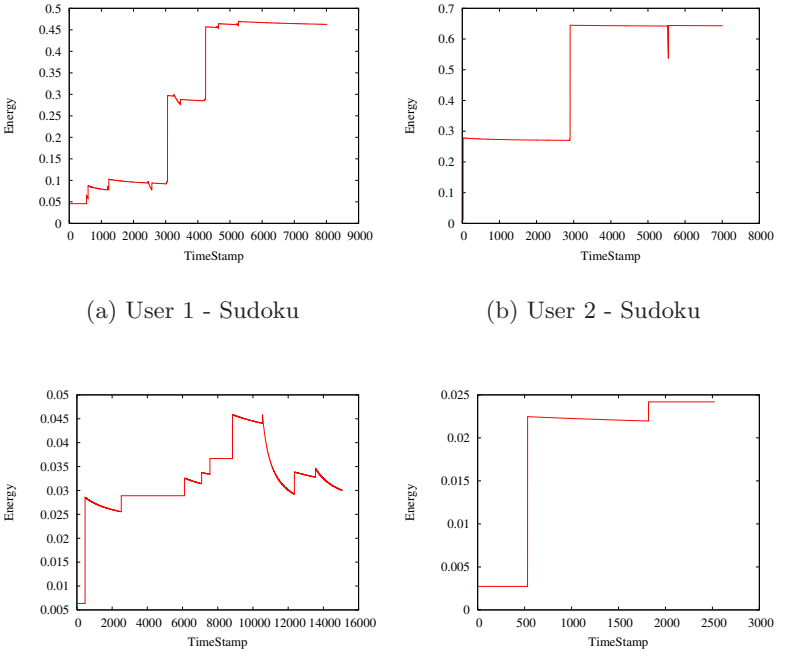
Fig. 10. Markov chains at the last user interaction in the iClass system

in each user interaction is represented at the same time interval, making possible the direct comparison between the chart and the user interaction. The figure 7 represents this new data distribution, where can be visualized the time intervals in which the user interacts to the system (Sudoku game in such case). Each unevenness presented in the charts depicts an user action in the game.

However, any distribution will present a deficit in the data representation, not separating the interaction time from the time between interactions. This occurs because the way the information is stored by iClass. The extracted information from iClass is in the form: *timestamp* for the number of points drew by users. In this way, to obtain the interaction time of one *stroke*, the *timestamp* of the next has to be deducted from the current one. The stroke interaction time represents the moment of user interaction, the idle time between interactions is common as the user make an action and later spends a time until the next one initiates.

After having defined the data distribution to represent user behavior, the experiments were conducted to evaluate the interaction of two different users. Those users had interacted, by means of the iClass system, to the Sudoku game and had solved a maze problem. The figure 8 represents the end of the interaction carried out by User 1 and User 2.

Afterwards, each user interaction behavior is represented by the data distribution extracted from the file “session.xml”. The data distributions of interaction of each user is demonstrated in the figure 9, where the behavior of the User 1



(a) User 1 - Sudoku

(b) User 2 - Sudoku

(c) User 1 - Maze

(d) User 2 - Maze

Fig. 11. Variation in the user pattern behavior in the iClass system

interacting to the Sudoku game and the maze is respectively represented by the figure 9(a) and figure 9(c) and, the behavior of the User 2 is represented by the figure 9(b) and 9(d), respectively.

The next steps, following the model in the section 3, is: to carry out the data classification, building the Markov chains and later measuring the average energy variation between the two chains (entropy). Using the neural network SONDE [23], the data distributions represented in the figure 9 were classified. As the main purpose of SONDE is to detect new features (novelties) by analyzing energy variations in datasets, the Markov chains and the energy variation chart were automatically generated. The figure 10 represents only the Markov chains at the last user interaction in the Sodoku and in the Maze (remember each time instant is represented by a Markov chain).

After generating the Markov chains, the user behavior variation is represented by the energy variation between Markov Chains. This behavior variation is represented in the figure 11.

By analyzing the figure 11, User 1 keeps some characteristics in the two interactions (Sudoku and Maze), what also occurs to User 2. In the figure 11(a), some stable points in the user behavior (declivity) are observed, this characteristic also is observed in the figure 11(c). A detailed observation of the User 2 allows to notice a common pattern among the interactions. In the figure 11(b),

the energy level is increasing, presenting steps, and the same occurs in the figure 11(d). By this, we confirm that User 2 contains a bigger dynamism in his/her actions, not having pauses throughout interactions. User 1 also presents a level of increasing energy, although, differently from User 2, the User 1 probably present pauses during its interactions. Maybe such pauses are related to the thinking time throughout interactions, in contrast of User 2 that thinks about the problem before starting deciding.

5 Conclusion

This paper proposes a model to extract data from user interactions to detect and classify user behavior patterns. The model summarizes user interactions through Markov Chains, and the user behavior profile is represented by a set of Markov Chains. The energy variation (entropy) between Markov Chains represents the user behavior variation along interactions.

Experiments were conducted to evaluate the proposed model in the iClass system. The obtained results show that the model detects users behavior profiles interacting in the iClass system. Even interacting in distinct problems, users may present similar behavior characteristics between experiments. It is important to notice that such results are preliminary and are not conclusive.

With the manipulation and analysis of user interaction data, we may establish relations among different behaviors and profiles, being possible to relate an user or group actions in classrooms to their test performances or still to identify an user by analyzing interactions.

Using techniques of energy measurement (entropy) in a system, it is possible to detect user behavior changes, relate them to unexpected events such as some special fact occurred in classroom, personal or medical problems and others.

Besides establishing user behavior patterns, it is also possible to anticipate user actions. This technique can assist in the artificial intelligence of games, Web sites recognizing users from interactions.

6 Future Work

The results presented in this paper were obtained by experiments conducted with two users. In order to give statistical relevance such experiments will be executed in a larger population. Besides the experiments, techniques of validation and comparison of user behavior patterns have been studied.

Acknowledgments

This paper is based upon work supported by CAPES, Brazil under grant no. 032506-6 and FAPESP, Brazil. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the CAPES or FAPESP.

References

1. Cattelan, R.G., Andrade, A.R., Rocha, C.F.P., Pimentel, M.d.G.C.: iclass: um sistema para captura e acesso de sessões em ambiente educacional. Revista Eletrônica de Iniciação Científica - REIC 3(1), 10–28 (2003)
2. Grinstead, C.M., Snell, J.L.: Introduction to Probability. American Mathematical Society 2nd Rev edn. (July 1, 1997), United States of America (1997)
3. Nogueira, F.: Cadeias de markov. (02, 2006), http://www.engprod.ufjf.br/fernando/epd042/cadeiaMarkov.pdf
4. Kaski, S., Oja, E.: Kohonen Maps. Elsevier Science Inc., New York (1999)
5. Kohonen, T., Kaski, S., Lagus, K., Salojrvi, J., Honkela, J., Paatero, V., Saarela, A.: Self organization of a massive document collection (2000)
6. Makhfi, P.: Competitive learning (2006), http://www.makhfi.com/tutorial/clearn.htm
7. Carpenter, G.A., Grossberg, S.: The ART of adaptive pattern recognition by a self-organizing neural network. Computer 21(3), 77–88 (1988)
8. Shannon, C.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423 and 623–656 (1948)
9. Santos, E.M.d.S., Albuquerque, M.P., Mello, A.d.R.G., Caner, E.S., Esquef, I.A.: Fundamentos da teoria da informaçào. Technical report, Centro Brasileiro de Pesquisas Físicas (Dezembro 2004)
10. Boltzmann, L.: Vorlesungen uber Gastheorie. Volume 1, 2. J. A. Barth Leipzig (1896) English Translation by S.G. Brush: Lecture on Gas Theory, Cambridge Univ. Press, Cambridge (1964)
11. Freeman, J.A., Skapura, D.M.: Neural networks: algorithms, applications, and programming techniques. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (1991)
12. Brosso, M.I.L.: Autenticação Contínua de Usuários em Redes de Computadores. Tese de doutorado, Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil (2006)
13. Godoy, D., Amandi, A.: User profiling for web page filtering. IEEE Internet Computing 9(4), 56–64 (2005)
14. Lee, H.K., Vageesan, G., Yum, K.H., Kim, E.J.: A proactive request distribution (prord) using web log mining in a cluster-based web server. In: ICPP 2006: Proceedings of the 2006 International Conference on Parallel Processing, pp. 559–568. IEEE Computer Society, Washington, DC, USA (2006)
15. Macedo, A.A., Truong, K.N., Camacho-Guerrero, J.A., da Graça Pimentel, M.: Automatically sharing web experiences through a hyperdocument recommender system. In: HYPERTEXT 2003: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, pp. 48–56. ACM Press, New York (2003)
16. Pepyne, D.L., Hu, J., Gong, W.: User profiling for computer security. American Control Conference, 2004. Proceedings of the 2004 2(6), 982–987 (2004)
17. Schuler, A.J.J., Perez, A.L.F.: Análise do perfil do usuário de serviços de telefonia utilizando técnicas de mineração de dados. RESI - Revista Eletrônica de Sistemas de Informação 7(1) (2006)
18. Schilit, B., Theimer, M.: Disseminating active map information to mobile hosts. IEEE Network 8(5), 22–32 (1994)
19. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)

20. Abowd, G.D., Mynatt, E.D.: Charting past, present, and future research in ubiquitous computing. ACM Trans. Comput.-Hum. Interact. 7(1), 29–58 (2000)
21. Godoy, D., Amandi, A.: Modeling user interests by conceptual clustering. Inf. Syst. 31(4), 247–265 (2006)
22. Markou, M., Singh, S.: Novelty detection: a review – part 2: neural network based approaches. Signal Process. 83(12), 2499–2521 (2003)
23. Albertini, M.K., Mello, R.F.: A self-organizing neural network for detecting novelties. In: ACM Symposium on Applied Computing (2007)

A Floorplan-Based Power Network Analysis Methodology for System-on-Chip Designs

Shih-Hsu Huang[1], Chu-Liao Wang[1], and Man-Lin Huang[2]

[1] Department of Electronic Engineering
Chung Yuan Christian University
Chung Li, Taiwan, R.O.C.
{shhuang,g8976035}@cycu.edu.tw
[2] Office of Information Technology
Feng-Chia University
Tai Chung, Taiwan, R.O.C.
mlhuang@fcu.edu.tw

Abstract. In order to enable the single-pass design methodology, the planning of power distribution should be performed as early as possible. In this paper, we will tackle this problem at the floorplan stage. First, at the block level, we will present an effective method to model the behavior of local power network structure of a reused block. Next, at the full-chip level, we will present a floorplan-based power network analysis methodology for system-on-chip (SOC) designs. The proposed methodology works well because it uses suitable models to represent the local power networks of blocks according to the properties of blocks. Experimental data shows that the new modeling technique can identify the most critical drop voltage of a reused block and the floorplan-based analysis methodology is useful for the planning of power distribution network of a SOC design.

Keywords: Modeling, Voltage Drop, Power Consumption, and Reused Block.

1 Introduction

Power distribution is always very important in the design of VLSIs because power network covers a large portion of chip area. So it is given the first priority in routing process [1][2]. There are two basic problems in the design of power network. The first is the undesirable wear-out of metal wiring caused by electromigration, and the second is the narrowing margins causes by voltage drops. Increasing wire widths can solve these problems. But it is too expensive to use the wiring resources freely.

In the deep submicron technology, the metal width tends to decrease with the length increasing due to the complex system integration into single silicon. Therefore, the resistance along the power metal line increases. Chips that are under-designed with a smaller margin often fail on the test bench or later in the field. However, most of the commercially available tools focus on the transistor-level and post-layout verification

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 507–516, 2007.

of power distribution. If any problem related to electromigration or voltage drop is revealed at this stage, it is very difficult or expensive to fix. In order to enable the single-pass design methodology, the accurate planning of power distribution should be performed as early as possible. In this paper, we will present a floorplan-based power network analysis methodology for system-on-chip (SOC) designs.

At the floorplan stage, the chip area is divided into a set of non-overlapping physical blocks by power trunks. Each physical block consists of many rows. Cells are placed on the row so that the power is fed from the power rail. The two end points of a power rail are the power trunks. Because of consistency between cell library and power rail, and wiring resource economy, the line width of power rail may not be increased. In other words, the local power network structure in a physical block is often fixed. However, with the increase of chip size, power supply current for more than a hundred cells is to be fed through this thin rail. The power trunks are necessarily used to improve the reliability and quality of power distribution. Therefore, the design and analysis of power network is important for SOC designs.

The main distinctions of our work are elaborated as the below:

- At the block level, we will propose an effective method to model the local power network structure of a reused block, whose power consumption was already measured by real instruments. As a result, when we integrate the reused block into a SOC design an accurate equivalent resistive circuit can be used to describe the behavior of local power network structure.
- At the full-chip level, we will propose an effective methodology to analyze the power distribution of a SOC design. As a result, the designer can accurately predict the electromigration or voltage drop problem at the floorplan stage. If any reliability problem is found, the designer can make power network changes when they are easiest and least costly to implement.

The rest of the paper is organized as the below. Section 2 briefly introduces the problem and surveys the related works. In Section 3, we will present a new modeling method, which is well suitable to model the behavior of local power network structure of a reused block. Next, a floorplan-based full-chip power network analysis methodology will be proposed in Section 4. Some experimental results will be reported in Section 5. Finally, we will give some concluding remarks in Section 6.

2 Preliminaries

At the stage of floorplan, the power trunks form a global power network and divide the full-chip into several blocks. Let's use Figure 1 as an example. Figure 1 (a) is a floorplan, whose blocks are divided by global power network. Figure 1 (b) illustrates an equivalent circuit of its power distribution.

At the block level, where local hot spots are located, a finer grid will be generated to model the local power network structure. The power grids may be defined as below. Each physical block consists of many rows. On each row, the power rails of adjacent cells are connected together. A straightforward method is that each cell in the row is represented as a grid.

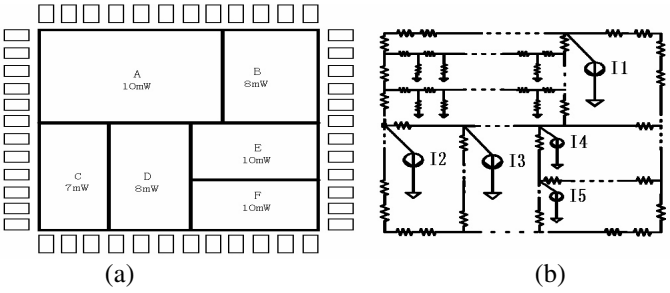


Fig. 1. (a) A floorplan. (b) An example of equivalent circuit of its power network.

Most previous works use the equivalent current source model [3][4] to describe the local power network structure of a block at the floorplan stage. The assumptions of the equivalent current source model are elaborated as below.

- All the switching activities within one block are lumped together.
- The current sources are uniformly distributed at the grids.

The equivalent current source model is useful for modeling blocks at floorplan view, where placement and route have not been done. The basic technique of equivalent current source model is described as the below. Suppose that the power consumption of a block, which is usually reported by the dynamic gate-level power calculator, equals to P_{block}. Note that, in order to tackle the worst case, P_{block} is usually the peak power consumption. Then, for this block, the equivalent current source Iblock is equivalent to P_{block}/V_{DD}. Assumes that this block consists of m rows and each row has n grids. Thus, for each gird in this block, the equivalent current source Igrid is Iblock/(m*n). As a result, for each grid in this block, the resistance Rgrid equals to VDD/Igrid. The Rp is equivalent to Rs * row length / power rail width, where Rs is unit resistor. The method to calculate Rring is similar to RP.

3 The Modeling Technique for Reused Blocks

In this section, we will present an effective modeling technique for the local power network structure of a reused block, whose power consumption was already measured by real instruments. Our motivation is shown in Section 3.1. Then, our method is given in Section 3.2.

3.1 The Motivation

First, we borrow the material from [5] to describe the equivalent current source modeling technique as below. In the equivalent current source modeling technique, the value of equivalent current source is obtained without considering the resistances in the power rails. As a result, when the metal resistances in the power rails are added into the equivalent circuit, the power consumption will be varied. In other words, the power consumption of the equivalent resistive circuit derived by this model will be different from the original given power consumption.

Let's use the unit placement row shown in Figure 2 as an example. A unit placement row is a resistive network whose end points are power trunks. The unit placement row in Figure 2 consists of the resistors R_{P1}, R_{P2}, R_{P3}, R_{P4}, R_{P5}, R_{C1}, R_{C2}, R_{C3}, and R_{C4}. The values of R_{P1}, R_{P2}, R_{P3}, R_{P4}, and R_{P5} are extracted from technology parameters. Assume that P_{row} is the given power consumption of the unit placement row. Then, if using the equivalent current source model, the values of R_{C1}, R_{C2}, R_{C3}, and R_{C4} are obtained by the following equation:

$$Rci = (4*V_{DD}^2)/P_{row} \text{ , where } i=1,2,3,4$$

Let P_{RP1}, P_{RP2}, P_{RP3}, P_{RP4} and P_{RP5} be the power dissipation of resistors R_{P1}, R_{P2}, R_{P3}, R_{P4}, and R_{P5} respectively. From [5], the power consumptions P_{RP1}, P_{RP2}, P_{RP3}, P_{RP4} and P_{RP5} can be calculated as follows.

$P_{RP1} = (V_{S1} - V_{M1})2 / R_{P1}$
$P_{RP2} = (V_{M1} - V_{M2})2 / R_{P2}$
$P_{RP3} = (V_{M2} - V_{M3})2 / R_{P3}$
$P_{RP4} = (V_{M3} - V_{M4})2 / R_{P4}$
$P_{RP5} = (V_{M4} - V_{S2})2 / R_{P5}$

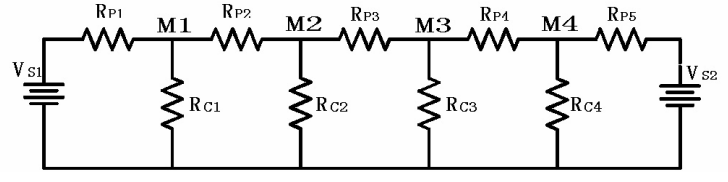


Fig. 2. An example of unit placement row

Let P_{RC1}, P_{RC2}, P_{RC3} and P_{RC4} be the power dissipation of resistors R_{C1}, R_{C2}, R_{C3}, and R_{C4} respectively. The power consumptions P_{RC1}, P_{RC2}, P_{RC3} and P_{RC4} can be calculated as follows.

$P_{RC1} = V_{M1}^2 / R_{C1}$
$P_{RC2} = V_{M2}^2 / R_{C2}$
$P_{RC3} = V_{M3}^2 / R_{C3}$
$PRC4 = V_{M4}^2 / R_{C4}$

Therefore, the total power consumption P_{total} of this resistive network is given by the following equation:

$$P_{total} = P_{RP1} + P_{RP2} + P_{RP3} + P_{RP4} + P_{RP5} + P_{RC1} + P_{RC2} + P_{RC3} + P_{RC4}.$$

It is obvious that P_{total} is not the same as P_{row}, which is the original given power consumption. In other words, although we use P_{row} to obtain the equivalent current source, the power consumption of the derived resistive circuit is not equivalent to P_{row}. However, the P_{row} may also often be over-estimated, because dynamic gate-level power calculator does not tackle metal resistances. Therefore, although there is a difference

between P_{row} and P_{total}, the equivalent current source model still works well in specifying the power grid requirements for typical ASIC design flow.

However, due to the trend of SOC design methodology, a block may be reused from a previous production chip. In such case, real instruments can measure the power consumption of this reused block. Once a block is measured, we need to be able to reuse the accurate measured result, rather than having to re-estimate for the block. Therefore, it is desirable to have a power network modeling technique such that the P_{total} equals to P_{row}. To achieve the goal, we need to have a different method to model the behavior of local power network structure.

3.2 The Proposed Method

The problem we study is described as the below. Given a measured power consumption of a reused block, find the value of equivalent grid resistance such that the power consumption of the derived resistive circuit is identical. With the same assumption as [3][4], we suppose that all grid resistances are the same. Let's use Figure 2 as an example. Suppose the reused block has m rows and the measured power consumption is P_{block}. Then, the P_{row} is P_{block}/m. The values of R_{P1}, R_{P2}, R_{P3}, R_{P4}, and R_{P5} can be extracted from technology parameters. Thus, our problem is to find the value of grid resistance RC such that $R_C = R_{C1} = R_{C2} = R_{C3} = R_{C4}$ and $P_{row} = P_{total}$.

It is difficult to find the correct R_C directly. Our method is to iteratively solve the linear resistive network and calculate the P_{total} in each time of iteration. The initial value of R_C is the same as the equivalent current source model. In other words, initially, $R_C = (n*V_{DD}^2)/P_{row}$, where n is the number of grids in a row.

```
Procedure Find_Grid_Resistance();
begin
R_C = (n*V_DD^2)/P_row;
solve the resistive network based on R_C;
while (|P_row-P_total|>tolerance) do
 begin
  if (P_row>P_total)
 then decrease R_C;
 else increase R_C;
 solve the resistive network based on R_C;
 end
end.
```

Fig. 3. The algorithm to find the grid resistance R_C

Note that the P_{total} is calculated in each time when the linear resistive network is solved. If P_{total} is smaller than P_{row}, R_C is decreased by a small value in the next time. If P_{total} is greater than P_{row}, R_C is increased by a small value in the next time. The iterations repeat until the difference between P_{row} and P_{total} are confined within a tolerance value. As a result, the value of R_C will be converged. The pseudo code of our method is given in Figure 3.

4 The Full-Chip Analysis Methodology

A SOC design often contains several blocks. In this section, we will present a floorplan-based full-chip power network analysis methodology, which is applicable to the design flow of a SOC design.

4.1 Full-Chip Equivalent Resistive Circuit

The first step is to generate an equivalent resistive circuit to represent the full-chip power distribution. The algorithm is depicted in Figure 3. For the need of further analysis, the output is in SPICE format. The details of the algorithm are as below.

The algorithm translates all the power pads, power trunks, all the physical blocks into corresponding equivalent circuits, respectively. Firstly, the power pads are tackled. For each power pad, the supplied voltage value is set on its corresponding grid position. Then, for each power trunk, it is translated into a series of resistance Rtrunk, where Rtrunk is obtained as below:

(1) For a vertical power trunk, R_{trunk} = Rs * cell height / power trunk width.
(2) For a horizontal power trunk, R_{trunk} = Rs * cell width / power trunk width.

Next, the local power network of each physical block is translated into equivalent resistive circuit according to the property of this block. For the blocks, whose power consumptions are reported by dynamic gate-level power simulator, their local power networks are generated into SPICE netlist using equivalent current source model. For the blocks, whose power consumptions are measured by real instruments, their local power networks are generated into SPICE netlist using the model presented in Section 3.2.

```
Procedure Generate_Full_Chip_Netlist();
begin
for each power pad do
 set the voltage value on the corresponding grid;
for each power trunk do
 generate the corresponding resistive circuit;
for each physical block do
   begin
   if it is a reused block
then
 generate the corresponding resistive circuit using the model as shown in Section
            3.2;
else
 generate the corresponding resistive circuit using the equivalent current source
            model;
  end
end.
```

Fig. 4. The algorithm to generate an equivalent resistive circuit for the full-chip

Finally, all the equivalent circuits, including power pads, power trunks, and physical blocks, are combined together. As a result, a full-chip equivalent resistive circuit in SPICE format is already generated.

4.2 The Analysis Procedures

After the full-chip equivalent resistive circuit is obtained, we can perform full-chip power network analysis through SPICE simulation. Because of metal resistances, the full-chip simulation has the following properties:

- The voltages on different grids of the global power network are different.
- The voltages on different grids of a block are different.
- For the same grid in a block, the voltage will be varied when the block is integrated into a SOC design.

Even though a block is safe in the block level analysis, it still may have voltage drop or electromigration problem in the full-chip analysis.

Early analysis enables us to make power network changes when they are easiest and least costly to implement. Because of consistency between cell library and power rail, the values of Rc and Rp in a block are often fixed. Therefore, if any voltage drop or current density problem found in the full-chip power analysis, the designer need to widen the power trunks or add more power trunks. Whenever the global power network is modified, the voltages on power grids may be varied. The effects caused by the modification of global power network are summarized as the below:

- The voltages on the grids of the global power network will be varied, because the corresponding resistances (i.e., Rtrunk) are increased or decreased.
- The voltages on the grids of a row will be varied, if any one voltage at the end points is varied.

If the power network is modified, a new equivalent resistive circuit is generated. Then, the modified power network should be validated again through SPICE simulation. The procedure repeats until no reliability problem found in the final power network.

The analysis methodology can be further generalized. Suppose that a SOC design has M execution modes. Thus, M equivalent resistive circuits need validation. The procedures are elaborated as below. Assume that the SOC design contains N blocks, including B_1, B_2, …, and B_N. Let $P_i(B_j)$ be the peak power consumption of block Bj at execution mode i, where $1 \le i \le M$ and $1 \le j \le N$. For each $P_i(B_j)$, where $1 \le i \le M$ and $1 \le j \le N$, we can choose the suitable modeling technique to describe the local power network structure. Next, for each execution mode i, we can obtain the corresponding full-chip equivalent resistive circuit according to $P_i(B_j)$, where $1 \le j \le N$. As a result, we will have M full-chip equivalent resistive circuits. Note that all the M equivalent resistive circuits should be validated through SPICE simulation. If any reliability problem is found, the related power trunks in the global power network are modified. Based on the modification, new M equivalent resistive circuits are generated. The procedures repeat until no reliability problem can be found.

5 Experimental Results

We have developed the floorplan-based full-chip power network analysis algorithm by using C programming language and integrated it with Star-Hspice on a Sun UltraSPARC-10 workstation. In the following, we will report some experimental results to show the effectiveness of the proposed analysis methodology. The first experiment reports the experimental results on the new modeling technique and gives the comparisons with other approaches [3][4][5]. The second experiment reports the studies on the resolution of grids. In these experiments, four test cases, which resemble the unit placement row in a real block, are used to test the effectiveness of the proposed analysis methodology.

Table 1 tabulates the characteristics of these four test cases with the measured power consumption, the voltage on the end points, and the row length. Without loss generality, we assume the voltages on the two end points are the same. However, note that, due to the metal resistances in power trunks, the voltages on the two end points of a row are often different with each other.

Table 1. Summary of test cases

Test Case	Measured Power	Voltage at End Points	Row Length
CKT1	8 mW	1.8 V	1500 μm
CKT2	4 mW	1.8 V	5000 μm
CKT3	25 mW	1.8 V	700 μm
CKT4	20 mW	1.8 V	250 μm

5.1 Studies on the New Modeling Technique

We compare the new modeling technique with previous works [3][4][5]. Table 2 tabulates the experimental results. The results include the comparison of (i) the derived equivalent grid resistance RC (the unit is in ohm); (ii) the power consumption of the derived equivalent resistive circuit; (iii) the minimum voltage found in the equivalent resistive circuit (the unit is in volt); and (iv) the CPU time to analyze the test case (the unit is in second).

As shown in Table 2, the equivalent resistive circuits derived from our approach have the same power consumption with the original given power consumption (i.e., the measured power shown in Table 1). On the other hand, the power consumptions of equivalent resistive circuits derived by other approaches are less than the original given power consumption. Moreover, because the metal resistances in the power rail are taken into account to derive the equivalent circuit, experimental data also shows that our approach can identify the most critical drop voltage.

The modeling technique of [5] is an extension of [3][4]. It repeats the procedures presented in Section 2 until a converged equivalent current source is obtained. As a result, the grid resistances obtained by [5] are different among all the grids in the row. The grid resistance of [5] reported in Table 2 is the minimum value.

Table 2. Experimental results on different modeling techniques

Test Case	Method	Derived R_C	**Derived Power (mW)**	Minimum Voltage Found	CPU time (second)
CKT1	**Ours**	3751099	**8.0**	1.601	0.20
	[3][4]	4049999	7.4	1.614	0.10
	[5]	4049999	7.3	1.616	1.77
CKT2	**Ours**	7115399	**4.0**	1.474	0.26
	[3][4]	8099999	3.5	1.508	0.12
	[5]	7940414	3.6	1.504	1.77
CKT3	**Ours**	1157669	**24.9**	1.513	0.30
	[3][4]	1295999	22.5	1.540	0.12
	[5]	1295906	22.5	1.540	1.74
CKT4	**Ours**	1532480	**19.9**	1.654	0.24
	[3][4]	1620000	18.9	1.661	0.12
	[5]	1620000	18.5	1.665	1.80

Table 3. Experimental results on different number of grids

Test Case	**The Number of Grids**	**Derived R_P**	**Derived R_C**	**Minimum Voltage**	**Position of Minimum Voltage**
CKT1	**20000**	0.018213	7502199	**1.601**	749.6
	10000	0.036425	3751099	**1.601**	749.7
	8000	0.045530	3000899	**1.601**	749.7
	5000	0.072843	1875589	**1.601**	749.8
	2000	0.182052	750299	**1.601**	749.6
	1000	0.363922	374999	**1.601**	749.2
CKT2	**20000**	0.060711	14231308	**1.473**	2499.8
	10000	0.121416	7115399	**1.474**	2498.7
	8000	0.151767	5692299	**1.473**	2499.0
	5000	0.242809	3557599	**1.474**	2499.5
	2000	0.606839	1422969	**1.474**	2498.7
	1000	1.213073	711499	**1.473**	2497.5
CKT3	**20000**	0.008500	2315299	**1.513**	349.9
	10000	0.016998	1157669	**1.513**	349.8
	8000	0.021247	926099	**1.513**	349.9
	5000	0.033993	578899	**1.513**	349.9
	2000	0.084958	231499	**1.513**	349.8
	1000	0.169830	115719	**1.513**	349.6
CKT4	**20000**	0.024285	3064900	**1.654**	124.8
	10000	0.048567	1532480	**1.654**	124.8
	8000	0.060707	1225969	**1.654**	124.8
	5000	0.097123	766300	**1.654**	124.9
	2000	0.242736	306499	**1.654**	124.9
	1000	0.485229	153199	**1.654**	124.8

5.2 Studies on the Resolution of Grids

At the block level, where local hot spots are located, finer grids are generated to model the local power network structure. The most comprehensive method is to define the resolution of grids as fine as possible. However, the CPU time may increase with the increase of the resolution of grids. Furthermore, the number of grids cannot be infinite. In order to study the effects caused by the resolution of grids, we analyze the test cases with different number of grids. Without loss of generality, the new modeling technique is applied. Table 3 tabulates the results on the four test cases.

For each test case, given a number of girds, Table 3 reports the value of derived Rp, the value of derived Rc, the minimum voltage found, and the position of the minimum voltage, respectively. Obviously, with the increase of resolution of grids, the derived Rp decreases and the derived Rc increases. However, experimental data shows that, in the range of 1000 to 20000 grids, the values of minimum voltage are converged to the same value. In other words, 1000 grids are enough to find the minimum voltage in these test cases. Furthermore, note that 1000 is not the lower bound in these test cases.

Based on the observation as shown in Table 3, we know that the critical drop voltage can be identified with a coarser resolution of grids. As a result, the CPU time of full-chip simulation can be significantly reduced.

6 Conclusions

In this paper, we presented an effective modeling technique for the local power network of a reused block. If compared with existing models, the main distinction of the proposed approach is that the power rails are into account to derive the equivalent resistive circuit. Experimental data shows that the proposed modeling technique can identify the critical drop voltage in reused blocks. We have developed a floorplan-based full-chip power distribution analysis system, which uses different models to represent the power networks of blocks according to their properties. As a result, the designer can accurately predict the electromigration or voltage drop problem at the floorplan stage and make power network changes when they are easiest and least costly to implement. This power network analysis system is well suitable for a SOC design.

References

1. Mitsuhashi, T., Kuh, E.S.: Power and Ground Network Topology Optimization for Cell-Based VLSIs. In: The Proc. of 29th Design Automation Conference, pp. 524–527 (1992)
2. Huang, S.H., Wang, C.L.: An Effective Floorplan-Based Power Distribution Network Design Methodology Under Reliability Constraints. In: Proc. of IEEE International Symposium on Circuits and Systems, vol. 1, pp. 353–356 (2002)
3. Rabaey, J.M., Pedram, M.: Low Power Design Methodologies. Kluwer Academic Publishers, Dordrecht (1996)
4. Yim, J.S., Bae, S.O., Kyung, C.M.: A Floorplan-Based Planning Methodology for Power and Clock Distribution in ASICs. In: Proc. of Design Automation Conference, pp. 766–771 (1999)
5. Cho, D.S., Lee, K.H., Jang, G.J., Kim, T.S., Kong, J.T.: Efficient Modeling Techniques for IR drop Analysis in ASIC Designs. In: Proc. of the 12th Annual IEEE International ASIC/SOC Conference, pp. 64–68 (1999)

A Multi Variable Optimization Approach for the Design of Integrated Dependable Real-Time Embedded Systems*

Shariful Islam and Neeraj Suri

Department of CS
TU Darmstadt, Germany
{ripon,suri}@cs.tu-darmstadt.de

Abstract. Embedded systems increasingly encompass both dependability and responsiveness requirements. While sophisticated techniques exist, on a discrete basis, for both *dependability/fault-tolerance* (FT) and *real-time* (RT), the composite considerations for FT+RT are still evolving. Obviously the different objectives needed for FT and RT make composite optimization hard. In this paper, the proposed *Multi Variable Optimization* (MVO) process develops integrated FT+RT considerations. We introduce dependability as an initial optimization criteria by confining error propagation probability, i.e., limiting the interactions. Subsequently, quantification of interactions together with RT optimization by minimizing scheduling length is developed. A simulated annealing approach is utilized to find optimized solutions. We provide experimental results for our approach, showing significant design improvements over contemporary analytical initial feasibility solutions.

1 Introduction and Paper Objectives

Embedded real-time systems with implications on system dependability[1] are being employed in diverse applications such as flight, drive and process control. More and more functionality is being integrated into such systems, invariably leading to a heterogeneous environment consisting of applications of different *criticality* (both safety critical (SC) and non-SC), each with associated *responsiveness* requirements. Each application introduces system level constraints such as software (SW) complexity, cost, space, weight, power and multiple other realization constraints making the overall system composition a complex resource optimization task.

Thus, efficient system design strategies are needed to integrate these diverse applications across limited hardware (HW) resources while considering the interplay of fault-tolerance (FT) and real-time (RT) objectives. *Mapping* of mixed criticality/responsiveness applications onto shared resources is a crucial step for

* This work has been partly supported by the EU IST FP6 DECOS.

[1] The terms Dependability and FT will be used synonymously in the paper.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 517–530, 2007.

such a system design strategy. A mapping is defined as: *(i)* assignment of jobs[2] to suitable HW nodes such that platform resource constraints and dependability requirements are met (*resource allocation*) and *(ii)* ordering job executions in time (*scheduling*). This design step faces new challenges under resource constraints and needs careful attention such that FT and RT requirements are not compromised. Moreover, design optimization involves simultaneous consideration of several incompatible and often conflicting objectives.

Recently, bi-criteria objectives have attracted attention by researchers, e.g., in [1],[2],[3]. The first two papers consider the trade-off between system reliability and scheduling length, while the third considers minimization of energy consumption using checkpointing to recover from faults. The complexity of the design endeavor becomes apparent when considering the huge design space of possible solutions on one hand, and the many competing design objectives on the other [4]. Overall, better design methodologies are needed to handle such complex problems.

We propose a generic optimization framework considering different design *variables*[3] both from Dependability/FT and RT perspectives. The approach is called *Multi Variable Optimization (MVO)*, which takes into account the satisfaction of various constraints as well as optimization of multiple competing variables. Since dependability is a primary design objective for safety-critical systems, the proposed framework puts emphasis on FT through *replication* of highly critical jobs. Dependability is then enhanced by providing error-containment mechanisms. If an error is present in a job, it is possible for this error to propagate to other jobs and cause multiple failures[4]. In ultra-dependable systems even a very small correlation of failures of the replicated units can have a significant impact on the overall dependability [5]. Thus, highly interacting jobs are assigned onto the same node, to prevent the spread of errors across HW nodes, i.e., enhance dependability by *error confinement* [6]. We also minimize the scheduling length while satisfying job precedence and deadline constraints and minimize the utilization of the network.

The contributions of our work are: *(i)* development of a generic framework which systematically guides the optimized system level design, *(ii)* quantification of application *interactions* and techniques to constrain the propagation of errors, *(iii)* combining interactions with scheduling length and bandwidth utilization that enables us to solve the MVO problem and *(iv)* application of an existing optimization algorithm within the approach, enabling a quantitative evaluation. For a representative target study, our evaluation shows significant design improvements for the considered variables. From a RT perspective, minimizing the scheduling length also gives the basis for maximizing the CPU utilization.

The paper is organized as follows. Section 2 discusses the related work. System models and problem statement are introduced in Section 3. Section 4 presents

[2] Applications are further decomposed into smaller executable units called jobs.
[3] A single variable refers to optimization of a single objective.
[4] The failure of a module (can be an application, a job or a node) due to the failure of another module is called a cascading failure.

the generic MVO framework and Section 5 provides the quantification of the variables. For the evaluation of the approach, we employ simulated annealing described in Section 6. The experimental evaluation and results are given in Section 7. Section 8 concludes the paper.

2 Related Work

Usually, FT is applied to an existing scheduling principle such as rate-monotonic or static off-line either by using task replication [7] or task re-execution [8],[9]. Satisfying RT constraints, in [10], the authors minimize the total completion and communication times. Maximization of the probability of meeting job deadlines is considered in [11]. A scheduling approach for distributed static systems is presented in [12], where the authors minimize the jitter for periodic tasks using simulated annealing. Satisfaction of multiple constraints (timing, dependability) and optimization of a single variable (bandwidth utilization) is presented in [13]. All these approaches either consider dependability as constraint or optimize a selected operational variable from a RT perspective.

Though optimizing one variable is straightforward, optimization of multiple variables is considerably more difficult. Several mapping techniques exist but few are concerned with optimizing dependability/FT and RT issues together. In [14], the authors propose that the combination of active replication and re-execution can provide an optimized design from the scheduling length point of view. In [15], the authors discuss multiple objectives such as minimizing communication, load balancing and minimizing the maximum lateness. [6] specifically addresses dependability (focuses on minimizing interaction) and presents heuristics for conducting the mapping. However, the focus is to aid integration between design stage SW objects. Overall, in design optimization, there is a dearth of work that addresses dependability as an optimization criterion. Commonly, dependability is considered directly as a constraint to be satisfied. Instead, we consider dependability and RT, both as constraints and as optimization criteria.

3 System Model and Problem Statement

Our system design framework is based on the following models: the SW and HW models, constraints model and the fault model. The SW model describes the functional and *extra-functional* (dependability, responsiveness, etc.) requirements of jobs and the HW model is the physical execution platform for those jobs. The fault model depicts the types of faults and their causes, whereas constraints restrict the possible solutions. The rest of this section details various important aspects and characteristics of the different models.

SW Model: The SW model consists of applications of varied criticality. Applications are further decomposed into a set of jobs $(j_1, j_2, ..., j_n)$. A *job* represents the smallest executable SW fragment, with basic communication capabilities for exchanging information with other jobs. We consider a job to have specific properties as required inputs to the mapping process, namely: *(i)* job name - each job

has a unique name; *(ii)* timing requirements (earliest start time-EST, computation time-CT, deadline-D); *(iii)* volume of data for inter job communication in terms of bytes; and *(iv)* dependability/FT requirements - the degree of replication dc_i necessary for the i^{th} job to provide the required level of FT. dc_i is specified by the system user.

HW Model: We assume a network topology allowing every HW node to communicate with all other nodes $(n_1, n_2, ..., n_k)$. A HW node is a self-contained computational element (single- or multiprocessor) connected to the network (e.g., using a bus topology) through a communication controller. We assume that the computing basis of all node processors is similar. HW nodes may also contain additional resources, e.g., sensors, actuators etc. The communication controller controls the exchange of messages with other nodes.

Constraints Model: Constraints define the conditions that limit the possible mappings from a dependability, RT or resource perspective. A set of constraints need to be satisfied for a mapping to be valid [16]. We consider the following constraints: *(i)* binding constraints - jobs that need to be allocated onto specific nodes due to the need of certain resources (e.g., sensors or actuators), *(ii)* FT constraints - separation of replicas to different nodes, *(iii)* schedulability - maintaining RT constraints and *(iv)* computing constraints - such as the amount of memory available for jobs.

Fault Model: We consider both SW and HW faults, therefore a fault can occur in any job, HW node or communication link. The consequence of a fault is an error which can propagate from a source module to a target module via a corrupted message or via a shared resource. In the case of communication links, only transient faults are considered. A single fault, either a transient or a crash [17] impacting a shared resource, is likely to affect several or all of the jobs or replicas running on that node.

Problem Statement: The set of all possible mappings for a given set of jobs and nodes is called the *problem space (X)*, shown in Figure 1. A mapping is either feasible or infeasible. A feasible mapping is a solution which satisfies all constraints. If some constraint is not satisfied, the mapping is infeasible. A *point* x in the problem space X represents a mapping of jobs onto nodes. The *neighborhood* space $N(x) \subseteq X$ of a point x is the set of all points that are reachable by performing a *move* operation (e.g., relocating a job to a different node). We employ a transformation operator (Γ) to perform move operations (see Section 6.3 for details). The *value* of a point is a measure of the suitability of the mapping represented by that point. The function $f(x)$ is used to measure the value of a point of the problem space. For an optimization problem, which minimizes the value of variables (v), good mappings have low values. Hence, the task is to find a mapping $x^* \in X$ with the lowest function value for multiple variables, i.e., $f(x^*) \leq f(x)\ \forall x \in X$. x^* is the best mapping from a global search space (X).

In this work, a feasible mapping is provided as an input to the algorithm and feasibility is maintained throughout the quest by an external function call, therefore the problem space remains in the feasible region $X' \in X$ (set of all

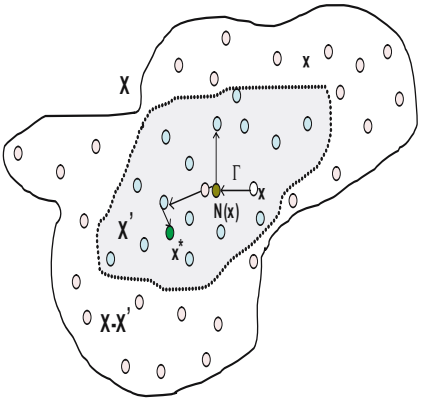


Fig. 1. Search space for the mapping

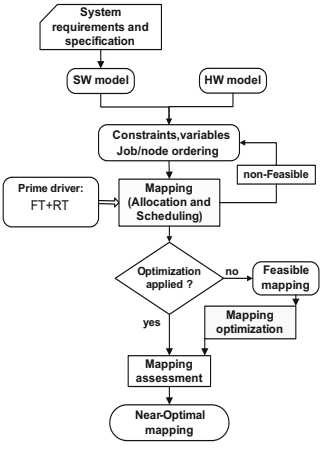


Fig. 2. System design optimization

admissible solutions), which reduces the search space considerably. We strive for finding the mapping $x^* \in X'$, where $f(x^*) \leq f(x)\ \forall x \in X'$. We use an MVO function ($MVO(v)$) to represent the mapping (see Section 6.1 for details).

4 The MVO Approach

The proposed MVO framework systematically guides the FT+RT driven mapping towards an optimized solution. In this section we discuss this generic design framework on the basis of the models presented in the previous section. The system design optimization flow and the corresponding steps are depicted in Figure 2 and in Algorithm 1 respectively. The design process starts with characterizing the SW and HW model. The properties of the model are extracted from the system requirements and specification document. In *Step* 2 (Algorithm 1), constraints are modeled, which need to be satisfied during the mapping. Design variables are defined in *Step* 3, which are employed in the *mapping optimization* phase shown in Figure 2. Variables are used for capturing the design criteria and they strongly depend on the objectives of the system design and on the considered system model [18].

Mapping algorithms need heuristics to achieve good performance. Of particular importance are job ordering and node ordering that decide which job to assign next and what node to assign that job onto [16]. Job and node ordering are described in *Step* 4. A crucial issue that arises at this design stage is the mapping of jobs onto suitable nodes. An initial mapping is created in *Step* 5 where allocation and scheduling is performed off-line in the early design phase. The result of this step is a feasible mapping. However, this mapping is likely to be very inefficient from a system design perspective. The purpose of the rest of the steps is to find a better mapping by using the proposed optimization framework. A candidate mapping from the set of possible solutions is generated in *Step* 6.

Algorithm 1. Generic framework for system level design optimization

```
1: derive the system model
2: extract design constraints
3: define design variables
4: ordering of jobs and nodes
5: generate an initial current mapping - apply heuristics
6: generate candidate mapping - exploring neighborhoods
7: a) compare candidate mapping with the current mapping
   b) go back to Step 6 until stoping criteria is met
8: define minimum requirements to select the mapping (the aspiration values)
9: assess the mapping and return the good mapping (a near-optimal one)
```

In order to select better designs, the candidate mapping is compared with the current mapping in *Step* 7. If a better mapping is found, the current mapping is updated. This step is iterative so that the comparison can be made with all the possible solutions. A detailed description of *Step* 6 and *Step* 7 is also given in Section 6.2. In *Step* 9, the mapping is assessed to ensure that it satisfies the minimum system requirements defined in *Step* 8. Essentially, we are interested in finding a near-optimal mapping meeting FT+RT design objectives.

5 Quantification of Design Variables

In this section we *quantify* the set of variables. This includes how to estimate variables, and how to formulate them in terms of function minimization. The primary objective is to enhance dependability by design, where our focus is to minimize interactions, i.e., to confine the propagation of errors between nodes. The second and third considerations are the scheduling length and the bandwidth utilization respectively that are important in terms of resource utilization and consequently lead to designs with lower cost. In the subsequent sections, we provide details of the variables used to quantify these objectives.

5.1 Interactions

Interaction is the probability of error propagation from a source to a target. This variable refers to how well the errors are contained within a single node. Low interaction values between nodes implies good error containment. Assigning highly interacting jobs on the same node reduces the error propagation probability across nodes. Below we describe two potential ways in which interactions between a source and a target could take place.

Case 1: Errors occur in the source and propagate to the target via message passing or shared resources. If a job is affected by an error of the node it is running on, it might propagate errors to jobs on other nodes with which it communicates or shares a resource. Such interactions risk the failure of multiple nodes and are undesirable.

Case 2: Messages sent over the network can be lost or erroneous due to transmission errors. Erroneous messages can propagate to different nodes and may cause unexpected behavior.

Estimating interactions: The interactions as shown in Figure 3 consists of three phases, namely: *(1)* an error occurring in a module or in a communication link, *(2)* propagation of the error to another module and *(3)* the propagating error causing a cascaded error in the target module. In order to measure interactions, let's assume P_e is the probability of error propagation from source to target considering no corruption over the network and P_l is the probability of message corruption over the network. The probability of error propagation from a source (s) to a target (t) is denoted by $P_{s,t}$ and defined as follows: $P_{s,t} = p\{error\ propagation | no\ corruption\ over\ the\ network\} \cdot p\{no\ corruption\ over\ the\ network\} = P_e \cdot (1 - P_l)$, where, $P_e = P_s \cdot P_t$. The probability that s outputs an error and sends it to the input of t is P_s and P_t is the probability that an error occur in t due to the error received from s. P_s indicates how often s allows errors to propagate and P_t indicates how vulnerable t is to errors propagating from s. Considering both $P_{s,t}$ and P_l, the interaction is calculated as follows: $I_{s,t} = P_{s,t} + P_l$.

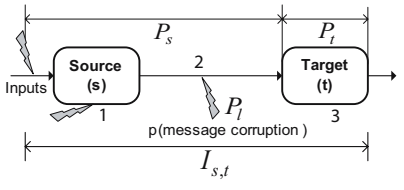


Fig. 3. Error propagation

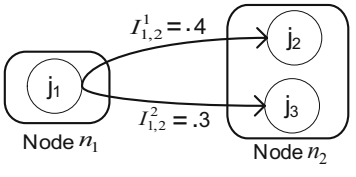


Fig. 4. Combining interactions

$I^o_{s,t}$ is the overall interactions between the set of jobs assigned together on a node and interacting jobs allocated on different nodes, which is expressed by the following equation:

$$I^o_{s,t} = 1 - \prod_{\rho}(1 - I^{\rho}_{s,t}) \tag{1}$$

Where ρ is the number of interactions paths between two nodes. For example, the overall interaction of node n_1 to n_2 as shown in Figure 4, will be $I^o_{n_1,n_2} = 1 - [(1-0.4) \cdot (1-0.3)] = 0.42$. Interactions are assumed to be zero for jobs which are assigned on the same node. However, it is not possible to assign all interacting jobs onto a single node due to the constraints. Also replicas need to be placed on different nodes which might have interactions with other jobs. Hence, there will be jobs interacting across nodes. We strive to minimize these interactions as much as possible for a mapping, such that dependability is enhanced by design. Values for error occurrence probabilities can be obtained, for example, from field data, by fault injection or from system specification [19]. The computation of the system level interactions $\hat{I}$ is expressed as follows, where k is the number of nodes:

$$\hat{I} = \sum_{i,j=1}^{k} I^o_{i,j} \tag{2}$$

5.2 Scheduling Length

This variable represents the total completion and communication time for a set of jobs on a node. As we use replication as the FT scheme, this results in more jobs needed to be scheduled and this naturally incurs a overhead on scheduling. The goal is to minimize overall scheduling length ($\hat{S}_l$) on a node satisfying precedence and deadline constraints. Minimizing scheduling length is important from the viewpoint of the uses of a set of processors, since it leads to maximization of the processor utilization. In every scheduling, gap may remain between two consecutive jobs executing on the same node due to precedence relations and communication cost. We define this gap as in-between slack (IBS). Slack can be used for future upgrading of jobs and also for energy savings.

We have developed a schedulability analysis in our previous work [16] and we employ that strategy in this work as well. The scheduling length for a candidate mapping is calculated using the following equation:

$$\hat{S}_l = \forall k \; max \left[\sum_{i,j=1}^{n} (M_{i,k} \cdot CT_{i,k} + IBS_{i,j}) \right] \tag{3}$$

where, n is the number of jobs, $CT_{i,k}$ is the computation time of the i^{th} job in the k^{th} node, $IBS_{i,j} = EST_j - LET_i$, where i is the job executed before j on the same node, and $M_{i,k} = 1$, if j_i is assigned to the k^{th} node and 0 otherwise. LET_i is the latest ending time of job i.

5.3 Bandwidth Utilization

In an integrated system design, jobs of different criticality and from different applications may be assigned onto a single node and jobs from a single application may be assigned onto different nodes. Therefore, good utilization of shared communication links is necessary. Bandwidth utilization ($\hat{B}_w$) is the ratio between the total bandwidth required by the system and the available bandwidth of the network (B_T) defined as follows:

$$\hat{B}_w = \sum_{i,j=1}^{k} b_{i,j}/B_T \tag{4}$$

where k is the number of nodes and $b_{i,j}$ is the total bandwidth requirements in terms of message size between nodes i and j. Minimizing this variable may allow for the use of a slower but cheaper data communications bus [15].

6 The Algorithm - Employing MVO

In the prior sections, we described the MVO framework and the quantification of considered variables. Next, we apply an existing optimization algorithm within our framework. For this purpose we have chosen simulated annealing (SA) [20].

SA is an algorithm, which converges to the global minima while solving an MVO problem (MVO-SA). SA is a long established effective metaheuristic with an explicit schema for avoiding local minima [12],[13],[20],[21]. Alternative approaches such as Genetic algorithm, Tabu search [21] were also investigated as options. However, the global minima possibility with SA makes it attractive. The overall optimization process is shown in Algorithm 2, which differs from usual single objective SA. We have adapted SA for multiple objectives, which returns the best values of variables together with the best mapping found so far.

6.1 The MVO Function

$MVO(v)$ is a function, which returns a natural number that corresponds to the overall quality of a given mapping. We construct the $MVO(v)$ function as a weighted sum of the variables, which is a widely used method for this class of problem [18],[22]. The value of the function is determined by using the values of variables $\hat{I}, \hat{S}_l,, \hat{B}_w$ and the trade-off factors ψ_i, ψ_s and ψ_b.

$$MVO(v) = \psi_i \cdot \hat{I} + \psi_s \cdot \hat{S}_l + \psi_b \cdot \hat{B}_w \tag{5}$$

The individual values of the variables are represented in a matrix form: $M[v] \equiv M[\hat{I}, \hat{S}_l, \hat{B}_w]$. After performing a move, the function is denoted as $MVO(v^{'})$ and the matrix as $M[v^{'}]$.

6.2 Application of SA

The MVO-SA algorithm requires the following inputs: *(i)* the set of jobs and nodes including their properties to create the initial mapping, *(ii)* variables, the $MVO(v)$ function and the trade-off factors, *(iii)* the Γ operator to change the mapping and *(iv)* SA parameters - initial temperature, the cooling schedule and the cooling factor for lowering the temperature. The output of the algorithm represents the optimized mapping of jobs onto nodes.

After setting the initial heating temperature T_h (Algorithm 2), the initial feasible mapping is created. The feasibility of the mapping is maintained throughout the search by an external function call, i.e., the best feasible mapping is sought. The values of all variables are set in the MVO function (Equation 5) and $MVO(v)$ is computed in *Step* 4. In order to generate the candidate mapping, neighborhoods are explored in *Step* 6. We apply the *transformation operator* *(Γ)* to explore neighborhoods. While applying this operator the feasibility of the mapping is checked. In *Step* 9, the candidate mapping $MVO(v')$ is evaluated in order to compare it with the current best mapping. If the difference $\delta v = MVO(v') - MVO(v)$ is less than zero (minimization) then we choose the candidate mapping. If δv is greater or equal to zero, then the candidate mapping is accepted with a certain probability, called the acceptance probability (a_p). One of the commonly used acceptance probability functions is $a_p = e^{-\delta v / T_h}$ [12],[13].

The technique used by SA to not get stuck at a *local optima* is to accept some worse moves as the search progresses. For larger δv, i.e., when the candidate mapping is extremely undesirable, the probability of acceptance diminishes.

Algorithm 2. MVO algorithm - SA based

```
1: initialization T; heating temperature T_h
2: generate an initial mapping
3: create the matrix M[v] ≡ M[Î, Ŝ_l, B̂_w] for this mapping
4: evaluate the initial mapping MVO(v)
5: repeat
6:    explore neighborhood of the current mapping using Γ
7:    generate candidate mapping
8:    create matrix M[v'] ≡ M[Î', Ŝ'_l, B̂'_w]
9:    evaluate candidate mapping MVO(v') for the new matrix
10:   calculate δv = MVO(v') − MVO(v)
11:   if δv < 0 then
12:      M[v] = M[v'] and MVO(v) = MVO(v')
13:   else
14:      calculate acceptance probability a_p = e^{−δv/T_h} and
         generate r = random[0, 1]
15:      if a_p ≥ r then
16:         M[v] = M[v'] and MVO(v) = MVO(v')
17:      else
18:         restore the current mapping, i.e., keep M[v] and MVO(v).
19:      end if
20:   end if
21:   reduce the temperature T_h by using a cooling schedule T_{h−1} = c_f · T_h.
22: until some stopping criterion is met
23: return the best matrix M[v] and corresponding mapping MVO(v)
```

The initial temperature (T) is set to a sufficiently high value to accept the first few candidate mappings. However, the a_p decreases as T_h decreases. If an acceptance criteria is met, the candidate mapping is chosen, otherwise the current mapping is restored and the process is continued. T_h is reduced according to the cooling scheduling $T_{h-1} = c_f \cdot T_h$, which is the most commonly used in the literature [12],[22], where c_f is the cooling factor. We perform several iterations at the same T_h (so called Metropolis Monte Carlo attempts [20]) to cover a larger search space. The algorithm returns the best mapping found so far when the temperature is reduced to a certain value.

6.3 The Transformation Operator Γ

As mentioned before, the operator Γ performs the changes/moves to the mapping in order to generate a candidate mapping. Specifically, Γ generates the move to perform the *local search*, i.e., to explore the neighborhood. Three commonly used moves [22] are discussed below: *(a) relocate* a job to a different node, *(b) swap* the nodes between two jobs and *(c) interchange* the allocated jobs between two nodes. A move is accepted when it satisfies all the constraints defined in Section 3. After a successful move, the candidate mapping is evaluated in *Step* 9 (Algorithm 2).

7 Evaluation of the MVO Framework

In this section we first present the experimental setting. Based on this we evaluate the effectiveness of the MVO approach and discuss the results. Results show

significant improvements in terms of *interactions, scheduling length, bandwidth utilization, CPU utilization and FT overhead.*

7.1 Experimental Settings

For the evaluation of our approach, we use randomly generated mixed-criticality sets of 40, 60 and 80 jobs denoted as $J40$, $J60$ and $J80$ respectively. All jobs, along with their replicas, are to be assigned in an optimized way onto the available nodes. All job properties are uniformly distributed within the following ranges: Replication factor $\in \{2, 3\}$, Interaction $\in [.04, .52]$, $EST \in [0, 50]$ ms, $CT \in [2, 17]$ ms, $D \in [15, 200]$ ms, Memory size $\in [4, 10]$ MB, Message size $\in [30, 120]$ bytes. Sensors and actuators are attached to arbitrary nodes. The message transmission delay time (size of the exchanged messages divided by transmission speed of the link) between communicating jobs executing on different nodes are subtracted from the deadlines. The HW model comprises 8 nodes, which are connected to a communication link with a speed of $150kbps$. The memory capacities of nodes were arbitrarily chosen as 100, 150 and 250 MB; nodes n_2 and n_3 have sensors and n_5 and n_7 have actuators attached to them.

As used in literature [20],[22] and after investigating different runs of our algorithm with various configurations, we tune the SA parameters as follows: the value of the initial temperature (T) was set to 50000, the cooling factor was set to 0.98, and the used trade-off factors were $\psi_i = 1500$, $\psi_s = 20$ and $\psi_b = 500$ respectively. In order to generate the candidate mapping we have performed two types of moves (random reallocation and swapping - 50% each of Monte Carlo iteration) at the same temperature to cover a larger search space. The third type of move is not relevant in our case study. Experiments showed that applying both types of moves together gives a better solution than only using a single type of move.

7.2 Experimental Results

Performance Evaluation: We first observe the convergence of MVO-SA. Figure 5 shows that after a certain number of iterations with decreasing temperature the MVO function reaches a minimum. At higher temperatures, more states have been visited by the operator Γ to cover the search space. Given the proper selection of the parameters and the problem size, SA gives the global solution by construction [20],[21]. Nevertheless, we performed several experiments to evaluate if the MVO algorithm converges to a single point. Even though the algorithm is started with different feasible mappings ($Feas1$, $Feas2$ as shown in Figure 5), MVO-SA converges towards a solution every time. However, the convergence points may differ negligibly, as shown in Figure 5 in case of $J60$. A good performance test of a mapping algorithm is to take a solvable problem and add resources [13], the algorithm should return a mapping no worse than the result of the original problem. We added two more nodes to the configuration of $J40$ and the resulted mapping displayed better performance. The convergence is shown in Figure 5 marked as $J40$ (10 nodes). To show the effectiveness of

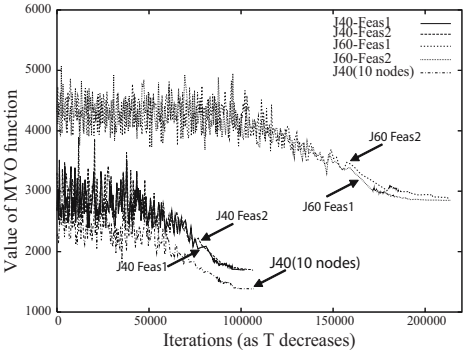


Fig. 5. Performance evaluation of MVO-SA

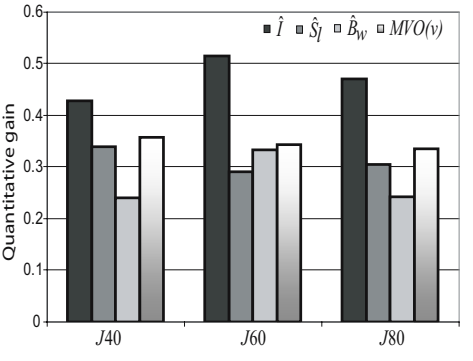


Fig. 6. Mapping M_{PF}

starting the optimization with a feasible mapping, we also ran the algorithm starting from an infeasible mapping. Though this can converge to an improved solution, it is slower than starting from a feasible solution (time for the creation of feasible mapping is included).

Quantitative Gain: We are interested in evaluating the *quantitative gain* compared to a contemporary initial solution. As this gain depends on the value of the initial mapping, we performed experiments using different initial feasible mappings. Figure 6 depicts the mapping performance profile (M_{PF}) for $J40$, $J60$ and $J80$ in terms of $\hat{I}, \hat{S}_l$, $\hat{B}_w$ and $MVO(v)$. M_{PF} is shown as relative gain with respect to the initial mapping. We observe that the gain is higher in case of $\hat{I}$, which ensures FT driven design. In our case studies, on average, our approach found 35% *better solutions* (composite FT+RT gain), which leads to significantly better designs for dependable real-time embedded systems.

CPU Utilization and FT Overhead: Figure 7 shows the computation utilization by different node's processors for jobs set $J40$, $J60$ and $J80$, which is about equally distributed among CPUs, i.e., *a proper load balancing* is maintained by the approach. It is calculated by $UF = \frac{\sum_{i=1}^{n}(M_{i,k} \cdot CT_{i,k})}{\hat{S}_l}$. We observe the FT overhead both for initial and optimized mapping in terms of scheduling length. We varied the replication factor (Replication factor = # jobs after replication/# jobs) from 1 to 3. On average, the quantitative gain is 34.33%. Obviously, scheduling length has increased due to increasing the replication factor. Therefore, a design trade-off

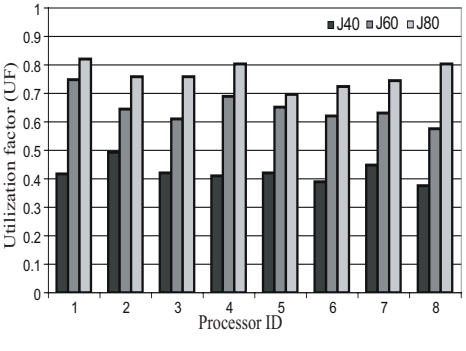


Fig. 7. CPU utilization

between RT properties and the level of FT is necessary. The quantitative gain shows that the overhead is reduced significantly by the optimized mapping, which provides an FT design with reduced scheduling length.

8 Conclusions

We have presented a generic Multi Variable Optimization (MVO) framework for designing embedded systems. The experimental results show the effectiveness of the approach and a significant improvement of the FT+RT system design compared to a straightforward solution where optimizations have not been applied. Particularly, we emphasize the following preeminent benefits of our approach: *(i)* FT is provided and then it is enhanced by restricting the possible nodes from correlated faults, *(ii)* RT requirements are met and the scheduling length is minimized, which increases the overall system performance and *(iii)* bandwidth utilization is reduced, which allows the use of a slower but cheaper bus. The generic framework also allows more variables to be considered, e.g., power.

References

1. Assayad, I., Girault, A., Kalla, H.: A Bi-Criteria Scheduling Heuristic for Distributed Embedded Systems under Reliability and Real-Time Constraints. In: DSN, pp. 347–356 (2004)
2. Dogan, A., Özgüner, F.: Biobjective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. Comput. J. 48(3), 300–314 (2005)
3. Melhem, R., Mosse, D., Elnozahy, E.: The Interplay of Power Management and Fault Recovery in Real-Time Systems. IEEE Trans. Comput. 53(2), 217–231 (2004)
4. Eisenring, M., Thiele, L., Zitzler, E.: Conflicting Criteria in Embedded System Design. IEEE Design and Test 17(2), 51–59 (2000)
5. Bouyssounouse, B., Sifakis, J.: Embedded Systems Design: The ARTIST Roadmap for Research and Development. Springer, Heidelberg (2005)
6. Suri, N., Ghosh, S., Marlowe, T.: A Framework for Dependability Driven Software Integration. In: ICDCS, pp. 406–415 (1998)
7. Oh, Y., Son, S.H.: Enhancing Fault-Tolerance in Rate-Monotonic Scheduling. Real-Time Syst. 7(3), 315–329 (1994)
8. Ghosh, S., Melhem, R., Mossé, D.: Enhancing real-time schedules to tolerate transient faults. In: RTSS, pp. 120–129 (1995)
9. Kandasamy, N., Hayes, J.P., Murray, B.T.: Tolerating Transient Faults in Statically Scheduled Safety-Critical Embedded Systems. In: SRDS, pp. 212–221 (1999)
10. Lo, V.M.: Heuristic Algorithms for Task Assignment in Distributed Systems. IEEE Trans. Comput. 37(11), 1384–1397 (1988)
11. Hou, C.-J., Shin, K.G.: Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems. IEEE Trans. Comput. 46(12), 1338–1356 (1997)
12. Natale, M.D., Stankovic, J.A.: Scheduling Distributed Real-Time Tasks with Minimum Jitter. IEEE Trans. Comput. 49(4), 303–316 (2000)

13. Tindell, K., Burns, A., Wellings, A.: Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. Real-Time Syst. 4(2), 145–165 (1992)
14. Izosimov, V., Pop, P., Eles, P., Peng, Z.: Design Optimization of Time-and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In: DATE, pp. 864–869 (2005)
15. Ekelin, C., Jonsson, J.: Evaluation of Search Heuristics for Embedded System Scheduling Problems. In: Constraitnt Programming, pp. 640–654 (2001)
16. Islam, S., Lindström, R., Suri, N.: Dependability Driven Integration of Mixed Criticality SW Components. In: ISORC, pp. 485–495 (2006)
17. Laprie, J-C., Randell, B.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (2004)
18. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives: Preferences and Value Tradeoffs. Cambridge University Press, Cambridge (1993)
19. Jhumka, A., Hiller, M., Suri, N.: Assessing Inter-Modular Error Propagation in Distributed Software. In: SRDS, pp. 152–161 (2001)
20. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. J. of Science 220(4598), 671–680
21. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Comput. Surv. 35(3), 268–308 (2003)
22. Silva, J.L.: Metaheuristic and Multiobjective Approaches for Space Allocation. University of Nottingham, UK, PhD thesis (2003)

SystemC-Based Design Space Exploration of a 3D Graphics Acceleration SoC for Consumer Electronics

Tse-Chen Yeh, Tsung-Yu Ho, Hung-Yu Chen, and Ing-Jer Huang

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 804, Taiwan
{tcyeh,tyho,hychen}@esl.cse.nsysu.edu.tw,
ijhuang@cse.nsysu.edu.tw

Abstract. In order to solve the system performance bottleneck of a 3D graphics acceleration SoC, we exploit design space exploration on performance evaluation and benchmark characteristics using SystemC. We find out the bottleneck according to the simulation results of 9 hardware/software configurations and find out the tradeoffs between different configurations. The performance issues of SoC have been explored under the low-cost constraints, such as cache size effect, hardware accelerations and memory traffic. In conclusions, we provide the performance/cost tradeoffs and 3D graphics benchmark features for designing a 3D graphics SoC.

Keywords: design space exploration, SystemC modeling, 3D graphics SoC, transaction-level modeling.

1 Introduction

Consumer electronics are cost-sensitive, such as digital television (DTV), and customers demanded higher performance with lower price. To develop a system-on-a-chip (SoC) as discussed above needs the hardware/software configurations to meet the performance under the cost constraint. In order to find the tradeoffs between performance and cost, design space exploration using SystemC has been evaluate the system performance in [1].

The fast hardware/software configuration can be achieved by using CoWare™ Platform Architect, which provides the hardware/software integration and simulation platform using SystemC [2]. Firstly, software written by C/C++ can be translated effortlessly into SystemC hardware model because SystemC is a C++ class library. Secondly, CoWare™ Platform Architect provides IP library and cycle-accurate transactional bus simulator (TBS) for IP reuse which can shorten the design flow from re-design the IP models and raise the simulation results up to the bus cycle accurate [3].

In this paper, we focus on 3D graphics hardware acceleration SoC for DTV system. The design flow starts from pure software simulation on the target platform, and then the software will be accelerated by additional hardware or architecture refinement according to performance bottlenecks we have found out. After acceleration, the new bottleneck will be revealed. Thus we continue to simulate new

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 531–540, 2007.

accelerated platform then overcome new bottlenecks repeatedly until the required performance has been approximated.

The organization of this paper is as follows. In Section 2, we describe the simple 3D graphics pipeline, 3D graphics acceleration SoC platform and platform building methodology. The hardware/software configurations are presented in Section 3. And design space exploration will be implemented in Section 4. Finally, we conclude the overall analysis and performance/cost tradeoffs for 3D graphics acceleration SoC Design.

2 Preliminary

3D graphics has been developed maturely on workstation and personal PCs from 1990. These high performance solutions are not suitable for the embedded systems or consumer electronics because they also paid high cost and high power consumption. Although the 3D graphics workloads have been investigated by static and dynamic forms [9, 10] and one is on the mobile device [11], the system performance still has been profiled on instruction-level evaluation.

Due to the proposed architecture of 3D graphics SoC will be applied on DTV applications, the differences will be introduced in the following section opposed to the conventional 3D graphics processing.

2.1 3D Graphics Rendering Pipeline

To obtain the software functionality, we exploit a tile-based simple 3D graphics pipeline (shown in Fig. 1) written by C/C++. The tile-based rasterization can reduce amount of memory accesses and is suited to the embedded system. In the simple pipeline, the geometry processing includes the viewing transformation, lighting and perspective transformation. After geometry computation, the triangle information will be passed to the tile divide function which handles the tile setup information. The raster processing begins at the scan conversion, shades fragment without texture, and the final results will be written into frame buffer for display after passing Z-test. In this paper, the texturing will not be modeled for performance evaluation.

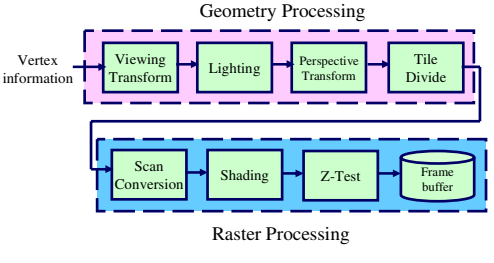


Fig. 1. Simple 3D graphics pipeline for performance analysis

2.2 3D Graphics Acceleration SoC

The basic combinations of a system includes core processor, system bus and system memory. In order to accelerate 3D graphics processing, a 3D graphics engine has to

handle the geometry and rasterization acceleration. Furthermore, the processing results should be display via a display engine (DE) which has responsibility to move processed data from frame buffer in memory to the display device. Finally, the direct memory access (DMA) will be appended for more efficient memory access consideration. All the hardware components connect the system bus via bus interface (BI).

Fig. 2 shows the final platform of 3D graphics acceleration SoC. We choose ARM926EJ-S for the system processor core and AMBA 2.0 AHB bus to be the system bus. Taking low cost into account, we select the single port SDRAM to be the system memory which contains vertex information, temporary data, frame buffer and Z buffer. To fit the DTV applications, the frame buffer and Z buffer are located with 640x480 screen size.

To approximate the realistic system memory traffic, the bus contentions from DE should be taken into consideration. The DE is a bridge between frame buffer and DTV display, and get the frame buffer data with read burst transfer periodically. According to the DTV frame rate, the data consuming frequency is 25 MHz which occupied 100 MB/sec of system bandwidth under 200 MHz system clock frequency.

In terms of 3D graphics engine in Fig. 2, the functions in 3D graphics pipeline have been translated into hardware model. Because of the complexity of geometric computation, we divide the geometry module into 3 pipeline stages and each stage runs 16 cycles [5]. For design a low cost raster module we choose the tile-based implementation to reduce area cost. Thus we first have to design a tile divider (TD) that receives the triangle data output from geometry module to generate tile list [6].

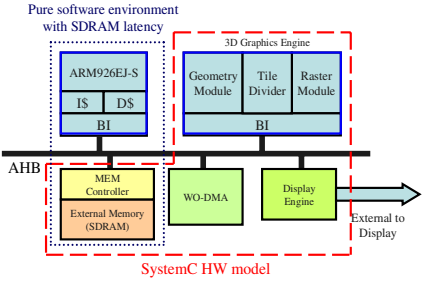


Fig. 2. System functional block of 3D graphics acceleration SoC

2.3 Platform Building

CoWare™ Platform Architect integrates the IP library and hardware models into system platform for simulation. Both of the processor and AHB bus IP are provided by the IP library. The processor IP contains an ARM instruction set simulator (ISS) for software execution. And AMBA library use a cycle-accurate TBS and a set of Transaction-Level Modeling (TLM) API for IP integration [7]. To declare ports type pre-defined in AMB bus library, hardware can read from or write to other components connected to the same bus by API function calls. All of the bus arbitrations and read/write latency are managed by TBS.

For the cycle-accurate simulation, we model hardware modules at transaction level (TL) with timing information [4]. The power of TLM is to separate hardware operations into computation and communication. The computation means the function of hardware, and the communication presents the read/write operations of hardware. Because we use SystemC as modeling language, the functions of simple 3D graphics pipeline can be migrated into computation part of our hardware model without any modification. With regard to communication, the primitive SystemC ports of our hardware modules will be replaced by AHB ports for platform integration.

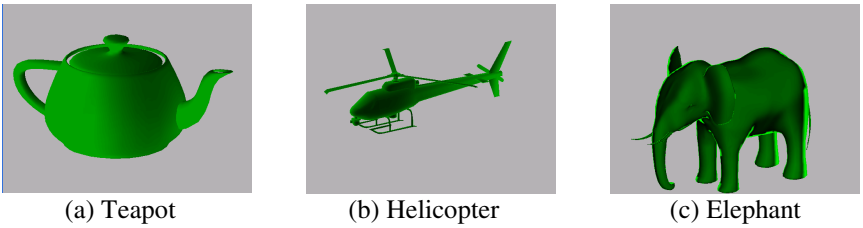
We use dot line to enclose the simulation environment for pure software in Fig. 2. And the rest blocks enclosed by dash line are the hardware modules modeled by SystemC. Through the SystemC modeling at transaction level, we can evaluate the system performance at early design stage and the simulation time of TLM is faster than the Register-Transfer Level (RTL) simulation.

3 System Configurations

In this section, the 3D graphics benchmarks and hardware/software configurations will be described. Total 9 hardware/software configurations will be introduced for our design space exploration, 4 of these configurations are used to evaluate software acceleration, and others are used to explore different hardware solutions.

3.1 Simulation Environment

We choose three benchmarks for 3D graphics rendering shown in Fig. 3, and features of these benchmarks are listed in Table 1. Helicopter benchmark has more geometric computations than raster processing in proportion compared with Teapot benchmark. And the Elephant benchmark has the most complexity on whether geometric or raster processing. All of the simulation statistics will be collected by TBS for the performance analysis.



(a) Teapot (b) Helicopter (c) Elephant

Fig. 3. 3D graphics benchmarks for design space exploration

Table 1. Features of three 3D graphics benchmarks

	Teapot	Helicopter	Elephant
Total vertices number	12,288	15,414	87,840
Processed tiles	135	64	141

3.2 Hardware/Software Configurations

Table 2 gives the detail hardware/software configurations for design space exploration. In this table, "pSW" means 3D graphics pipeline execution without any hardware acceleration. And "ADS" enclosed by parentheses indicate the simulation use ARMulator, i.e. ARM ISS of ARM Developer Suite, without SDRAM model and DE contentions. And "$" suffix refers to software performance improved by adding instruction/data (I/D) cache. "PA" is the abbreviation of Platform Architect, which means the software execute with SDRAM model.

Table 2. Hardware/software configurations for design space exploration

Configuration Name	hardware/software Configurations							
	Function			Bus Interface	SDRAM latency	Cache	Display Engine	Geometry FIFO
	Clear Buffer	Geometry	Raster	Transfer Mode				
pSW (ADS)	SW	SW	SW	N/A	N/A	N/A	N/A	N/A
pSW (ADS $)	SW	SW	SW	N/A	N/A	ON	N/A	N/A
pSWD (PA)	SW	SW	SW	N/A	ON	N/A	ON	N/A
pSWD (PA $)	SW	SW	SW	N/A	ON	ON	ON	N/A
GED_s	SW	HW	HW	Single	ON	N/A	ON	N/A
GED_b	SW	HW	HW	Burst	ON	N/A	ON	N/A
GDGED_b	GDMA	HW	HW	Burst	ON	N/A	ON	N/A
WDGED_b	WDMA	HW	HW	Burst	ON	N/A	ON	N/A
WDGED_Fb	WDMA	HW	HW	Burst	ON	N/A	ON	ON

The prefix "GE" of fifth and sixth configurations indicate the 3D graphics pipeline accelerated by 3D Graphics Engine shown in Fig. 2, and the appending character "s" and "b" means hardware read/write in single transfer or burst transfer mode individually. For bus cycle-accurate simulation, we insert execution cycles into hardware models which refer to the synthesizable RTL design [4-5]. Furthermore, the suffix "D" indicates the DE will occupy 15% bus bandwidth.

"Clear Buffer" is not the function of 3D graphics pipeline, but is necessary which reset values in frame buffer and Z buffer before rendering the next frame. The last three configurations with prefix "GD" or "WD" mean the different DMAs used to improve the performance of "Clear Buffer". We use "GD" to indicate a generic DMA which can read/write memory, and use "WD" to represent a write-only DMA which only write a fixed value from the internal register to reset memory, thus the specific DMA can reduce memory traffic by decreasing lots of read operations from memory.

"Geometry FIFO" will be used for the comparisons of performance improvement on geometry module with or without a ping-pong buffer. In the "Function" column, the "HW" indicates the configuration use hardware implementation and "SW" indicates software implementation. In "Transfer Mode" column, the "Single" means the single transfer mode of AMBA bus, and the "Burst" means the burst transfer mode of AMBA bus.

4 Design Space Explorations

In the section, we exploit these hardware/software configurations introduced in previous section on design space explorations. Bus and memory overhead will be

explored in section 4.1, 3D graphics engine will be appended in section 4.2, and the efficiency of different transfer mode of AHB bus will be discussed in section 4.3. The new bottlenecks will revealed by our explorations in section 4.4. In addition, an imagined geometry FIFO buffer will be modeled for advanced performance/cost tradeoff in section 4.5. Finally, the benchmark characteristics will be extracted for advanced survey on system performance of 3D graphics in section 4.6.

4.1 Cache Size Optimization Without Hardware Acceleration

The design space exploration starts from software evaluation without any hardware acceleration. And we apply cache to achieve more efficient performance on pure software executions. Fig. 4 shows the execution cycles variations of cache size from 128 bits to 16k bits are estimated in "ADS", in contrast to "PA" started from 4k bits due to processor IP limitation. We adapt I/D cache of 4kb/4kb for performance optimization. Either "pSW (ADS$4)" or "pSW (PA$4)" configurations can gain at least 36.7% performance improvement shown in Table 3. The ideal bus assumes the bus without transaction latency, and the subtraction of execution cycles of "ADS" and "PA" configurations is the bus and SDRAM overhead in pure software solutions.

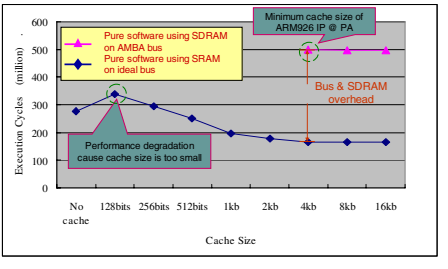


Fig. 4. Performance improvement by cache size variation of Teapot benchmark

Table 3. Cache size optimization on software execution

	pSW (ADS) (cycles)	pSW (ADS$4) (cycles)	improvement (ADS)	psW (PA) (cycles)	pSW (PA$4) (cycles)	improvement (PA)
Clear Buffer	2,935,822	2,167,247	26.2%	9,003,353	5,778,533	35.8%
Geometry	133,002,461	66,931,465	40.1%	396,033,200	209,441,410	47.1%
Raster	139,867,002	97,181,194	30.5%	460,224,000	284,340,000	38.2%
Total	412,376,118	260,850,843	36.7%	865,260,553	499,559,943	42.3%

4.2 Performance of Hardware Acceleration

Due to the "pSW" results, the geometry and raster processing are the bottlenecks on software execution, because the best case of drawing Helicopter benchmark with I/D cache need 1.216 second for one frame at 200MHz system clock frequency shown in Table 4. The DTV applications only can tolerant the frame rate down to 5~6 frame per second under the lowest resolution. To overcome these bottlenecks, we exploit hardware acceleration on geometry and rasterization parts. To compare "pSW" with cache optimization with "GE" configurations, the system performance can be improved

Table 4. Comparisons of execution time at 200MHz system clock frequency

	Teapot	Helicopter	Elephant
pSW (PA$4)	2.498 sec	1.216 sec	9.078 sec
GED_s	0.070 sec	0.067 sec	0.108 sec
GED_b	0.063 sec	0.053 sec	0.070 sec

approximate 99% ((Execution cycle of software – Execution cycle of hardware) / Execution cycle of software) by hardware acceleration.

4.3 Efficiency of Bus Transfer Mode

Due to the cost consideration, we use the SDRAM to be the system memory. Amount of memory read/write accesses occur significant memory latency. The AMBA AHB provides burst mode transfer which can overlap these SDRAM CAS and row change latency [8], but may degrade few bus traffic if too many burst transfers need to compete.

We use "GE_s" and "GE_b" configurations to find out the benefit by using burst transfer. Notice that benefit will not reveal if we use low-latency SRAM to be system memory. The burst transfer benefits are shown in Table 5, and the results show approximate 33.1% improvement in Teapot benchmark, approximate 47.4% and 49.7% improvement at least in Helicopter and Elephant benchmarks.

Table 5. Single transfer vs. Burst transfer

	Teapot	Helicopter	Elephant
Geometry Speedup	36%	52%	49.7%
Raster Speedup	33.1%	47.4%	54.7%

4.4 Performance Bottleneck on Clear Buffer

After the hardware acceleration, the hidden problems have been enhanced. "Clear Buffer" occupies 86.9% execution time of the total system performance compared with the software solution only 1.0% and becomes the new performance bottleneck shown in Fig. 5.

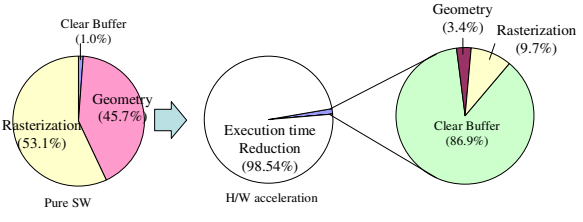


Fig. 5. New bottleneck on Clear Buffer after hardware acceleration

Because the "ClearBuffer" contains the reset operations to the whole frame buffer in memory, we append a generic DMA (GDMA) or write-only DMA (WO-DMA) mentioned in Section 3 to automatically clear frame buffer and Z buffer. Thus large mount of memory traffic between processor and memory can be reduced. Table 6 shows the performance improvement by comparing the execution cycles of "GED_b" with "GDGED_b" and "WDGED_b" configurations. The WO-DMA will be adapted for this system configuration, because the performance has improved 89.4% at least on "Clear Buffer" execution.

Table 6. Performance improvement on Clear Buffer

	Teapot	Helicopter	Elephant
Generic DMA	50.5%	41.77%	41.40%
Write-only DMA	90.76%	89.69%	89.20%

4.5 Performance of Geometry Processing

In order to accelerate the geometry processing, we design a ping-pong buffer to prefetch the vertex information that can prevent the waiting time if bus is too busy to provide the necessary data. "WDGD_Fb" configuration refers to appending the ping-pong buffer.

Table 7. Performance/cost tradeoffs of different hardware/software configurations

hardware/software configuration	Performance improvement			hardware cost (gate count)
	Teapot	Helicopter	Elephant	
Software only	0%	0%	0%	N/A
I/D cache	42.60%	72.51%	35.70%	8kb SRAM
GE	98.54%	98.81%	99.51%	542.8k
GE + WO-DMA	99.69%	99.74%	99.45%	542.8k + 6.2k
GE + WO-DMA+ Ping-pong buffer	99.80%	99.75%	99.44%	542.8k + 6.2k + 32B SRAM

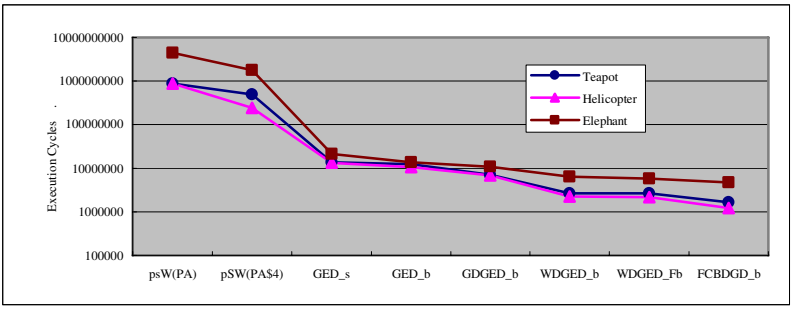


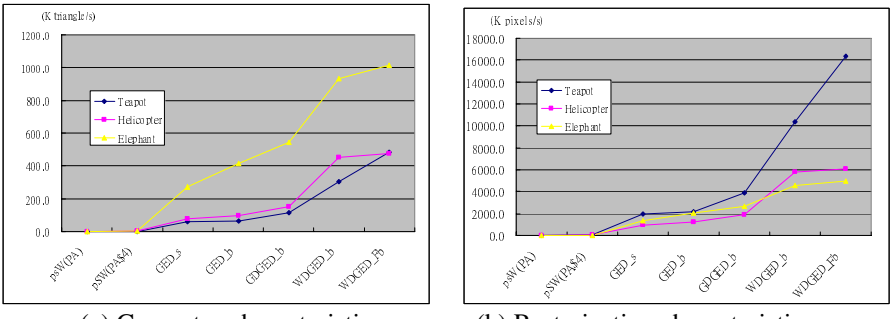
Fig. 6. Performance improvement of different hardware/software configurations

To extract the benefit of ping-pong buffer for geometry processing, we use the "WDGD_b" and "WDGD_Fb" configurations into comparisons. The performance has been improved at least 13.8% on geometry processing by appending 8x4 bytes

ping-pong buffer. Finally, the performance/cost tradeoffs are listed in Table 7, and the hardware costs are estimated by synthesis in [5, 6]. And the performance variations of different configurations are shown in Fig. 6. Also notice Fig. 6 uses logarithm scale on Y axis and start from 100k cycles.

4.6 Characteristics of 3D Graphics Benchmarks

3D graphics workloads have been investigated in [9-11] at instruction-level. After evaluating the system performance of 9 hardware/software configurations, the characteristics of benchmarks can be extracted from 7 configurations because the ISS of ADS can not simulate too large benchmarks, such as Helicopter or Elephant.



(a) Geometry characteristics (b) Rasterization characteristics

Fig. 7. Benchmark characteristics of 3D graphics acceleration

The characteristics of 3D graphics can be divided into geometry and rasterization parts. The triangle rate or vertex rate give the index of geometry performance, and the pixel fill rate give the index to rasterization performance. The benchmark characteristics are shown in Fig. 7. Although the slops of these benchmarks are similar, the amplitudes of variation are not the same. In geometry processing, the Elephant benchmark in Fig. 7(a) gives the significant amplitude because it contains the largest the vertex number of object compared with Teapot and Helicopter benchmarks. On the other side, Teapot benchmark in Fig. 7(b) gives the highest processing area per triangle, thus it can get the largest performance improvement on the rasterization processing. In summary, we use hardware/software configurations not only for design space exploration, but also for extracting the benchmark characteristics.

5 Conclusions

The market of consumer electronics always changes by following consumers' requirements. In this paper, we use the design space exploration for performance evaluation and the performance/cost tradeoffs by exploiting 9 hardware/software configurations. And we also characterize the benchmark features for advanced survey

on 3D graphics processing. The performance evaluation and benchmark characteristics can give many benefits and references for system designers to determine their system architecture and hardware/software configurations.

Notice that memory traffic is still the performance bottleneck of a low-cost SoC which applied on 3D graphics rendering. And texture mapping, memory allocation, comparisons of alternative architecture modeling, power/performance analysis, and multi-layer bus architecture for 3D graphics acceleration SoC will become open issues in our future works.

References

1. Jang, H.O., Kang, M., Lee, M.J., Chae, K., Lee, K., Shim, K.: High-level System Modeling and Architecture Exploration with SystemC on a Network SoC: S3C2510 Case Study. DATE 1, 538–543 (2004)
2. Platform Architect Datasheet. CoWare, http://www.coware.com/PDF/products/Platform Architect.pdf
3. Model Library Datasheet. CoWare, http://www.coware.com/PDF/products/ModelLibrary.pdf
4. Black, D.C., Donovan, J.: SystemC: From The Ground Up. Springer Science+Business Media (2004)
5. Hsiao, S.F., Huang, T.Y., Tieng, T.C.: Design and Verification of a Platform-Based Low-Cost 3D Graphics Geometry Engine Using Area-Reduced Arithmetic Function Units. In: 17th VLSI Design/CAD, pp. 8–11 (2006)
6. Chang, T.N., Tsai, C.H., Tsai, M.C., Lin, H.L.: Design of a Low-cost 3-D Graphic Rendering Accelerator. In: 17th VLSI Design/CAD, pp. 533–536 (2006)
7. AMBA TLM API. CoWare, http://www.coware.com/
8. Synchronous DRAM. Micron, http://wwwmicron.com/sdram
9. Chiueh, T.-C., Lin, W.-J.: Characterization of Static 3D Graphics Workloads. ACM SIGGRAPH/EUROGRAPHICS (1997)
10. Mitra, T., Chiueh, T.-C.: Dynamic 3D Graphics Workload Characterization and the Architectural Implications. In: 32nd AISM, pp. 62–71(1999)
11. Mochocki, B.C., Lahiri, K., Cadambi, S., Hu, X.S.: Signature-Based Workload Estimation for Mobile 3D Graphics. In: 43rd DAC, pp. 592–597 (2006)

Optimal Allocation of I/O Device Parameters in Hardware and Software Codesign Methodology

Kuan Jen Lin, Shih Hao Huang, and Shih Wen Chen

Dept. of Electronic Engineering, Fu Jen Catholic University,
242 Taipei County, Taiwan
kjlin@mail.fju.edu.tw, {a9250627,a9150628}@st2.fju.edu.tw

Abstract. For a programmable I/O device controller, the allocation of device parameters on I/O registers affects the code size and execution time of its associated I/O device driver. In traditional design flow, the development of device drivers can not begin until the allocation is fixed. This paper presents a new design methodology that allows a designer to seek an allocation that reduces the software or hardware cost concurrently with developing device drivers. The software cost means the code size or execution time and the hardware cost the number of I/O registers. The exact allocation with the minimum cost under constraints is formulated as zero-one integer linear programming problem. Heuristic algorithms based on iterative refinement are also proposed. The proposed design methodology was implemented in C language. Compared with current industrial designs, the approach can obtain design alternatives that reduce both software and hardware costs. Furthermore, the experimental results also investigate design spaces for various application features. It turns out that the HW/SW codesign approach is favorable in development of embedded systems.

Keywords: Hardware/Software Codesign, I/O interface, Device Driver, Programmable controller.

1 Introduction

A programmable I/O controller is used to manage a peripheral physical I/O device. A device driver is a software layer that lies between an operation system and the I/O controller. It can configure the operation modes of the device, observe its statuses, and transfer the data via accessing registers in the I/O controller. Figure 1 shows such a hardware and software interface. Usually, a register contains several fields (each occupying several consecutive bits), each of which represents an operation mode, a status or a datum. Each field is referred as a *device parameter* [3] (or device variable as defined in [8]). For example, the length of stop bits is a device parameter of a UART controller. To configure a data frame for a UART transfer, besides of the stop bits, one should also set parity mode, word length and baud rate. These parameters associated with a common purpose form a *parameter group.* To minimize the number of registers and the number of register accesses, hardware designers often allocate device parameters in the same group into the same register. However, this may increase the number of I/O accesses and bit-operations (such as shift instruction and logic

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 541–552, 2007.

instruction) when a driver wants to manipulate individual parameters. Let us see examples in the Fig. 2, that shows various allocations for two parameters A and B. Fig. 3 shows C codes of several access functions in the device driver (or HAL), including *set_B( ), get_A( ),* s*et_A_B( )* (modifying A and B simultaneously) and *get_A_B( )*, for allocation (2). Fig. 4 shows C codes of the same access functions for allocation (1). As shown in these codes, the allocation of device parameters on I/O registers affects the code size and execution time of its associated I/O device driver. As reported in [8], these low-level codes have been found to represent up to 30% of a device driver. Their size and performance inevitably become important issues for I/O intensive embedded systems.

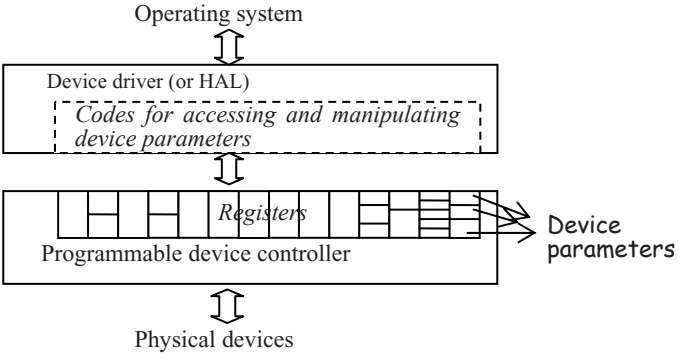


Fig. 1. Interface hardware and software for an I/O device

In traditional design flow, the development of device drivers can not begin until the device parameter allocation is fixed. In the proposed HW/SW codesign flow, a software programmer can write the device driver using pre-defined parameter-access functions such as *set_B( )*. However, the real C codes of these access functions depend on the physical allocation of parameters. This paper presents a novel design methodology that allows a designer to seek an allocation that reduces the codes of parameter-access functions or the number of I/O registers when developing device drivers. The exact allocation with the minimum cost under constraints is formulated as a zero-one integer linear programming problem. The HW/SW codesign approach is favorable in development of embedded systems. The formulations of an allocation with minimum software or hardware costs are derived. Heuristic algorithms based on iterative refinement are proposed to explore the optimization. The proposed design methodology were implemented in C language and evaluated with a set of real devices. Compared with current industrial designs, we can obtain design alternatives that reduce both software and hardware costs. Furthermore, the results also indicate design spaces for various application features.

To the best of our knowledge, the paper [5] should be the first work to investigate how the parameter allocation can affect the performance of drivers and to try finding an optimal allocation. However, it does not address hardware optimization. Furthermore, it does not allow parameters belonging to different groups to share a register. Other related works handling the synthesis of device drivers or interface codesign all assume

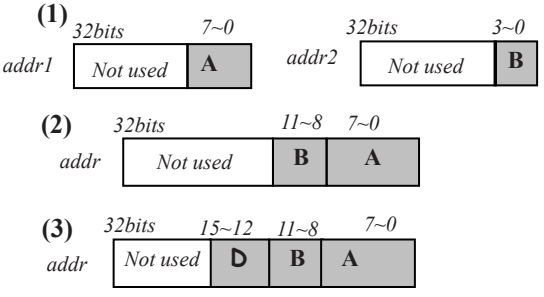


Fig. 2. Three different allocations for a device-parameter group {A, B}, where in case (3) the parameter D belongs to another group

```
void set_B(int val) {
      int temp;
      temp = io_read (addr) ;
      temp = temp & 0xfffff0ff;
      temp = temp | (val << 8) ;
      io_write(temp, addr);
}//Cost=2C1+2C2

void set_A_B(int val1, int val2) {
      int temp;
      temp= val1;
      temp = temp | (val2 << 8);
      io_write (temp, addr) ;
}// Cost= C1+2C2
```

```
void get_B(int *val) {
      int tmp;
      temp = io_read(addr);
      *val = (temp>>8) & 0x0f ;
}//Cost=C1+C2

void get_A_B(int *val1, int *val2) {
      int temp;
      temp = io_read(addr);
      *val1 = temp & 0xff ;
      *val2 = (temp>>8) & 0x0f ;
}//Cost=C1+2C2
```

Fig. 3. The C codes *set_B ( )*, *set_A_B( ), get_B( )* and *get_A_B( )* for device parameters A and B which are allocated in the same register, i.e. allocation (2) in Fig. 2

```
void set_B(int val) {
      io_write (val, addr2) ;
}//Cost=C1
```

```
void get_B(int *val) {
      *val=io_read (addr2) ;
}//Cost=C1
```

```
void set_A_B(int val1, val2) {
      io_write (val1, addr1) ;
      io_write (val2, addr2) ;
}//Cost=2C1
```

```
void get_A_B(int *val1, int *val2) {
    *val1 = io_read(addr1);
    *val2 = io_read (addr2) ;
}//Cost=2C1
```

Fig. 4. The C codes *set_B ( )*, *set_A_B( ), get_B( )* and *get_A_B( )* for device parameters A and B which are allocated in different registers, i.e. allocation (1) in Fig. 2

the allocation of device parameters are pre-fixed. F. Merillon et al. [8] address the automatic synthesis of such low-level codes from a higher level specification, called as Devil language. Recent works [1, 9, 10] extend the language's descriptive capability

and try to automatically synthesize more dedicated parts of a device driver. P. Chou et al. [2] propose an interface HW/SW cosynthesis work, which synthesizes driver codes as well as glue hardware logic to connect I/O controllers. G. Gognoit et al. proposed communication synthesis and HW/SW integration for Embedded System Design in [4].

The next section defines the software and hardware cost models. Section 3 formulates the exact solution of a parameter allocation with minimum costs. The whole design methodology and tools are presented in Section 4. Experimental results are shown in Section 5. The final section draws conclusions.

2 Cost Models

The proposed system allows users to trade off the HW cost for an I/O controller and the SW cost for its associated device driver. The HW and SW cost modes are defined in this section. Based on the observations on the C codes in Fig. 3 and Fig. 4, we summarize the software costs for different accesses and allocations in Table 1. A parameter can be allocated in three types of registers: (1) *single*: the register contains the parameter only; (2) *shared*: the register contains more than one parameter and all parameters in it belong to the same group; (3) *group-shared*: the register contains more than one parameter and the parameters in it are from different groups. The allocation (3) in Fig. 2 is such an example. The I/O access functions include reading (writing) an individual parameter and reading (writing) K parameters of the same group simultaneously. Most access functions for shard register and group-shared register are the same, except writing data to some parameters of a group, in that accessing a group-shared register needs one more I/O read access for retaining the values of parameters belonging to other groups.

Table 1. Software cost of various I/O access for different allocations of device parameters

Allocations / Accesses	Single register	Shared register	Group-shared register
Read an individual	C_1	$C_1+ C_2$	$C_1+ C_2$
Write an individual	C_1	$2\ C_1+ 2C_2$	$2\ C_1+ 2C_2$
Read K parameters of a group	KC_1	$C_1+ KC_2$	$C_1+ KC_2$
Write K parameters of a group	KC_1	$C_1+ KC_2$	$2C_1+(K+1)C_2$

C_1: Execution time (or instruction length) of an I/O access instruction
C_2: Execution time (or instruction length) of a bit-manipulation instruction

The total software cost under our consideration also depends on the execution numbers of these I/O accesses. These numbers are application-specific. The number of times a driver reads (writes) an individual parameter X_i in a period is given as *read frequency,* f_i^r (*write frequency,* f_i^w). The number of times a driver reads (writes) the

whole group is given as *group-read frequency* f_g^r, *(group-write frequency*, f_g^w). Given these access profiles, the software cost can be determined by the allocation of device parameters. In this work, the software cost can be code size or execution time. If code size is the main concern of software cost, the execution number means the number of parameter-access functions used as in-line functions.

Register index	**Variables allocated in this register**
Register *1*	$....X_2X_1$
….	….
Register *p*	$X_{m_p}X_{m_p-1}....$
Register *p+1*	$....X_{m_p+2}X_{m_p+1}$
….	….
Register *p+q*	$X_{m_p+m_q}X_{m_p+m_q-1}....$
Register *p+q+1*	$X_{m_p+m_q+1}$
….	….
Register $n-m_p-m_q$	X_n

Fig. 5. The physical allocation of $\Phi(V_g, m_p, m_q, p, q)$

The following defines an allocation for a parameter group, $V_g = \{ X_1, X_2,..., X_n \}$:

Definition 1 (Allocation $\Phi(V_g, m_p, m_q, p, q)$): For a device parameter group, $V_g = \{ X_1, X_2,..., X_n \}$, an allocation $\Phi(V_g, m_p, m_q, p, q)$ denotes that there are m_p parameters of V_g allocated in p shared registers and m_q parameters of V_g allocated in q group-shared registers, and the remaining $|V_g|-m_p-m_q$, parameters are allocated in single registers. Without loss of the generality, let $X_1X_2...X_{m_p}$ be in the shared registers, $X_{m_p+1}X_{m_p+2}...X_{m_p+m_q}$ *be in the group-shared registers,* and $X_{m_p+m_q+1}...X_n$ be in single registers.

Figure 5 illustrates the physical layout for $\Phi(V_g, m_p, m_q, p, q)$. Given access profiles, we can calculate the software cost as follows:

Definition 2 (Software Cost Function $C_s(\Phi)$): Given f_g^r, f_g^w, f_i^r and f_i^w, $i= 1,..., n$, for a device parameter group $V_g = \{ X_1, X_2,..., X_n \}$, the software cost $C_s(\Phi)$ of an allocation $\Phi(V_g, m_p, m_q, p, q)$ equals

$$
\begin{aligned}
C_s =& (f^r_{m_p+m_q+1}+....+f^r_n+f^w_{m_p+m_q+1}+....+f^w_n)C_1\\
&+(f^r_g+f^w_g)(n-m_p-m_q)C_1\\
&+(f^r_1+f^r_2....+f^r_{m_p})(C_1+C_2)+(f^w_1+f^w_2+...+f^w_{m_p})(2C_1+2C_2)\\
&+(f^r_g+f^w_g)(pC_1+m_pC_2)\\
&+(f^r_{m_p+1}+f^r_{m_p+2}....+f^r_{m_p+m_q})(C_1+C_2)+(f^w_{m_p+1}+f^w_{m_p+2}+...+f^w_{m_p+m_q})(2C_1+2C_2)\\
&+f^r_g(qC_1+m_qC_2)+f^w_g(q(2C_1+C_2)+m_qC_2)
\end{aligned}
$$

The first two rows in the above equation calculate the costs in the first column of Table 2. The third and fourth rows calculate the costs in the second column. The final two rows calculate the costs in the third column. Generally, the proposed system handles each parameter group separately since the register sharing between different groups always increases software cost. The following shows the software cost function for the case that a group does not share any register with other groups.

Definition 3 ($C_s(\Phi)$ for $\Phi(V_g, m, 0, p, 0)$): Given f_g ($=f_g^r + f_g^w$), f_i^r and f_i^w, $i= 1,\ldots, n$, for a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$, the software cost $C_s(\Phi)$ for an allocation $\Phi(V_g, m, 0, p, 0)$ equals

$$
\begin{aligned}
&(f_{m+1}^r + \ldots + f_n^r + f_{m+1}^w + \ldots + f_n^w)C_1 \\
&+ (f_1^r + f_2^r \ldots + f_m^r)(C_1 + C_2) + (f_1^w + f_2^w + \ldots + f_m^w)(2C_1 + 2C_2) \\
&+ f_g (n-m)C_1 \\
&+ f_g (pC_1 + mC_2)
\end{aligned}
$$

The hardware cost is defined as the number of registers used to allocate device parameters, as described in the following definitions.

Definition 4 (Hardware Cost Function $C_h(\Phi)$): For a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$, the hardware cost $C_h(\Phi)$ of an allocation $\Phi(V_g, m_p, m_q, p, q)$ equals $p + q + n - m_p + m_q$.

Definition 5 ($C_h(\Phi)$ for $\Phi(V_g, m, 0, p, 0)$): For a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$, the hardware cost $C_h(\Phi)$ of an allocation $\Phi(V_g, m, 0, p,0)$ equals $p + n - m$.

3 Formulation of Exact Minimization

This work seeks an allocation with the minimal SW or HW cost. The exact solution assuming no group-shared register being used is formulated. The software cost can be formulated as a problem of zero-one integer linear programming by using binary decision variables with two indices : $Y=\{y_{i,j};\ i=1, 2,\ldots, n;\ j=1, 2,\ldots, \lambda\}$, where n is the number of parameters, and λ is the upper bound of the register used. The $y_{i,j} = 1$ (0) means that the device parameter X_i is (is not) allocated in the register j. First, the number of shared registers, p, is assumed to be fixed. In other words, at most $\lambda - p$ single registers are used. General cases are discussed later. Let B_i be the bit-length of device parameter X_i, and RL the width of register. The registers numbered 1, 2,…, p denote shared registers. The problem now can be stated as follows:

Minimize software:

$$
\sum_{j=1}^{p} f_g C_1 + \sum_{i=1}^{p} \sum_{i=1}^{n} \left[(C_1 + C_2)\left(f_i^r + 2f_i^w\right) Y_{i,j} + f_g C_2 Y_{i,j} \right] + \sum_{j=p+1}^{\lambda} \sum_{i=1}^{n} C_1 \left(f_i^r + f_i^w + f_g\right) Y_{i,j}
$$

Subject to:

$$(a)\ \sum_{j=1}^{\lambda} Y_{1,j} = 1\ ,\ \sum_{j=1}^{\lambda} Y_{2,j} = 1\ ,\ \ldots\ldots\ldots\ldots\ ,\ \sum_{j=1}^{\lambda} Y_{n,j} = 1$$

$$(b)\ \sum_{i=1}^{n} Y_{i,1} \geq 2\ ,\ \sum_{i=1}^{n} Y_{i,2} \geq 2\ ,\ \ldots\ldots\ldots\ldots\ ,\ \sum_{i=1}^{n} Y_{i,p} \geq 2$$

$$(c)\ \sum_{i=1}^{n} Y_{i,p+1} \leq 1\ ,\ \sum_{i=1}^{n} Y_{i,p+2} \leq 1\ ,\ \ldots\ldots\ldots\ldots\ ,\ \sum_{i=1}^{n} Y_{i+\lambda} \leq 1$$

$$(d)\ \sum_{i=1}^{n} B_i X_{i,1} \leq RL\ ,\ \sum_{i=1}^{n} B_i X_{i,1} \leq RL\ \ ,\ \ldots\ldots\ ,\ \sum_{i=1}^{n} B_i X_{i,\lambda} \leq RL$$

Condition (a) means that any parameter must be allocated in exactly one register. Condition (b) means that the number of parameters allocated in a shared register must be at least 2. Condition (c) means that the number of parameters allocated in a single register must not be greater than 1 (but could be zero). Condition (d) means that the sum of the bit lengths of parameters in a register must not be larger than the length of a register.

In the hardware side, the exact cost minimization under no constraint is just a bin packing problem [7]. Under a software constraint, the exact minimization can also be formulated as a zero-one ILP problem. We still need the decision variable Y. Furthermore, we define a binary decision variable Z_j, $j=\{1, 2,\ldots, \lambda\}$, where $\lambda=n-p$, is the maximum number of registers probably used. The $z_{,j} = 1$ (0) means that the j^{th} register is (is not) used. Let SC be the given software-cost constraint. The symbols B_i, X_i, and RL are used as for software minimization formulation. The problem now can be stated as follows:

Minimize hardware: $\sum_{j=1}^{p} Z_j + \sum_{j=p+1}^{\lambda} Z_j$

Subject to:

(a), (b), (c) (the same as ones for the software part)

$$(e)\ \sum_{i=1}^{n} B_i X_{i,1} \leq RL \cdot Z_1\ \ ,\ \sum_{i=1}^{n} B_i X_{i,1} \leq RL \cdot Z_1\ \ ,.,\ \sum_{i=1}^{n} B_i X_{i,\lambda} \leq RL \cdot Z_\lambda$$

$$(f)\ \sum_{j=1}^{p} f_g C_1 + \sum_{i=1}^{p} \sum_{i=1}^{n} \left[(C_1 + C_2)\left(f_i^r + 2 f_i^w\right) Y_{i,j} + f_g C_2 Y_{i,j} \right] + \sum_{j=p+1}^{\lambda} \sum_{i=1}^{n} C_1 \left(f_i^r + f_i^w + f_g\right) Y_{i,j}\ \leq SC$$

The former three conditions are the same as those for software minimization. Condition (e) constrains the total bit length of parameters in a register. Condition (f) is the software constraint. Based on the above formulations, an ILP-solver can be used to obtain an exact minimization if p shared registers are used. To obtain a global exact minimization, the ILP-solver must be run $n/2$ times with $p = 1, 2,\ldots.n/2,$ and the best result then chosen.

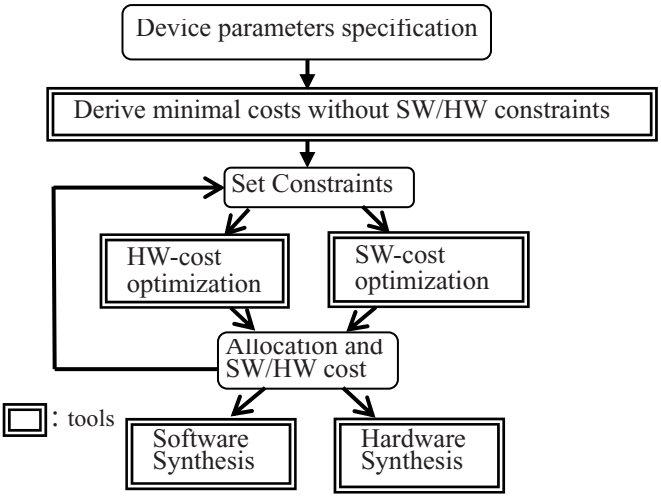


Fig. 6. An HW/SW codesign flow for I/O device controllers

4 Design Methodology

The flow of the proposed design methodology is shown in Fig. 6. The device-parameters specification, defines the bit length of each device parameter, the members of each parameter group and access profiles f_g^r, f_g^r, $f_i^{\ r}$ and $f_i^{\ w}$. The bit length of a parameter and groups' members are fixed, while the access profiles are determined by a dedicated application. Given the device-parameter specification, the system first derives two allocations on the assumption that the software (hardware) constraint is unlimited when minimizing the hardware (software) cost. One has a minimal software cost and the other minimal hardware cost. They are two extreme solutions of the design space. Then a user can give a hardware constraint (or a software constraint) and obtain an allocation having a minimal software cost (or hardware). The inner loop allows a user to iteratively refine the solution to meet a certain purpose. The acceptable allocation for all device parameters can then be fed into a software synthesis tool [6, 8], which generates the low level codes of parameter-access functions. Furthermore, the allocation will be used by a hardware synthesis tool to generate I/O registers.

In both HW and SW optimizations, we first handle each device parameter group separately. Then we allow parameters in different groups to share registers if the following situations occur: (1) if no solution exists to meet the hardware constraint and (2) the hardware cost can be further reduced while still meeting the software constraint.

Although an ILP solver can derive the solution with exact minimum cost, it is time-consuming for handling large cases. Heuristic approaches are presented here. The heuristic algorithm starts from an initial allocation with the smallest number of registers, which corresponds to an exact solution of a bin-packing problem. Then, the allocation is iteratively refined to obtain a lower cost until the constraint is violated or no further improvement can be obtained. The strategy is applied to both software and

hardware optimization. Each refinement process relies on proven lemmas, which have been presented in [5]. To show these lemmas, a new notation is defined as follows:

Definition 6 ***Φ'(V_g, V', m-k, 0, p-s, 0)*****:** An allocation $\Phi'(V_g, V', m-k, 0, p-s, 0)$ is derived from $\Phi(V_g, m, 0, p, 0)$ by reallocating a set of parameters V', $V'=\{X_j \mid j \in 1,\ldots, m\}$ and $|V'|= k$, to single registers. The remaining parameters are repacked into $p-s$ shared registers, where $k \geq 1$, $s \geq 0$ and $k \geq s$.

The lemmas exploited in our approach are described in the followings, whose proofs can be directly derived by subtracting $C_s(\Phi)$ from $C_s(\Phi')$.

Lemma 1: Given f_g, f_i^r and f_i^w, $i= 1,\ldots,n$, for a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$ and an allocation $\Phi(V_g, m, 0, p, 0)$, if $\Phi'(V_g, V', m-1,0, p-1,0))$ exists, then $C_s(\Phi')$ is smaller than $C_s(\Phi)$.

Lemma 2: Given f_g, f_i^r and f_i^w, $i= 1,\ldots, n$, for a device parameter group $V_g = \{ X_1, X_2,\ldots, X_n \}$ and an allocation $\Phi(V_g, m, 0, p, 0)$, $C_s(\Phi'(V_g, \{X_j\}, m-1, 0, p, 0)) < C_s(\Phi(V_g, m, 0, p, 0))$ if and only if $\dfrac{C_2 f_j^r + (C_1+2C_2) f_j^w}{C_1 - C_2} > f_g$.

The SW-optimization algorithm can be found in [5]. The HW-optimization part is shown in Fig. 7. An initial allocation with the minimum number of registers is derived by an exact bin-packing procedure, which uses a branch-and-bound approach as proposed by Martello and Toth [7]. Starting from the initial allocation, the solution is iteratively refined. If the software cost of the initial allocation is larger than the given software constraint (SC), the algorithm runs *bin_packing_maximize_single()*, which tries to derive a better allocation according to Lemma 1. Then, if the constraint is still not met, the Lemma 2 is applied.

```
MinmizeHCostUnderSC (VarList, f_g , f^r List, f^wList, SC) {
//SC : Software Constraint
// Let Q(j) = (C2 f_j^r + (C1+2C2) f_j^w) / (C1 - C2) > f_g
//Aloc: An allocation of device variables
// Aloc-> ShVarlist: The variables packed in shared registers
// Aloc->RN : The number of registers used in the allocation
bin_packing(Varlist, & Aloc ); //initial
if ( SCost(Aloc) > SC )
bin_packing_maxmize_single(Varlist, &Aloc);
while ( SCost(Aloc) > SC) {
if Q(j) is the largest in Aloc->ShVarlist and Q(j) >1
   move the Var_j in  Aloc->ShVarlist to a single register,
  if no such j exists, return no-solution;
   }
 return Aloc;
}
```

Fig. 7. The proposed algorithm for HW-Cost minimization under an SW constraint

The solution obtained by the algorithm in Fig. 7 can be further reduced by using group-shared registers. However, as shown in Table 1, the use of group-shared registers increases the software cost if $f_g^w > 0$ for some group to be allocated in the group-shared register. Hence, the proposed algorithm firstly sorts groups according to the f_g^w. Then, two registers used by two different groups which have smaller f_g^w are selected to be merged. The merging is tried repeatedly until the software constraint is violated. Fig. 8 shows the procedure.

UseGroup-SharedRegisters( ) {
//Assume there is K different device-parameter groups. Without loss of generality, let groups GP_1, GP_2,..., GP_k, *be the increasing list of* f_g^w.
// T = GP_1
for i=2 to K {
try to merge any pair of registers (x, y) to be a register, where $x \in T$, $y \in GP_i$, *and both are a shared or group-shared register ;*
calculate SW-cost;
if the software constraint is violated, discard the merging and return;
$T = T \cup GP_i$;
}
}

Fig. 8. An algorithm to further improve HW-cost by using group-shared registers

5 Experimental Results

The proposed design methodology was implemented in C. Experimental results were obtained for several industrial devices. These devices are originally used to show driver synthesis in [8], and specified in Devil language. Their specifications define the bit length of each device parameter and the members of a group according to industrial IC documents. The access profiles (i.e. f_g^r, f_g^w, f_i^r and f_i^w) are given by ourselves. The value of the SW cost depends on the given values of f_g^r, f_g^w, f_i^r and f_i^w. To illustrate the influence of access profiles, two cases were evaluated: (a) $f_g^r + f_g^w >> f_i^r + f_i^w$ and (b) $f_g^r + f_g^w << f_i^r + f_i^w$. For simplicity, in both cases, we let all groups have the same values for f_g^r and f_g^w. And all parameters have the same values for f_i^r and f_i^w.

Table 2 shows the experimental results. The "industrial" column shows the hardware and software costs of the allocation in original Devil specifications [8]. The two costs are used as the hardware and software constraints. Under the two constraints, software and hardware optimizations were derived by the proposed heuristics. The column SOP and HOP indicate the results assuming no group-shared register being used. The results show that an allocation with minimal software (hardware) cost is not necessarily to use maximal hardware (software cost). Comparing the experimental results with industrial designs, the proposed approach can obtain design alternatives that reduce both software and hardware costs. The "GHOP" column shows the HW costs which are derived by applying *UseGroup-SharedRegisters()* procedure. In several cases, the HW-cost can be further reduced. As for the program time, the results for all examples can be obtained within 30 seconds on a 2.4 GHz P4-based PC.

Table 2. Experimental results

Devices	G/N(S)[2]	$f_g^r = f_g^w = 10;\ \forall i, f^r_i = f^w_i = 1$				$f_g^r = f_g^w = 1;\ \forall i, f^r_i = f^w_i = 10$			
		Industrial	SOP	HOP	GHOP	Industrial	SOP	HOP	GHOP
		HW cost (SW cost)[1]				HW cost (SW cost)			
BusMr	1/8(8)	2 (496)	2 (342)	2 (342)	2 (342)	5 (802)	5 (678)	5 (790)	5 (790)
IDE	3/38(22)	11 (2506)	5 (1490)	5 (1516)	3 (2116)	11 (4468)	11 (4206)	9 (4380)	7 (4416)
X11	5/47(23)	13 (3534)	11 (2094)	11 (2146)	11 (2146)	13 (6234)	13 (5916)	11 (6214)	11 (2146)
8290	7/107(26)	48 (11344)	17 (4242)	17 (4249)	15 (5874)	48 (10678)	48 (9698)	37 (10384)	35 (10440)
83905	7/103(32)	45 (13398)	17 (4214)	17 (4244)	14 (4404)	45 (11022)	45 (9920)	46 (10196)	43 (10212)
Total		122 (31278)	52 (12382)	52 (12497)	43 (14540)	122 (33204)	122(31818)	108(31964)	96(27214)
Compared to the industrial		1 (1)	0.43 (0..40)	0.43 (0.40)	0.35 (0.46)	1 (1)	1 (0.96)	0.89 (0.96)	0.79 (0.82)

1. *`The SW-cost is calculated by assuming $C_1=3$ and $C_2=1$.*
2. *G/N(S) : # of group / # of all parameters (# of parameters in the maximal group).*
3. *Industrial: The cost of the original allocation in the Devil specification.*
4. *SOP: SW minimization using the industrial HW cost as the constraint.*
5. *HOP: HW minimization using the industrial SW cost as the constraint.*
6. *GHOP: HW minimization by allowing different groups to share a register.*

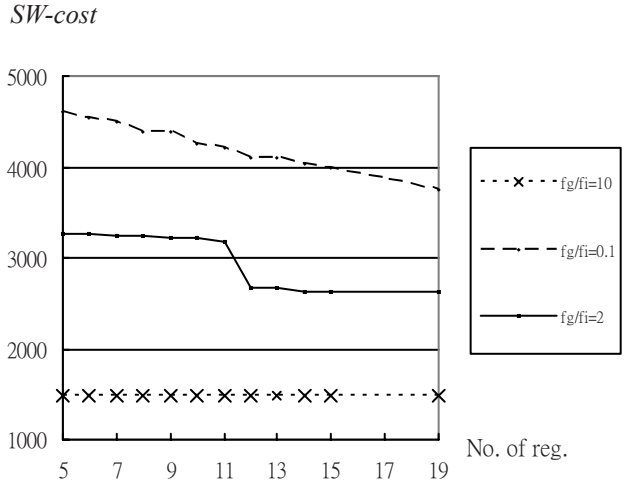


Fig. 9. SW-optimizations under various HW constraints (IDE example)

To investigate the design spaces for various application features, we derived the allocations with the minimal SW cost under various HW constraints. Figure 9 shows such an investigation using IDE as an example. The three curves show the results for three different frequency ratios of $f_g^r + f_g^w$ to $f_i^r + f_i^w$. Some observations are obtained. In case of $f_g/f_i=10$, the solution with the minimal SW-cost generally is also the one with the minimal HW-cost. In case of $f_g/f_i=0.1$, the SW-cost is almost inversely proportional

to the HW-cost. In case of $f_g/f_i=2$, the result shows that even more hardware can be used, the SW-cost is not further reduced.

6 Conclusion

We have proposed a HW/SW codesign methodology that allows a designer to seek a parameter allocation that reduces the software or hardware cost during the development of a programmable I/O controller and its associated device driver. The approach is favorable in development of embedded systems. The exact allocation problems under constraints are formulated as zero-one integer linear programming problems. Heuristic algorithms based on iterative refinement are also proposed. The proposed design methodology was implemented in C language. Compared with current industrial designs, the system can obtain design alternatives that reduce both software and hardware costs. Furthermore, the results also indicate design spaces for various application features.

Acknowledgements. The authors would like to thank Taiwan NSC for financially supporting this research under Contract No. NSC 94-2215-E-030-007.

References

1. Conway, C.L., Edwards, S.A.: NDL: A Domain -Specific Language for Device Drivers. In: Proceeding of ACM LCTES, pp. 30–36 (2004)
2. Chou, P., Ortega, B.R., Borriello, G.: Interface Co -synthesis Techniques for Embedded Systems. In: Proceedings of IEEE/ACM ICCAD, pp. 280–287 (1995)
3. Givargis, T., Vahid, F.: Parameterized System Design. In: IEEE/ACM International Workshop on Hardware/Software Codesign, CODES, pp. 98–102 (2000)
4. Gognoit, G., Auguin, M., Bianco, L.: Communication synthesis and HW/SW integration for Embedded System Design. In: Proceeding of 6th international workshop on HW/SW codesing, pp. 49–53 (1998)
5. Lin, K.J., Huang, S.S., Chen, S.W.: A hardware/ software codesign approach for programmable IO devices. In: Proceedings of the 15th ACM Great Lakes Symposium on VLSI, pp. 323–327 (2005)
6. Lin, K.J., Chen, J.L.: An Extension of C Preprocessor Directives for Device Programming. In: International Computer Symposium, Taiwan, pp. 1279–1284 (2004)
7. Martello, S., Toth, P.: Knapsack problems. Wiley, Chichester (1990)
8. Mérillon, F., Muller, G.: Dealing with hardware in embedded software: A general framework based on the Devil language. In: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems, pp. 121–127 (2001)
9. Wang, S., Malik, S., Bergamaschi, R.A.: Modeling and Integration of Peripheral Devices in Embedded Systems. In: DATE, pp. 136–141 (2003)
10. Zhang, Q.L., Zhu, M.Y., Chen, S.Y.: Automatic Generation of Device Drivers. ACM SIGPLAN Notices, 60–69 (2003)

A Semantic P2P Framework for Building Context-Aware Applications in Multiple Smart Spaces

Tao Gu[1], Hung Keng Pung[2], and Daqing Zhang[1]

[1] Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore
[2] National University of Singapore, 3 Science Drive 2, Singapore
tgu@i2r.a-star.edu.sg, punghk@comp.nus.edu.sg,
daqing@i2r.a-star.edu.sg

Abstract. Context information has emerged as an important resource to enable autonomy and flexibility of ubiquitous applications. The widespread use of context information necessitates an efficient lookup service in a wide-area network over multiple smart spaces. In this paper, we propose a context lookup framework based on a semantic peer-to-peer network to support the building of context-aware applications in multiple smart spaces. Collaborative context-aware applications that utilize different context information in multiple smart spaces can be easily built by invoking a pull or push service provided by our framework. We describe the design of our system, demonstrate the development process of context-aware applications, and report the measurements obtained from our prototype.

Keywords: Context lookup, semantic peer-to-peer network, context-aware applications, multiple smart spaces.

1 Introduction

The recent convergence of ubiquitous computing and context-aware computing has seen a considerable rise in interest in various context-aware applications. These applications exploit various aspects of the contextual environment to offer services, present information, tailor application behavior and trigger adaptation, based to the changing context.

Context information gathered from various sensor systems is the basis for enmeshing ubiquitous computing into our daily lives and exhibiting the autonomy of applications. Storing and acquiring such information in a single smart space can be easily handled by a centralized database server. The server can provide fast response to a lookup query. However, handling large-scale context information over multiple smart spaces requires an appropriate context lookup architecture. Distributing database servers to multiple smart spaces in a wide-area network does provide a scalable and reliable solution. However, this approach requires a significant investment on servers, the bandwidth costs of storing and updating context information, and administration restrictions.

Emerging Peer-to-Peer (P2P) approaches have been proposed to overcome some of these obstacles, and providing potential solutions for a large-scale distributed lookup

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 553–564, 2007.
IFIP International Federation for Information Processing 2007

system. This paper proposes a semantic P2P framework to support storing and acquiring context information in multiple smart spaces. In this framework, context data is stored in a context producer where it was generated. Each context producer is only responsible for managing its local context data that may be acquired from the sensors attached. For an efficient context lookup, we design and implement a semantic P2P overlay network in which context data is organized and retrieved according to their semantics. In this network, context producers are arranged in such a way that those with semantically similar data are grouped together so that a context query can be routed efficiently. Many existing or potential collaborative context-aware applications can use our framework, especially those collaborative context-aware applications over multiple smart spaces. For examples, Family Intercom [5] – an advanced communication application between multiple smart homes, is able to identify the caller and the recipient and mediate the initiation of the audio conversation. In health care applications, a tele-monitoring application tracks a patient wherever he/she goes or a tele-medical record application can provide anywhere availability of personal medical history.

The rest of the paper is organized as follows. We describe the system architecture and its details in Section 2. We then present collaborative context-aware applications in Section 3, and report the results obtained from our prototype in Section 4. We discuss related work in Section 5. Finally, we conclude the work in Section 6.

2 System Architecture

2.1 Overview

Our framework consists of many individual nodes called ContextPeers, which act as context producers. Users and context-aware applications act as context consumers to obtain context data by submitting their queries to ContextPeers and receive the results. ContextPeers are self-organized into a semantic P2P network [6] for supporting P2P search. ContextPeers exploits semantic P2P overlay as the underlying network layer and extends it with RDF-based context storage, context queries and context subscription.

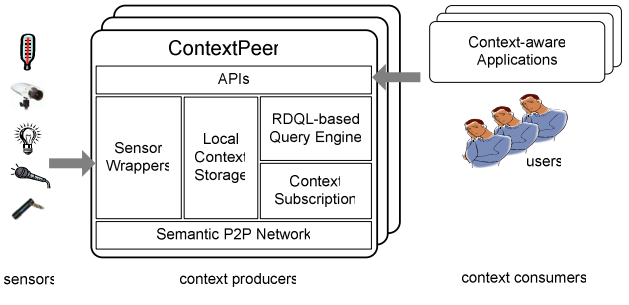


Fig. 1. The architecture of ContextPeers

As shown in Fig. 1, each ContextPeer consists of five components: the semantic P2P network layer, the sensor wrappers, the local context storage, the RDQL [10] based query engine and the context subscription. The sensor wrappers capture various sensor data from physical or virtual sensors, and convert raw (i.e., direct sensor output) format into an RDF-based data model (i.e., in the form of RDF triples) and store the triples into the local context storage. The RDQL-based query engine parses and resolves context queries from users or applications. The context subscription registers subscription requests and notifies context consumers when context changes occur. The semantic network layer is responsible for network construction and maintenance, and query routing. Application developers utilize a set of APIs provided by our framework to access the functionalities of ContextPeers and build context-aware applications. The class diagram containing the major classes in a ContextPeer is shown in Fig. 2.

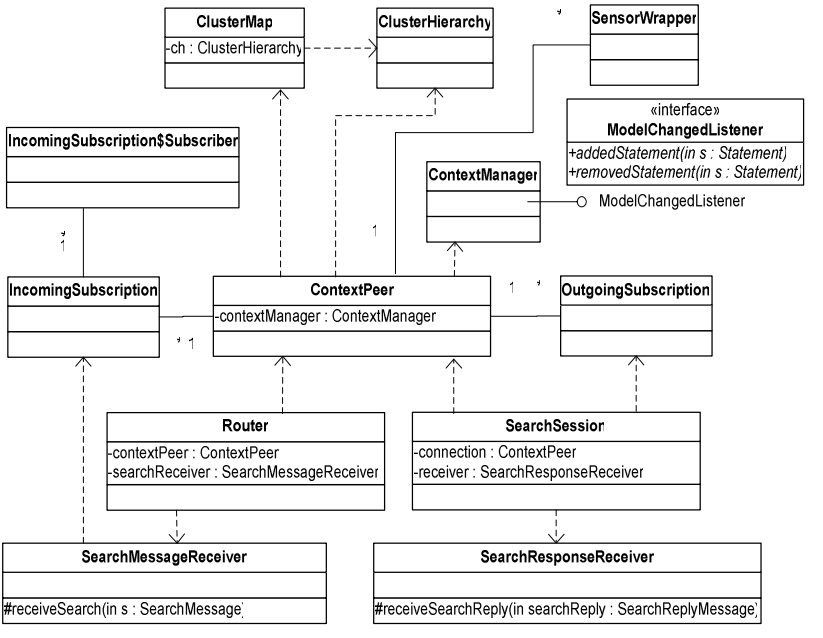


Fig. 2. Class diagram of a ContextPeer

2.2 Data Model

In our system, we use an RDF-based context model to represent context data. RDF provides a universal platform for representing resources and asserting relations between resources in a machine-readable and machine-understandable way. Each RDF statement is represented as a triple of the form <*subject*, *predicate*, *object*>. We adopt a hierarchical context ontology model defined in [6] which consists of a shared upper ontology and a set of domain-specific ontologies. The upper ontology defines common concepts, and it is shared among all ContextPeers. Each ContextPeer can

define its own concepts in its low-layer ontologies which extend the leaf concepts in the upper ontology. Different ContextPeers may store different sets of low-layer ontologies based on their applications' needs. This design approach offers application developers the flexibility to define domain knowledge which is specific to their applications.

2.3 Sensor Wrapper

A ContextPeer acquires context information from the various sensors attached to. We create an appropriate wrapper for each type of sensors. This approach can avoid explicit binding of the application to a particular underlying context sources technology. Wrappers capture sensor data, and convert them into RDF-based context data. A ContextPeer selects a set of wrappers based on its contextual interests, subscribes to the wrappers and gets updated.

We use the *SensorWrapper* class to construct a wrapper. A *SensorWrapper* object instance is associated with a set of *ContextTriple* objects specifying the provided context and an *UpdateHandler* object implementing actions for context update. For example, an RFID-based location sensor wrapper is able to convert sensor data `<RoomID RFID-John>`, where `RoomID` represents the ID of a bedroom and `RFID-John` represents the RFID sensor attached to a person – John, to the RDF statement `<John locatedIn Bedroom>`, representing that John is currently located in the bedroom.

2.4 Local Context Storage

Each ContextPeer maintains a local repository for storing context data. The repository stores context data ontologies, static context data and sensed context data. Static context data refers to context data that does not change frequently, such as the spatial information of buildings (e.g., John's bedroom is located in John's house). Sensed context data refers to data obtained from sensors (e.g., John is located in his bedroom). Such data is typically dynamic and changes frequently.

The *ContextManager* class built on Jena's *Model*s [8] is responsible for managing the repository. It provides methods to add or remove context data and answer context query, and also provides a set of operations to combine ontologies or context data. Since sensed context data changes frequently, the *ContextManager* class attaches a *ModelChangedListener* to the sensed context data *Model* to monitor these changes. This is especially important for responding to incoming subscribed queries.

2.5 Data Mapping

Upon creation, a ContextPeer needs to decide which cluster to join in the semantic P2P overlay network. This is done by using an ontology-based semantic mapping technique to map a ContextPeer's local data to semantic cluster(s) as defined in the context ontology, and count the number of triples corresponding to each semantic cluster. This technique traces the hierarchy of OWL [7] classes and maps their predicates to the associated classes. We create two structures – *ClusterHierarchy* and

ClusterMap. We first map each triple to an OWL class using *ClusterMap*, and then map the triple to an appropriate semantic cluster using *ClusterHierarchy*. Let SCn_{sub} , SCn_{pred} , SCn_{obj} where n = 1, 2, . . . denote the semantic clusters extracted from the subject, predicate and object of a triple respectively (Note: unknown subjects/objects or variables are mapped to all). If the predicate of a triple is of type *ObjectProperty*, we obtain the semantic clusters using ($SC1_{pred} \cup SC2_{pred} \cup ... SCn_{pred}$) $\cap$ ($SC1_{obj} \cup SC2_{obj} \cup ... SCn_{obj}$). If the predicate of a triple is of type *DatatypeProperty*, we obtain the semantic clusters using ($SC1_{sub} \cup SC2_{sub} \cup ... SCn_{sub}$) $\cap$ ($SC1_{pred} \cup SC2_{pred} \cup ... SCn_{pred}$). The semantic cluster with the highest triple counts (called the major semantic cluster) is selected for the ContextPeer to join. To achieve this, each ContextPeer creates and maintains a *HashMap* and iterates through all the triples in its model. Upon successful execution, the method returns a vector containing all the semantic cluster IDs corresponding to all its local context data. The first element in this vector indicates the ID of the major semantic cluster.

2.6 Query Routing

We follow the principle of a small world network model [9] and extend it with clustering operations to build the semantic network. After obtaining the semantics of its local data, nodes are organized in such a way that those have semantically similar data are grouped together in a semantic cluster. As a node's data may correspond to multiple semantic clusters, a node joins its major semantic cluster and publishes the indices of its data (i.e., reference pointer) to its minor semantic clusters.

Routing Table Construction: Each node builds its routing table by creating a set of local contacts in its own cluster, a short-range contact in each of its neighboring clusters, and a small number of randomly chosen long-range contacts. Each newly joining node builds its routing table in the same way resulting in all the clusters being linked linearly in a ring fashion. As illustrated in Fig. 3, *Node 1* builds two local contacts (*Node 2* and 3) in *SC1*, two short-range contacts (*Node 4* and *5*) in *SC0* and *SC2* respectively, a long-range contacts (*Node 6*) in *SC5*, and publishes its indices to a random node (*Node 7)* in *SC3*.

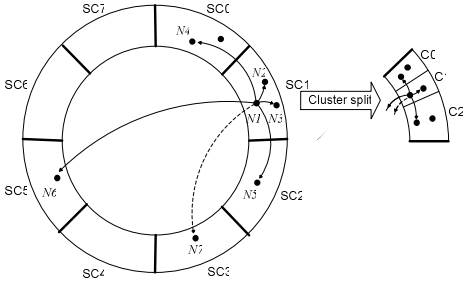


Fig. 3. Constrcution of the Semantic Network

Grouping context providers with similar data according to their major semantic clusters has the effect of minimizing the cost of node joining, leaving, and data changes. A node will stay in its major semantic cluster as long as the majority of data does not change. However, a large number of nodes in a semantic cluster may result in a scalability issue. We design the follow clustering operations to enable the network to scale to a large number of nodes.

Clustering Operations: When the number of nodes in a semantic cluster exceeds a certain size, cluster splitting occurs. Let M represent the maximum cluster size. If the size of a cluster exceeds M, the cluster is split into two. Each node maintains a *CurrentLoad* which measures its current load in terms of the number of triples and data indices it stores. When node x joins the network, it sends a join request message to an existing node, say y. If y falls into the same semantic cluster that x wishes to join, x joins the cluster by connecting to y if its cluster size is below M; otherwise y will direct the request to a node, say z, in the semantic cluster that x wishes to join, and x will connect to z if its cluster size does not exceed M. If the cluster size exceeds M, node y or z (called an initial node) will initiate the splitting process. The initial node first obtains a list of all the nodes in this cluster which is sorted according to their *CurrentLoads*. Then it assigns these nodes in the list to the two sub-clusters alternatively. After splitting, we obtain two clusters with relatively equal load. The initial node is also responsible for generating a new cluster *ID* for each of the two sub-clusters. To obtain a new cluster ID, each node maintains a *bit split pointer* which indicates the next bit to be split in the n-bit binary string. Initially, the *bit split pointer* points to the most significant bit of the n-bit string. When cluster splitting occurs, the bit pointed by the *bit split pointer* is split into 0 and 1 and move the pointer forwards to the next bit in the n-bit string. The same mechanism follows for the insertion of a new semantic cluster. A semantic cluster can be split into a maximum number of 2^n clusters. After splitting, a node updates its *cluster ID*, the *bit split pointer* as well as its local contacts and short-range contacts.

When the number of nodes in a cluster falls below a threshold, cluster merging occurs. When node x leaves the network, it first checks whether its cluster size has fallen below a threshold M_{min}. If the current size is above M_{min}, x simply leaves the network by transferring its indices to a randomly selected node in its cluster. Otherwise, this cluster needs to be merged into one of its neighboring clusters within the same semantic cluster. The leaving node triggers cluster merging which is an inversed process of cluster splitting.

Query Routing: The query routing process involves two steps: inter-cluster routing and intra-cluster routing. Upon receiving a query, node x first obtains the destination *Semantic Cluster ID* (denoted as D). This is done following the same mapping process as described in Section 2.5. Then node x will check whether D falls into its own semantic cluster by comparing D against the most significant m-bits of its *ClusterID*. If that is the case, x will flood the query to all the local contacts and also forward the query to its short-range contacts in its adjacent clusters corresponding to D. The forwarding processes are recursively carried out until all the clusters corresponding to D have been covered and all nodes in each of the clusters are reached.

If *D* falls into neither node *x*'s own cluster nor its adjacent semantic cluster, *x* will rely on its long-range contacts to route the query across clusters. To initiate a search, *x* obtains *D* based on a query and checks which cluster range (partitioned by *x*'s long-range contacts) *D* falls into. Then node *x* forwards the query to the closer semantic cluster through its long-range contact. If *D* is closer to SC_x, node *x* will forward the query across its adjacent cluster towards *D*.

2.7 Subscription

Other than a context query which pulls context data from the network, context consumers can issue a subscribed query to subscribe context data and be notified when data changes occur.

Upon receiving a subscription request, a ContextPeer attempts to match it against the context data in its base model. If the request's predicate is of type *DatatypeProperty*, the ContextPeer determines if its base model contains statements with the same *subject-predicate* pair as the request. Similarly, if the predicate is an *ObjectProperty*, the ContextPeer determines if its base model contains statements with the same *predicate-object* pair as the request.

Whenever a change occurs with respect to the subscription request, the *ModelChangedListener* informs the *ContextManager* of the RDF statement that has been added or removed. Subsequently, the *ContextManager* scans through all *IncomingSubscriptions* and identifies those that are affected by the change. This is done in the following manner: Let the added or removed RDF statement be $\langle subject_c, predicate_c, object_c \rangle$. Let the RDF triple pattern of a particular *IncomingSubscription*'s criteria be $\langle subject_{sr}, predicate_{sr}, object_{sr} \rangle$. Define the Boolean variable $isAffected_c$ as:

$isAffectedc = (subjectc == subject_{sr}) \text{ ^ } (predicate_c == predicate_{sr}) \text{ ^ } (object_c == object_{sr})$

where ^ denotes the logical AND operation. A variable can take the value of any arbitrary constant and is thus equal to any constant value. An *IncomingSubscription* is affected by a change *c* if $isAffected_c$ is true. For each affected *IncomingSubscription*, the *ContextManager* sends *QueryHit* messages to all its subscribers to supply them with the updated context data.

3 Application Development

We have fully implemented our framework in Java SDK 1.4.1, and also developed two context-aware applications: Tele-monitoring alert and Tele-medical record.

The Tele-monitoring alert application monitors a patient's health parameters and informs the relevant parties when abnormal signs are observed. The application scenario is illustrated in Fig. 4. Alan just had his heart bypass operation and is now discharged from his hospital and recuperating at home. The doctor gives him a wearable health monitoring device to track his pulse rate, temperature or the blood pressure of the wearer. The Health Monitor runs on ContextPeer with the connections to physical sensors. The HealthCare Assistant application runs in Alan's portable device (i.e., PDA). It tracks the health status of Alan by subscribing to the sensor

information provided by Health Monitor. When abnormal situation occurs, for example, HealthCare Assistant detects a drop in blood pressure and a decrease in pulse rate which it deduces could be a case of imminent heart failure, it immediately alert the Hospital Services Assistant in Alan's private hospital or a nearby emergency center. A health status report can also be transferred to the Hospital Services Assistant for the doctor to have more information to diagnose Alan's condition. The HealthCare Assistant will send a SMS to alert a caretaker of the situation.

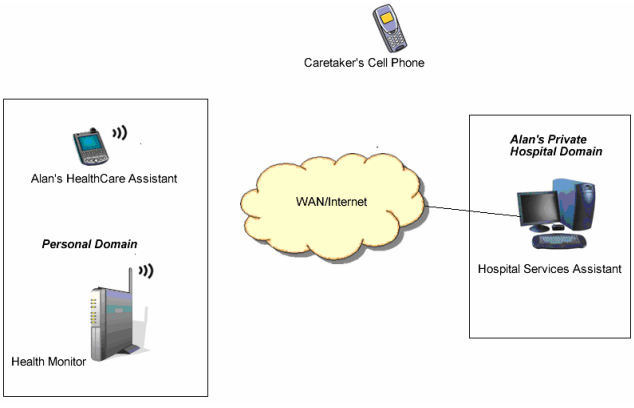


Fig. 4. Scenario illustration of the Tele-monitoring alert application

The Tele-medical record application allows for easy transfer of medical records from one hospital to another hospital facilitating the doctor's treatment of a patient. In the above scenario, upon reaching a nearby hospital or an emergency center, Alan's HealthCare Assistant automatically sends his personal information (i.e., name, age, gender, medicine allergies, etc) to a local Hospital Services Assistant. The Hospital Services Assistant checks and realizes Alan is in his first time visiting and hence the hospital does not have Alan's medical record. Thus the Hospital Services Assistant sends a request to Alan's HealthCare Assistant to request for his medical record. The HealthCare Assistant next sends a query to Alan's private hospital's Hospital Services Assistant to retrieve his latest medical record. After receiving the record, the HealthCare Assistant filter out any privacy information, e.g., his drug addiction ten years back, and forward the medical record to the local hospital's Hospital Services Assistant.

4 Prototype Measurements

We have deployed a prototype system to demonstrate the working principle of ContextPeers and assess practical issues. In this section, we report the measurement results obtained from our prototype testbed.

We set up the prototype testbed in a wide-area network. Most of the ContextPeers run on Pentium 800MHz desktop PCs with 256MB memory. We create a set of context ontologies and context data for each ContextPeer. Each ContextPeer stores

the upper context ontology and one or more domain-specific context ontologies. Before the evaluation starts, we need to place context ontologies and context data at each ContextPeer. The evaluation starts by connecting each ContextPeer to the network. The network is constructed when ContextPeers randomly join the network. A ContextPeer obtains the IP of an existing ContextPeer from the bootstrap server. We test the bootstrap process by connecting all the ContextPeers to the network in different joining sequences; hence, the structure of the network obtained may differ from one to another.

4.1 Bootstrapping

When a ContextPeer starts, it first goes through the semantic clustering mapping process to identify which semantic cluster to join. The mapping process is done by iterating each of the RDF data triples and identifying its corresponding semantic cluster. Then the ContextPeer chooses the major semantic cluster to join. On average, the program initialization process takes about 4.26 seconds, and the mapping process for each RDF data triple takes about 0.251 ms. The initialization process involves reading and merging the ontology files stored locally and generating internal data structures for mapping. It is done only once when a ContextPeer starts and is only repeated if there is a change in these ontologies. Upon joining the network, each node creates and maintains a set of peers in its routing table. The joining process involves initiating the Join message, connecting to those nodes in the JoinReply message received and registering its reference if needed. The results for different steps in the bootstrap process are summarized in Table 1.

Table 1. The results for the bootstrapping process

Processes	Average Time Taken
Program Initialization	4.26 s
Semantic Clustering Mapping	0.251 ms/RDF triple
Joining Process	2.56 s

4.2 Dynamic Characteristic

We evaluate the dynamic characteristic of the network in our prototype by forcing ContextPeers to join and leave different semantic clusters randomly. Cluster splitting/merging may occur when the cluster size is greater/lower than the default size. For testing the dynamic characteristic of the network, we introduce a parameter: ***T**ime-to-**S**tability (TS)*. We define the steady state of ContextPeer as the state in which a ContextPeer maintains live connections to the peers in its routing table. The steady state of a ContextPeer may collapse if one of the following events occurs:

- Its *short-range contacts* or *long-range contacts* leave the network or some of these peers change their major semantic clusters (due to their local data change).
- Its reference peer(s) leave the network or their major semantic clusters change.

Queries routing may be affected when ContextPeers are not in the steady state. The *TS* parameter is measured from the time when the steady state of a ContextPeer collapses until it reaches the steady state again. We measure the *TS* of the affected ContextPeers for different test cases and the results are summarized in Table 2 (note that no backup links are used in these cases).

Table 2. Results on *TS*

Test Cases (without backup links)	**Average *TS***
Case 1: The short range contacts or long range contacts leaves the network or changes its major cluster or cluster splitting/merging occurs	271 ms per connection
Case 2: Reference hosting nodes leave/change	87 ms per reference

In a highly dynamic network, peers leave and join frequently; this may result in relapse rate very high. A high relapse rate may affect query routing in the network. To prevent this, we use a backup link for each type of connections. Once the steady state collapses, a ContextPeer can switch to the backup link immediately for the affected connection. With this backup scheme, we can minimize the disruption to query routing in the highly dynamic network where peers frequently leave and join.

4.3 Response Time Analysis

In this experiment, we analyze the important factors that affect the query response. We randomly select context queries, and measure the average response time. The query response time can be broken down into three portions: query mapping, query processing and communication. Query mapping is the time taken by a ContextPeer to map a query to the appropriate semantic cluster(s). Query processing is the time taken

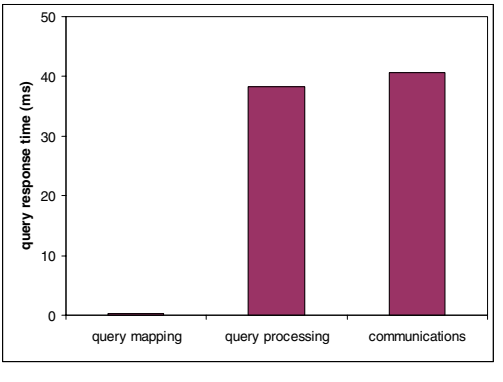


Fig. 5. Response time for context queries

by a ContextPeer to process a query. Query processing involves performing a local lookup against the base model. Communication represents the time taken for queries and their responses to travel over the network. It is the sum of the time taken to send a query from a consumer to a ContextPeer and the time taken to send the query's response from the ContextPeer back to the consumer. The results are shown in Fig. 5. As we can see from the above results, the processing time for query mapping can be ignored; the costs of query processing and communication are the major factors.

4.4 Query Processing Capability

This experiment evaluates the capability of a ContextPeer to process simultaneous queries. In the experiment, a context consumer continuously sends a varying number of queries to the network by randomly picking them from a large query pool. Fig. 6 plots average query processing time against number of simultaneous queries. The graph displays a linear relationship; and shows that the capabilities of a ContextPeer scale well to number of context queries.

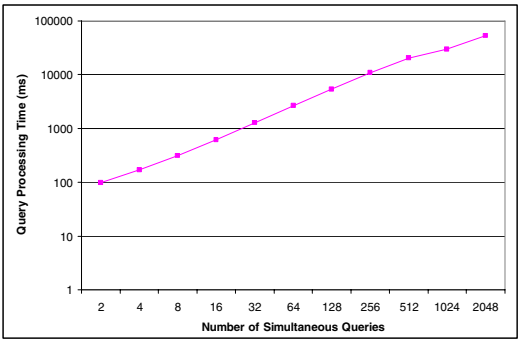


Fig. 6. Query processing capability

5 Related Work

The Context Toolkit [1] provides a software framework and a number of reusable components to support rapid prototyping of sensor-based context-aware applications. However, its context delivery assumes the priori knowledge about the presence of a widget or a context broker. Chen, et al. [2] proposed a platform, named Solar, to support data fusion services and context dissemination to context-aware applications. Solar provides a policy driven data dissemination service based on a multicast tree. However, building a multicast tree for context dissemination may incur large overhead in the presence of node changes. Hong, et al. [3] proposed the Confab infrastructure, which includes a flexible and distributed data store to make it easy to model, store and disseminate context data; and a context specification language for declaratively stating and processing context needs. While our context lookup framework shares the similar idea of distributed context storage of Confab in which the context data is kept close to where it was generated and where it is likely to be used, our emphasis is on how to provide a scalable semantic lookup service using an

overlay network in multiple smart spaces. Gaia [11] is an infrastructure supporting the construction of applications for smart spaces. It consists of a set of core services and a framework for building distributed context-aware applications. Different from the context service in Gaia, we focus on providing a semantic lookup service which context information can be shared in a semantic manner. Knoll, et al. [12] proposed a P2P architecture for context-based system based on Pastry [4]. They modified the Pastry algorithm to optimize the data distribution towards geographic locality. In our framework, data distribution is based on where the context data was generated, and nodes are self-organized into the network according to their semantics.

6 Conclusion

This paper presents the design of a semantic P2P framework for context lookup in multiple smart spaces. The framework offers a de-centralized way for acquiring context data from sensors, storing data and resolving context queries.

References

1. Dey, A.K., Salber, D., Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Anchor article of a spe1cial issue on Context-Aware Computing. Human-Computer Interaction (HCI) Journal 16(2-4), 97–166 (2001)
2. Chen, G.: Solar: Building a Context Fusion Network for Pervasive Computing. Ph.D. Dissertation. Department of Computer Science, Dartmouth College (August 2004)
3. Hong, J.I., Landay, J.A.: An Infrastructure Approach to Context-Aware Computing. Human-Computer Interaction 16 (2001)
4. Rowstron, A., Druschel, P.: Pastry: Scalable, Distributed Object Location for Routing for Large-scale Peer-to-peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, Springer, Heidelberg (2001)
5. Nagel, K., et al.: The Family Intercom: Developing a Context-Aware Audio Communication System. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) Ubicomp 2001: Ubiquitous Computing. LNCS, vol. 2201, pp. 176–183. Springer, Heidelberg (2001)
6. Gu, T., Pung, H.K., Zhang, D.: Information Retrieval in Schema-based P2P Systems using One-dimensional Semantic Space. Elsevier Journal of Computer Networks, Special Issue on Innovations in Web Infrastructure (2007)
7. Smith, M., Welty, C., McGuinness, D.: Web Ontology Lanugauge (OWL) Giude (August 2003)
8. Jena 2 - A Semantic Web Framework, http://www.hpl.hp.com/semweb/jena2.htm
9. Kleinberg, J.: The Small-World Phenomenon: an Algorithm Perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing (2000)
10. RDQL, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
11. Ranganathan, A., Campbell, R.H.: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, Springer, Heidelberg (2003)
12. Knoll, M., Weis, T.: A P2P-Framework for Context-based Information. In: 1st International Workshop on Requirements and Solutions for Pervasive Software Infrastructures at Pervasive 2006, Dublin, Ireland (May 2006)

Usage-Aware Search in Peer-to-Peer Systems

Irene Sygkouna and Miltiades Anagnostou

School of Electrical and Computer Engineering, National Technical University of Athens,
9, Heroon Polytechneiou Str., 15773, Zografou, Athens, Greece
{isygk,miltos}@telecom.ntua.gr

Abstract. We study solutions to a source discovery problem defined in the framework of providing time-critical context-aware services over a Peer-to-Peer communication paradigm. The proposed mechanisms, which take place in ubiquitous computing environments, exploit the locality of reference properties exhibited by context usage patterns, with efficient means provided by Active Networks. Simulation results show that the new methods reduce network traffic while they maintain the good search time of flood broadcasting methods.

Keywords: Ubiquitous computing, Context-aware services, Peer-to-Peer search, Locality of reference properties, Active Networks.

1 Introduction

Ubiquitous computing paradigm offers users the opportunity to truly perform their operations anywhere and anytime. Many practitioners envision a future empowered with context-aware computation and communication infrastructure that allow users to access data and receive service at any place and any time. A context-aware system can be seen as a human assistant given user's context to be responsible to make decisions in a proactive fashion, anticipating user needs while not disturbing the user, except for an emergency [1]. The exploitation of complete context-awareness in state-of-the-art services requires taking advantage of all types of context available to them. Thus, context-aware services (CASs) need a flow of information from and about their environment, in order to be able to adapt to it [2]. Elaborating on the efficient provisioning of CASs, the current work is based on a framework in which a CAS generates context requests and addresses them to the nearest broker. Brokers act as mediatory players between CASs and context sources. The need to support CASs that are highly robust and can scale well with the number of nodes and information sources points to Peer-to-Peer (P2P) architecture [3] as a more promising approach than most centralized solutions offer. Each peer Context Broker (CB) can make information available for distribution and depends on each other for getting information, forwarding requests, etc.

The system faces the challenge to ensure efficient and scalable distribution of context information. Among the most popular approaches applied to improve performance are replication, caching, and intelligent routing. In our framework, the first two are considered less appropriate due to the volatile nature of context information, and thus we are concerned with routing techniques applied every time a

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 565–576, 2007.

request arises and a real-time search has to take place. In its purest form, the P2P model uses message forwarding mechanisms to search for information since there is no centralized server with a global view of all the peers in the network or the information they provide. The respective algorithms are typically implemented in the form of an application-level protocol. However, most existing techniques result in opposite extremes of bandwidth and response time. In this paper, we propose a set of usage-aware mechanisms that exploit the locality of reference properties exhibited by context usage patterns. The context demand profile is monitored by the distributed peer nodes, which exchange their knowledge with efficient means provided by Active Networks. The results show that the proposed mechanisms reduce the network traffic while they maintain the good search time of flood broadcasting methods.

Section 2 presents a literature overview on decentralized search algorithms and section 3 describes the problem we deal with. Section 4 provides first the definition of the locality properties, and then describes the role of Active Networks in searching along with the proposed search mechanisms. Simulation results are presented in section 5 and section 6 concludes the paper.

2 Literature Overview

Resource Discovery constitutes a fundamental problem in large-scale distributed systems, and even though finding resources in a network of computers is a problem probably as old as distributed computing itself, different system requirements and conditions of current large-scale applications have led to a flurry of different approaches to the problem. Thus, the problem of searching for information in P2P networks can be treated in different ways, ranging from centralized indexing schemes, such as Napster [4], to decentralized mechanisms that navigate the underlying network without knowledge of its global structure.

Decentralized search algorithms have been studied both for *unstructured* and *structured* systems. In the former case, there is not any precise control over the network topology or data placement. A client seeking information searches across scattered collections stored at numerous member nodes by forwarding queries to one's neighbors until the target is found. Moreover, both search by identifier and by content is supported. Gnutella [5] uses flood routing to broadcast queries, generating a large amount of unnecessary traffic. There have been continuous efforts to improve the naïve search algorithm based on flooding. The authors of [6] proposed *Random Walk*, which forwards a query to a randomly chosen neighbor at each step and *Expanding Ring*, which performs successive floods with increasing time-to-live (TTL), until the target is found. It was shown through extensive experiments that a 32-walker Random Walk reduces the amount of network traffic by two orders of magnitude at the expense of slight decrease in search speed and generally outperforms Expanding Ring as well. Similar strategies, including Expanding Ring and a variation of Random Walks were examined in [7, 8].

In [9], hybrid search schemes that combine Flooding and Random Walks were proposed, which rectify the performance of Flooding in the case of a sparse network with a few vertices of large degrees. Moreover, it was shown that *Random Walk with Local Flooding*, which performs shallow floodings on each step of the random walk, is more preferable than simple random walk on regular graphs since it achieves

savings in response time, while the savings are much sharper if the graph has supernodes. In [10] a probabilistic message dissemination method was developed. By altering the probability of routing search request messages, it varies the probability that the search is successful. Contrary to the above works that propose alternatives to Flooding based mainly on specific properties of the network topology, in the current paper we expect to reduce network traffic without retarding search, by taking into account the locality of reference properties that are likely to be exhibited by the monitored context usage patterns.

In structured systems, objects are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy. Such systems (e.g. [11, 12, 13, 14, 15]) implement Distributed Hash Tables (DHTs), and search is performed by looking up the DHTs. In more recent works, techniques based on DHTs have been proposed for multiple-keyword search [16] and full-text search [17, 18]. They differ from our approach since we are interested in finding sources that are managed by peers, which have complete autonomy over their location.

3 Problem Description

We consider a set of Context Brokers forming an overlay network. Each Broker manages a set of local information sources registered to it via a registration protocol and thus it maintains the necessary interfaces for interacting with its local sources. Peers can use an enquiry protocol to query other peers in order to discover sources. Once a Broker receives a search request coming from either another peer or a local consumer (e.g. a CAS), it first looks up the request in its local information. If a matching source is not found, the Broker forwards the request to different peers until the target source is located. Precluding dependence from central control, the aim is to design efficient mechanisms for discovering and retrieving data. Due to the volatile nature of context, we deal only with dynamic information sources and thus, there is no caching support for actual data.

Each peer maintains a local directory, the *Local Sources Directory* (LSD), with entries to the sources it manages. Note that sources cannot be replicated. Each peer maintains additionally the *Remote Sources Directory* (RSD), which caches directory entries for sources maintained by other peers. An entry in the RSD is a pair (*source_info*, *loc*), where *source_info* provides a description of the information produced by a source and *loc* is the network address of the peer that is presumed to manage the given source.

Each peer n has a local neighborhood, denoted by $N(n)$ and defined as the set of peers that are close (e.g., at one hop distance or within the same local area network) to that peer. Finally, global network topology is unknown and a peer only contacts peers in its neighborhood, as well as peers indicated in its RSD.

4 Solution

Based on the distributed computing environment provided by Active Networks, we propose specific algorithmic solutions built on top of the P2P communication paradigm, that aim to enhance the system's efficiency and scalability for the provision

of time-critical CASs. The relevant algorithms avoid the inefficiency of flood broadcasting methods, and a source exhibiting a locality of reference property can be easily located by applying an appropriate limited flooding algorithm, according to the type of locality.

4.1 Locality of Reference Properties

We define the following notation and terminology: Given a set of nodes ($n \in N$) and a set of objects ($o \in O$), we denote by $r(o,n)$ a request ($r \in R$), where o ($o \in O$) is the requested object and n ($n \in N$) the node the given request originated from, henceforth called *initiatory-node* of the request. Moreover, we will use the term *home-node* of an object to refer to the node that hosts the source of the given object. Note that in the following text, the words "source" and "object" are used interchangeably.

Temporal Locality. It implies that an object frequently accessed in the past, namely a popular object, is likely to be accessed in the future [19]. We define the popularity f of an object o as:

$$f_o = \sum_{n \in N} \left| R_n^{\,o} \right| . \tag{1}$$

where N is the set of nodes, and $R_n^{\,o}$ the set of requests for object o initiated from node $n \in N$, namely: $R_n^{\,o} = \{ \forall\, r(o',n') \in R \mid \quad o' = o, \quad n' = n \}$.

Observing the context usage patterns in our real test scenarios conducted under the IST project Context [20] we concluded that temporal locality was evident since a few objects were most popular and thus were repeatedly requested from the majority of peers. Such objects usually refer to some elementary types of information that constitute basic components of many complex types and are thus repeatedly requested. For example, a source that provides pure location information of mobile users for a wide geographical area is most likely to be accessed repeatedly, since location information is a basic component of many complex and more specialized types of information.

Geographical Locality. It accounts for the location of the nodes from which a repeated request originates and implies that an object accessed by a client is likely to be accessed again in the future by "nearby" clients [19]. We first define the set $N(n_o, l) \subseteq N$, which consists of the nodes included in a geographical area centered at $n_o \in N$ and extended at distance *l*, as:

$$N(n_o, l) = \{ \forall n \in N \mid \quad dist(n, n_0) < l \} . \tag{2}$$

where *dist(n, n_0)* represents the path length between nodes *n*, n_o. An object *o* exhibits geographical locality with radius *L*, if a significant number of repeated requests

originate from the same geographical area, which is extended around the home-node of *o*, namely:

$$\sum_{n \in N} \frac{\left| R_n^{\,o} \right| \cdot x(n, N(H(o), L))}{f_o} > T_{geo} \ . \tag{3}$$

where f_o represents the popularity of *o*, *H(o)* its home-node, $R_n^{\,o}$ the set of repeated requests initiated from node $n \in N$, T_{geo} a certain threshold and *x(n,N)* equals to 1 (0) if $n \in N$ ($n \notin N$).

Geographical locality is most likely to be exhibited by context usage patterns, since CASs are usually developed and designated to be provided at specific areas, usually near the sources of information they utilize. For instance, in a university campus, a source that provides academic-related context information is meaningful for and thus used from relevant services offered in the campus, which constitutes a case of geographical locality.

Spatial Locality. It is looking for dependencies among the requested objects and implies that objects neighbouring an object frequently accessed in the past are likely to be accessed in the future. Defining a traversal stride to be a sequence of requests where the time between successive requests is less that StrideTimeout seconds [19], an object *o* exhibits spatial locality, if there exists a traversal stride s_o starting with object *o*, such that:

$$\frac{f_{s_o}}{f_o} > T_{spat} \ . \tag{4}$$

where f_o is the popularity of object *o*, f_{s_o} the popularity of the traversal stride s_o and T_{spat} a given threshold.

Spatial locality of reference is likely to be found in a context access pattern due to the modular design based on which context objects are constructed. In particular, a context object may be composed of other simpler objects in a cascaded way, requiring a cascaded assignment of values to each object in turn.

4.2 Active Networks Distributed Computing

Active network developers envisage transforming IP packets into encapsulated fragments of executable code that traverse the network and execute in limited environments at intermediate nodes. P2P networking, on the other hand, was devised as a lightweight, primitive, networking concept that achieves adequate results at minimum cost without sophisticated network protocols. We could thus easily think of active capsules that carry P2P queries to locate particular information, or code for determining traffic and usage patterns for individual sources at each node and dynamically make decisions on redistributing information across the P2P network [3].

Because P2P queries are lightweight, the mobile code would pose a minimal computational burden and impose minimal network overhead, leading to a highly efficient self-sustaining and self-maintaining P2P system. In this framework, *passive monitoring* of context usage patterns is supported, which substantially reduces traffic by piggybacking the relative information on existing active packets traversing the network, as opposed to *active monitoring*, in which the measurements are done by sending additional control messages [21].

4.3 Usage-Aware Search Mechanisms

The proposed mechanisms are based on a limited version of the flooding algorithm that takes advantage of the locality of reference properties exhibited by the monitored context usage patterns. We assume that appropriate Tables maintained by all the peers provide information on the objects exhibiting locality properties: *T-Table* for temporal locality, *G-Table* for geographical locality and *S-Table* for spatial locality. When a peer initiates a request, it first searches the local copies of the Tables to find potential matching sources. In case of a hit, it applies an appropriate limited flooding algorithm, depending on the type of property. Otherwise, it resorts to pure Flooding. In each case, once the requested object is found, the response message follows the reverse path to reach the initiatory-node and a new entry is created in the RSD of the initiatory-node. Note that the Tables are consulted once for each request, namely at the initiatory-node of the request, whereas the RSDs at each visited peer.

The rationale behind the proposed approach is based on two observations: first, only a small percentage of all the available objects are likely to exhibit a locality of reference property. Second, most requests refer to such objects. Therefore, we expect that the majority of requests can be satisfied by looking up the Tables and applying limited flooding, whereas the locality information requires limited storage space and thus can be retrieved quickly with small overhead.

Range-limited Flooding. It proposes to limit the range of flooding to an extent determined by the popularity of the object requested and in a way to save as much bandwidth as possible without increasing the time to locate the appropriate source. Thus, the more popular the object, the lower the number of neighbours K to which the request should be forwarded in each step, since popularity provides a clue about the number of previous repeated requests for the object, which should have thus located the corresponding source. Supposing that peer i receives a request $r(o,n)$, K is given by:

$$K \propto \frac{1}{f_o} \cdot d_i \ . \tag{5}$$

where f_o is the popularity of o and d_i the degree of i.

We assume that the T-Table hosts popularity information of the most popular objects. An entry (o, f_o) indicates the popularity f_o of an object o, as measured by its home-node, which keeps logs of the requests it has serviced. An object o is registered with T-Table only if its popularity is greater than a given threshold a, while an update operation is performed only if a noticeable change, which is determined by a given percentage

threshold b, is recorded by its home-node. In case a peer decides to perform a register or an update operation, the respective information will be encapsulated in a following request message initiated from that peer. In this case, the respective active packet has to load a modified code that executes both the query and the register/update operation in each visited peer, and thus updates all the copies of T-Table.

Depth-limited Flooding. It is proposed as an alternative that aims to save bandwidth by limiting the depth (TTL) of flooding. The success of this algorithm when searching for an object is based on a strong indication that the given object is most likely to be located following a limited number of hops. Therefore, if an object proves to exhibit geographical locality of reference with radius L, it is likely that the same object will be requested from nearby nodes in the future. These nearby nodes could limit the depth of flooding in following repeated requests initiated from them to the value $D=L$, with the expectation that either the source itself will be located nearby or the location of the source will be found cached in the RSD of a nearby node.

The existence of geographical locality of reference for a given object is verified by its home-node. Each peer maintains information related to its broader neighborhood, namely the nodes located in close geographic proximity, and periodically examines the request history of each source it owns, so as to correlate the initiatory nodes of the relevant requests. It thus registers every object that proves to exhibit geographical locality, based on inequality (3), with the G-Table maintained by each node in its broader neighborhood. A register/delete operation is performed by encapsulating the respective information in a following request message initiated from this peer.

Prefetch-limited Flooding. It exploits the idea of prefetching within flooding, namely search for more than one object within a single flooding, in anticipation of future requests. In particular, when a node initiates a request for an object that proves to exhibit spatial locality of reference, it piggybacks its request with a set of queries for the dependent objects that it speculates will be requested by the client in the near future, in order to resolve all of them during the same flooding search.

Contrary to the other properties that are server-detected, a thorough analysis of the clients' access patterns is required in order to identify the spatial locality of reference that may exist among the various objects. Assuming that each peer keeps logs of the requests it has initiated, it is capable of testing for the existence of the property locally for every object that is frequently requested, based on the inequality (4). Note that each entry (o, s_o) of the S-Table indicates the object o that exhibits spatial locality and the respective stride s_o. Every peer that frequently initiates requests for an object registered in the S-Table is responsible for verifying that this object indeed exhibits spatial locality. In the opposite case, it should initiate a delete operation in order to remove the object from all the S-Tables. The register/delete operation is performed with the same piggybacking mechanism.

5 Simulation Results

We have implemented an event-based multithreaded simulation in *Java* to test our algorithms. As a reference for comparison we use the pure Flooding, which achieves

the best search time at the cost of intensive network use, and the Random Walk with Local Flooding of TTL=1, which was proposed recently as an alternative to the pure Random Walk that improves its search time. Henceforth, the latter one will be called Random Walk for the sake of brevity. We assume constant peer participation, no failures, and that the sources do not migrate during the simulation. The simulated requests are simple and there is exactly one matching source to a request. Moreover, we assume that the RSD size is infinite. The initial topology of the overlay network, as formed by the neighbors' connections, is modeled as a random graph that is generated based on the *Waxman model* [22]. We follow the *analytic workload generation* method, which starts with models for various workload characteristics i.e., the locality of reference properties in our case, and then generates outputs that adhere to these models [23]. Since each property is proposed to be exploited in a different way, we generate distinct synthetic traces for each property in order to experiment exclusively with the potential benefits achieved in each case. Zip's law [24] is used to model the distribution of context requests and the default TTL value is set to the graph diameter. As a metric for the time to locate a source we use the *average path-length per request* (defined as the ratio of the total number of hops incurred across all requests to the total number of requests), while the bandwidth consumption is well captured by the *average number of messages per request* (defined as the ratio of the total number of messages sent across all requests to the number of all requests).

5.1 Test Case 1: Temporal Locality

The network size equals to 200 nodes and each peer node controls one source, for a total of 200 sources. Each peer monitors the popularity of the local sources and updates the T-Tables accordingly. For simplicity we assume that the T-Tables are updated at constant periods, namely every time the overlay has executed 100 new requests, for a total of 1000 requests. Fig.1 and Fig.2 depict the evolution of the average path-length and the average number of messages per request, over time, under Range-limited Flooding (R-F), Flooding (F) and Random Walk (RW), for two different values of the Zipf parameter a, namely α=0.7 and α=0.95, respectively. The time evolution is given in terms of consecutive time periods, each corresponding to the execution of 100 requests.

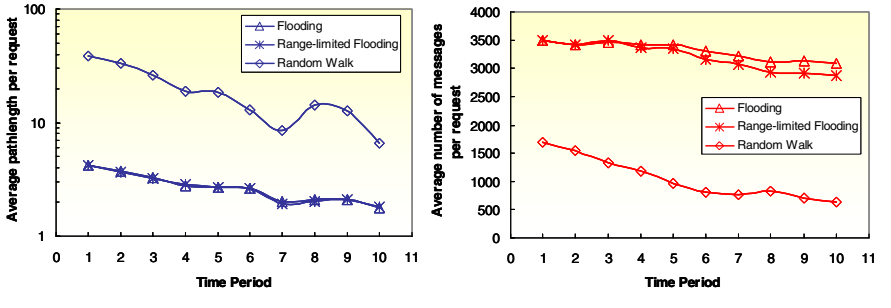


Fig. 1. Performance of R-F versus F and RW, for α=0.7

Clearly, the average path-length per request decreases over time and R-F achieves to follow F very closely. Similarly, the average number of messages per request decreases with time, but in this case R-F clearly outperforms F, achieving lower bandwidth waste. Moreover, the bandwidth savings achieved by R-F becomes more intense with time. Comparing R-F with RW, it becomes obvious that the latter one achieves high bandwidth savings, but at the cost of a noticeable increase in the average path-length per request. It is also remarkable that the performance of RW improves with time. In particular, the bandwidth savings it achieves with regard to R-F increases from 53% to 78%, whereas the path-length savings of R-F with regard to RW decreases from 89% to 73%.

Moreover, the value of a affects the bandwidth savings achieved by R-F with reference to F. In particular, usage patterns with a higher skew in popularity distribution seem to take greater advantage of the savings achieved by R-F (Fig.2). On the other hand, the percentage of bandwidth savings and path-length waste achieved by RW with regard to R-F seem not to be affected from the value of α.

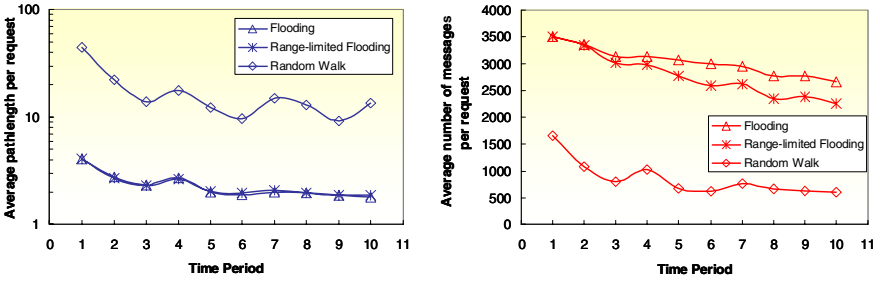


Fig. 2. Performance of R-F versus F and RW, for α=0.95

5.2 Test Case 2: Geographical Locality

We first consider an overlay graph of 500 nodes and density equal to 0.035. Each peer maintains 10 sources. We experiment with synthetic traces of 400 requests produced according to Zipf distribution (with a=0.7), that exhibit geographical locality at different degrees. The degree, namely the fraction of the requests executed in the whole trace that exhibit geographical locality, has been set to 0.2, 0.4, 0.6, and 0.8, with 0.48%, 1.28%, 2.72% and 4.32% of the most popular objects exhibiting the property, respectively. We evaluate the performance of pure Flooding (F), Random Walk (RW) and Depth-limited Flooding (D-F), assuming that the last one has set the radius of geographical locality to 3 and then we repeat the experiment on a graph of density 0.07. The results are depicted in Fig.3 and Fig.4, respectively.

As the degree of geographical locality increases, the average path-length drops and D-F achieves to maintain the path-length at the same level with F. On the other hand, while the average number of messages per request remains almost constant under F as the degree increases, D-F achieves growing savings. Comparing the performance of RW to the one of D-F, we observe that the bandwidth savings achieved by the former with regard to the latter is on the order of 50%, whereas the path-length savings of the

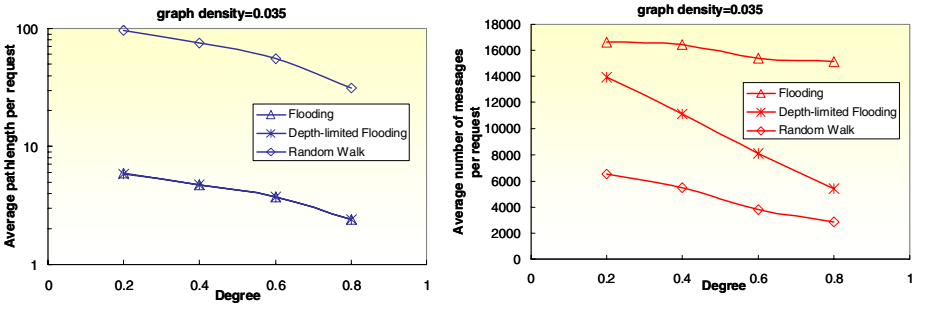


Fig. 3. Performance of D-F versus F and RW for graph density equal to 0.035

latter with regard to the former is 90%. Moreover, both values are not affected by the variation of the degree of geographical locality.

As far as the graph density is concerned, the sparser the graph, the more profitable the application of D-F compared to F would be. On the other hand, the performance of D-F compared to RW seems not to be affected by the graph density.

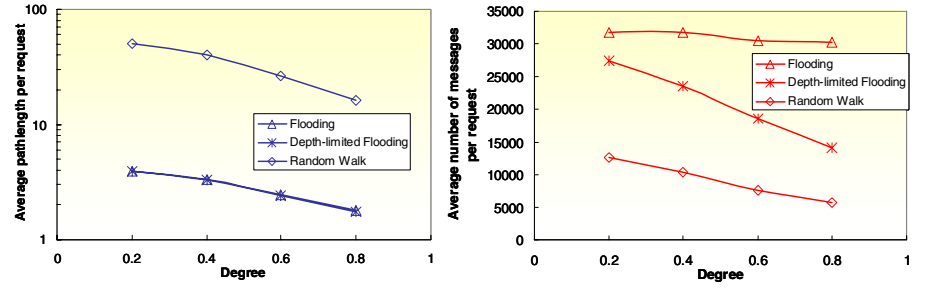


Fig. 4. Performance of D-F versus F and RW for graph density equal to 0.07

5.3 Test Case 3: Spatial Locality

The goal of the third experiment is to quantify the cost of Prefetch-oriented Flooding (P-F) in terms of the degree of spatial locality exhibited by a trace. The network graph consists of 200 nodes, each maintaining 1 source. We generated 4 different synthetic traces according to Zipf distribution (a=0.7), each consisting of 1000 requests. The degree of spatial locality in each of them is adjusted to 0.2, 0.4, 0.6 and 0.8, respectively, with 2.5%, 10%, 26% and 54.5% of the most popular objects exhibiting the property, respectively. The traversal strides consist of 2 objects. Measuring the same metrics, the graphs that result are depicted in Fig.5.

Clearly, while F is not affected notably by the degree variation, P-F achieves significant savings in terms of the average path-length and the number of messages per request, which become more evident as the degree increases.

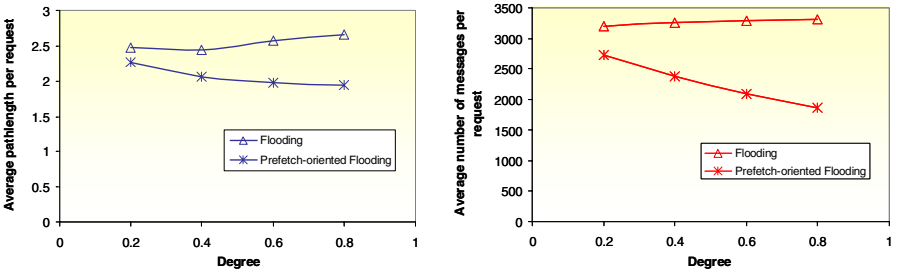


Fig. 5. Performance of P-F versus F

6 Conclusions

While the majority of mechanisms proposed in literature are based on network topology properties, the current work shows the potential of exploiting usage-awareness toward improving search efficiency. The simulation results show that the new mechanisms maintain the good search time of Flooding, while they achieve to reduce the bandwidth consumption that tends to cripple such systems. Moreover, the degree of bandwidth savings is tightly connected to the degree of the locality of reference properties exhibited. In the proposed framework, the overhead imposed by usage-awareness is kept low because the locality-Tables are of small-size, with light registries, and no additional traffic is needed to maintain them. We thus believe that the current work could be considered as a first step toward proposing even more powerful mechanisms that combine both approaches. It therefore contributes to a growing development toward economic activity in decentralized search mechanisms.

References

1. Satyanarayanan, M.: Challenges in Implementing a Context-Aware System. Editorial Introduction in IEEE Pervasive Computing 2 (2002)
2. Xynogalas, S., Chantzara, M., Sygkouna, I., Vrontis, S., Roussaki, I., Anagnostou, M.: Context Management for the Provision of Adaptive Services to Roaming Users. IEEE Wireless Communications 11(2), 40–47 (2004)
3. Parameswaran, M., Susarla, A., Whinston, A.B.: P2P Networking: An Information-Sharing Alternative. Computer 34(7), 31–38 (2001)
4. Napster, http://www.napster.com
5. Kan, G.: Gnutella. In: Oram, A. (eds.) Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly (2001)
6. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and Replication in Unstructured Peer-to-peer Networks. In: International Conference on Supercomputing, pp. 84–95. ACM Press, New York (2002)
7. Yang, B., Garcia-Molina, H.: Efficient Search in Peer-to-peer Networks. In: IEEE ICDCS, IEEE Press, Vienna, Austria (2002)
8. Gkantsidis, C., Mihail, M., Saberi, A.: Random Walks in Peer-to-peer Networks. In: Infocom 2004, pp. 120–130. IEEE Press, Hong Kong, China (2004)

9. Gkantsidis, C., Mihail, M., Saberi, A.: Hybrid Search Schemes for Unstructured Peer-to-Peer Networks. In: Infocom 2005, pp. 1526–1537. IEEE Press, Miami, Florida (2005)
10. Menascé, D.A., Kanchanapalli, L.: Probabilistic Scalable P2P Resource Location Services. ACM Sigmetrics Performance Evaluation Rev. 30(2), 48–58 (2002)
11. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 46–66. Springer, Heidelberg (2001)
12. Stoica, I., Morris, R., Karger, D., Kaashoek, M.E., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: ACM Sigcomm, San Deigo (2001)
13. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content Addressable Network. In: ACM Sigcomm 2001, San Deigo, CA (2001)
14. Rowstron, A., Druschel, P.: Pastry: Scalable, Distributed, Object Location and Routing for Large-scale Peer-to-Peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
15. [15] Zhao, Y., Kubiatowicz, J.D., Joseph, A.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical report, UCB/CSD-01-1141, Berkeley (2000)
16. Reynolds, P., Vahdat, A.: Efficient Peer-to-peer Keyword Searching. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 21–40. Springer, Heidelberg (2003)
17. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-Peer Information Retrieval Using Self-organizing Semantic Overlay Networks. In: ACM SIGCOMM 2003, Germany, pp. 175–186 (2003)
18. Tang, C., Dwarkadas, S.: Hybrid Global-local Indexing for Efficient Peer-to-peer Information Retrieval. In: Symposium on Networked Systems Design and Implementation (NSDI), pp. 211–224, San Francisco, California (2004)
19. Bestavros, A.: Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems. In: International Conference on Data Engineering, New Orleans, Louisiana, pp. 180–187 (1996)
20. IST-2001-38142-CONTEXT http://context.upc.es
21. Caripe, W., Cybenko, G., Moizumi, K., Gray, R.: Network Awareness and Mobile Agent Systems. IEEE Communications Magazine 36(7), 44–49 (1998)
22. Waxman, B.M.: Routing of Multipoint Connections. IEEE Journal on Selected Areas in Communications 6(9), 1617–1622 (1988)
23. Barford, P., Crovella, M.: Generating Representative Web Workloads for Network and Server Performance Evaluation. In: ACM SIGMETRICS 1998, Madison, pp. 151–160 (1998)
24. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web Caching and Zipf-like Distributions: Evidence and Implications. In: INFOCOM,, New York, pp. 126–134 (1999)

A Service Query Dissemination Algorithm for Accommodating Sophisticated QoS Requirements in a Service Discovery System

Liang Zhang and Beihong Jin

Institute of Software, Chinese Academy of Sciences,
Hai Dian, Beijing, PRC
{zhangliang1216,jbh}@otcaix.iscas.ac.cn

Abstract. For many service discovery protocols, user service queries need to be disseminated to some service directories within the system for discovering matched services. The scope of dissemination identifies the scope of service discovery. How to determine the service discovery scope needs to be discussed. In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. The service query dissemination algorithm will dynamically transform the QoS requirements into a set of directories as well as the routing paths to them. The performance analysis shows a sound result of our algorithm.

Keywords: QoS, multicast, service discovery, mobile computing, NP-complete.

1 Introduction

Recent trends in mobile and ubiquitous computing have created new requirements for automatic configuration of network devices. Furthermore, the exploding deployment of network devices in diverse environment has increased the need to simplify the network administration for different kinds of networks. In response to these requirements, a variety of new protocols have been proposed, which attempt to provide automatic discovery and configuration of network devices and services. These protocols are called *service discovery protocol*s [1].

The majority service discovery protocols rely on a special component called *service directory*, or *directory* in short, for maintaining service information. Users post service queries to a directory for services they need. To enhance scalability and robustness, the whole set of service information within a system is normally distributed to a set of directories across the system. In this circumstance, user service queries are first sent to a specific directory called *access directory* and then dynamically forwarded to a set of other directories for service discovery. The challenge is how to select an appropriate subset of directories within the system as well as the routing paths to them. Within the entire spectrum of directory selection, there are two extreme

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 577–586, 2007.

strategies: *full-directory strategy*, where the user desires the service query to be processed at all the directories within the system, and *no-directory strategy*, where the system restricts the service query only at the access directory for minimizing the transmission cost. Both the user requirement of maximizing the processing directories and the system requirement of minimizing the transmission cost are referred to as QoS requirements. Our problem can be defined as dynamically determining the directories and routing paths for disseminating service queries with the consideration of an ordered list of user and system QoS requirements according to the priority. For example, a service query will be disseminated through the minimum spanning tree covering all the directories within the system in order to fulfill both the above mentioned QoS requirements with the user QoS requirement taking the higher priority than the system QoS requirement.

In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. The service query dissemination algorithm will dynamically transform the QoS requirements into a set of directories as well as the routing paths to them. The rest of the paper is organized as follows. Section 2 presents a classification scheme for QoS requirements. Section 3 formally models the problem and proposes a solution to it. Section 4 conducts the performance analysis. The final section concludes the paper.

2 QoS Requirement Classification

Before proposing the general algorithm for accommodating QoS requirements in Section 3, let us first study the characteristics of QoS requirements. All QoS requirements have to be expressed in some measurable QoS metrics. There are two types of QoS metrics: link state metrics and server state metrics, which measure the state information of links and servers, respectively.

The link state QoS requirements can be categorized in three dimensions as shown in Fig. 1. For the first dimension, the link state metrics can be divided into subtypes, within which additive, multiplicative and concave are the most common ones [2]. Let $ls(n_i, n_j)$ denote the link state between the two servers n_i and n_j. For any routing path $rp = (n_1, n_2, \ldots, n_{k-1}, n_k)$, the link state metric ls is

- additive, if $F(ls(rp)) = F(ls(n_1, n_2)) + \ldots + F(ls(n_{k-1}, n_k))$; or
- multiplicative, if $F(ls(rp)) = F(ls(n_1, n_2)) \times \ldots \times F(ls(n_{k-1}, n_k))$; or
- concave, if $F(ls(rp)) = min\{F(ls(n_1, n_2)), \ldots, F(ls(n_{k-1}, n_k))\}$,

where F is a function over link states. For example, transmission delay is an additive QoS metric because the transmission delay of a routing path is the summation of those of the constituent links (Here $F(x) = x$); link-failure rate is a multiplicative QoS metric because the compliment of the link-failure rate of a routing path is the multiplication of the compliment of those of the constituent links (Here $F(x) = 1 - x$); network bandwidth is a concave QoS metric because the network bandwidth of a routing path is the minimum of those of the constituent links (Here $F(x) = x$). Besides, a general link state metric can be expressed by the following equation.

$F(ls(rp)) = Func(F(ls(n_1, n_2)), \ldots, F(ls(n_{k-1}, n_k)))$,
where *Func* represents a special function.

For the second dimension, a classification is made between whether the QoS requirement is to find the best routing path or any routing path bounded by a certain value. For example, "transmission delay is minimum" aims to find the routing path with the minimum transmission delay and we call this type of QoS requirements QoS optimization requirements. Another example, "transmission delay < 30 seconds" aims to find the routing path with the transmission delay smaller than 30 seconds and we call this type of QoS requirements QoS constrained requirements.

For the third dimension, a QoS requirement may only pose constraint to the routing path to each receiver; or it may not concern any individual routing path, but desires the overall routing paths to all the receivers to possess a certain characteristic. For example, "transmission cost < 10" specifies the transmission cost to each receiver should be smaller than 10, while "the summation of transmission cost < 100" guarantees the cumulated transmission cost to all the receivers should be smaller than 100. Another example of group QoS requirement is "the variance of transmission delay < 1 second" meaning that the maximum difference of transmission delay to all the receivers should be smaller than 1 second, which is particular important to applications like VoIP Conference. Of course, it is possible for other general group QoS requirements.

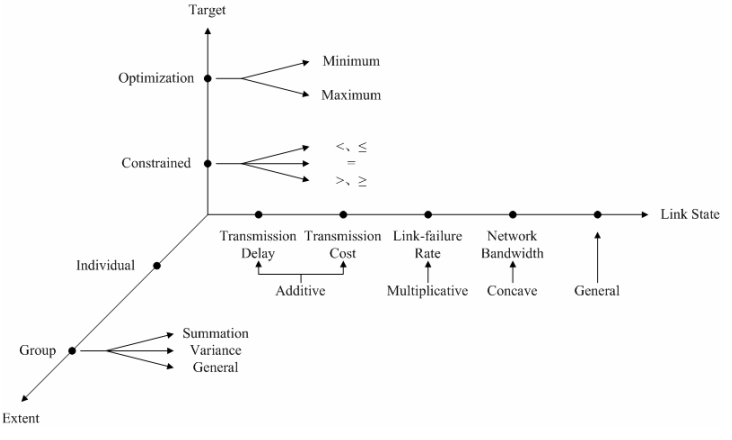


Fig. 1. Link state QoS requirement classification

Similar to the link state QoS requirements, the server state QoS requirements can also be categorized in three dimensions as shown in Fig. 2. The first dimension depicts different server state metrics. The second dimension differentiates optimization and constrained requirements. The third dimension considers either each individual server state or the overall server states of all the servers.

With the above classification scheme defined, any QoS requirement can be modeled in the following format

QoS Requirement = [Extent] + QoS Metric + Target,

where the omission of the optional Extent part represents an individual QoS requirement. With the scheme, we can easily identify different types of QoS requirements with each type representing a specific problem with a certain solution:

- Individual link state QoS requirements are similar to the shortest path tree problem and can be solved by applying the Dijkstra's algorithm either directly or with slight modification.
- Group link state QoS requirements are similar to the minimum spanning tree problem and can be solved by applying the Prim's algorithm either directly or with slight modification.
- Individual server state QoS requirements are simple and can be solved by directly picking the correct servers.
- Group server state QoS requirements are also simple and can be solved by picking the correct sets of servers.

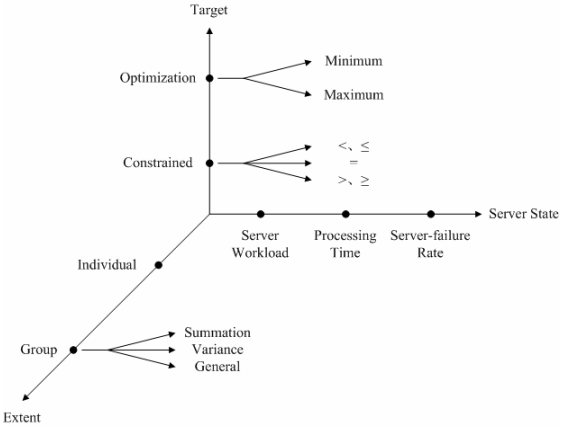


Fig. 2. Server state QoS requirement classification

Although it is easy for a single QoS requirement, it may become extremely challenge to solve even two QoS requirements. [3] points out there are three possible combination of two QoS requirements that will result in NP-complete complexity. Our algorithm aims at providing a general solution for solving a sequence of QoS requirements with polynomial complexity.

3 A General Algorithm

Let us first model our problem formally. Like all the other problems, our problem has input and output. There are five input parameters in our problem (V, v_a, S, L, Q), where V represents all the DSs within the system, v_a stands for the access directory, S keeps the global server state information, L records the global link state information and Q maintains the list of QoS requirements. Notice that we do not introduce an E parameter for all the links within the system as many other papers do. This is because

our system adopts a mesh topology that every DS knows all the other DSs, i.e., E is implicitly defined by V. The output of our problem is (V', P'), which respectively means the set of DSs within the service discovery scope as well as the routing paths to them. With the output, the access directory can easily perform source routing for disseminating the service query.

The input parameter Q needs a bit more explanation. Each QoS requirement in Q is associated with a priority. The higher the priority is, the topper a QoS requirement appears in Q, and the earlier it is processed by the general algorithm. There are three types of QoS requirements with different purposes in Q: *user QoS requirements*, *system QoS requirements* and *default QoS requirements*. The service requestor only issues user QoS requirements to manually control the service discovery scope. However, it is usually unwise to leave clients take full control of the system behavior. Certain administrative control is necessary to protect the system from exhausting the resources. For example, the administrator may restrict all the service discovery within 1 kilometer. We name these QoS requirements system QoS requirements. The user QoS requirements have higher priority than the system QoS requirements. To ensure the uniqueness of the output, it is sometimes necessary to append default QoS requirements at the end of Q. For example, the general algorithm suggests two options of DSs and routing paths after processing the user QoS requirements and the system QoS requirements. Then a default "the summation of transmission cost is minimum" QoS requirement may break the tie and leave a unique output. The default QoS requirements must belong to the optimization type. They are added one by one at runtime in a predefined order until the output is unique. The default QoS requirements have the least priority.

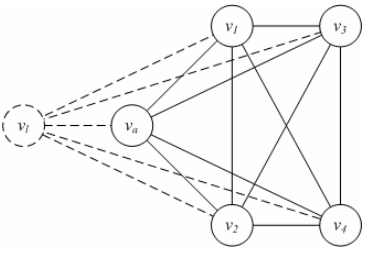


Fig. 3. The logical server

The last thing necessary for mention before presenting the general algorithm is that although the access directory by itself is a DS, we consider there is a separate logical server as shown in Fig. 3 receiving the service query and performing the calculation, and treat the access directory the same as other DSs. That implies the possibility that even the access directory is not included for the dissemination. When the access directory does be included, a logical message is sent to it and executed the same way as a physical message being sent to another DS. We denote the logical server v_l.

Now let us present our general algorithm with the pseudo code below. The main idea is simple. We design a routine procedure for sequentially processing each QoS requirement. During each processing, the algorithm applies the corresponding solution described in the last section either directly or with slight modification for the

specific QoS requirement type. The graph theory and the set theory are heavily depended on in our algorithm.

- At the beginning, there is a mesh graph $\{V, E\}$ where E is implicitly defined by V
- With the logical server v_l included, another mesh graph $G = (\{V, v_l\}, E \cup \{v_l v_i \mid v_i \in V\})$ is defined
- Define $VP = \{V, P\}$ where $P^1 = \{P(v_i) \mid v_i \in V$ and $P(v_i)$ includes all the non-loop routing paths from v_l to v_i in $G\}$
- Define $VP_I = \{VP\}$ and $VP_O = \{\}$
- For the next prioritized QoS requirement q in Q
  - Switch q's major type
    - Case: individual link state optimization
      - For each $VP_i \in VP_I$
        - Define $VP_o.V = VP_i.V$
        - If $VP_i.P = \forall$, then it is a mesh graph
          - Apply the Dijkstra's algorithm over the mesh graph to construct the shortest path tree T
          - $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes the routing path from v_l to v_o in $T\}$
        - Else
          - $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes the optimum routing path in $VP_i.P(v_o)\}$
        - Add VP_o to VP_O
    - Case: individual link state constrained
      - For each $VP_i \in VP_I$
        - Define VP_o
        - If $VP_i.P = \forall$, then it is a mesh graph
          - Apply the Dijkstra's algorithm over the mesh graph to construct the shortest path tree T until further growth of the tree will have the routing path not fulfill the constraint
          - $VP_o.V = T.V$ and **$VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes all the routing paths from v_l to v_o in the mesh graph fulfilling the constraint$\}$**
        - Else
          - $VP_o.V = \{v_o \mid v_o \in VP_i.V$ and $\exists\, p \in VP_i.P(v_o)$, p fulfills the constraint$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes all the routing paths in $VP_i.P(v_o)$ fulfilling the constraint$\}$
        - Add VP_o to VP_O
    - Case: group link state optimization
      - For each $VP_i \in VP_I$
        - Apply the Prim's algorithm over the graph defined by VP_i to construct the minimum spanning tree T
        - $VP_i.P = \{P(v_i) \mid v_i \in VP_o.V$ and $P(v_i)$ includes the routing path from v_l to v_o in $T\}$

[1] The initial P is conceptual as it is impractical, if not impossible, to maintain it in the memory. A symbol $\forall$ is used to denote this concept.

- Define $VP_o = VP_{i\text{-opt}}$ where $VP_{i\text{-opt}} \in VP_I$ and $VP_{i\text{-opt}}.V$ has the optimum overall link states
- Add VP_o to VP_O

- Case: group link state constrained
  - For each $VP_i \in VP_I$
    - **Define VP_{TMP} include all VP_{tmp} with $VP_{tmp}.V \subseteq VP_i.V$ and $VP_{tmp}.P \subseteq VP_i.P$ has the overall link states fulfilling the constraint**
    - Merge VP_{TMP} to VP_O
- Case: individual server state optimization
  - For each $VP_i \in VP_I$
    - Define VP_o
    - If $VP_i.P = \forall$, then it is a mesh graph
      - $VP_o.V = \{v_{o\text{-}opt} \mid v_{o\text{-}opt} \in VP_i.V$ has the optimum server state$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = \{v_l v_o\}\}$
    - Else
      - Define $VP_{tmp}.V = \{v_{tmp} \mid v_{tmp} \in VP_i.V$ and $VP_i.P(v_{tmp})$ contains the routing path $v_l v_{tmp}\}$ and $VP_{tmp}.P = \{P(v_{tmp}) \mid v_{tmp} \in VP_{tmp}.V$ and $P(v_{tmp}) = \{v_l v_{tmp}\}\}$
      - $VP_o.V = \{v_{o\text{-}opt} \mid v_{o\text{-}opt} \in VP_{tmp}.V$ has the optimum server state$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = VP_{tmp}.P(v_o)\}$
    - Add VP_o to VP_O
- Case: individual server state constrained
  - For each $VP_i \in VP_I$
    - Define VP_o
    - If $VP_i.P = \forall$, then it is a mesh graph
      - $VP_o.V = \{v_o \mid$ the server state of $v_o \in VP_i.V$ fulfills the constraint$\}$ and $VP_o.P = \forall$
    - Else
      - $VP_o.V = \{v_o \mid$ the server state of $v_o \in VP_i.V$ fulfills the constraint$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = VP_i.P(v_o)\}$
      - For each $v_o \in VP_o.V$
        - Delete all the routing paths from $VP_o.P(v_o)$ that involve DSs not belonging to $VP_o.V$
        - Delete v_o from $VP_o.V$ if $VP_o.P(v_o) = \varnothing$
      - Loop the previous step until VP_o no longer changes
    - Add VP_o to VP_O
- Case: group server state optimization
  - Define $VP_o = VP_{i\text{-opt}}$ where $VP_{i\text{-opt}} \in VP_I$ and $VP_{i\text{-opt}}.V$ has the optimum overall server states
  - Add VP_o to VP_O
- Case: group server state constrained
  - For each $VP_i \in VP_I$
    - **Define VP_{TMP} include all VP_{tmp} with $VP_{tmp}.V \subseteq VP_i.V$ has the overall server states fulfilling the constraint** and $VP_{tmp}.P = \{P(v_{tmp}) \mid v_{tmp} \in VP_{tmp}.V$ and $P(v_{tmp}) = VP_i.P(v_{tmp})\}$
    - For each $VP_{tmp} \in VP_{TMP}$

- For each $v_{tmp} \in VP_{tmp}$
  - Delete all the routing paths from $VP_{tmp}.P(v_{tmp})$ that involve DSs not belonging to $VP_{tmp}.V$
  - Delete VP_{tmp} from VP_{TMP} if $VP_{tmp}.P(v_{tmp}) = \varnothing$ and break
- Merge VP_{TMP} to VP_O

- $VP_I = VP_O$, $VP_O = \{\}$

- While VP_I includes more than one member
  - For the next predefined default QoS requirement q
    - Switch q's major type
      - Case: individual link state optimization
        - Same as the corresponding block above
      - Case: group link state optimization
        - Same as the corresponding block above
      - Case: individual server state optimization
        - Same as the corresponding block above
      - Case: group server state optimization
        - Same as the corresponding block above
- $V' = VP_I[0].V$ and $P' = VP_I[0].P$

There are three areas in the pseudo code that cannot be solved in polynomial time as highlighted in boldface. We handle them by approximation as follows.

- Case: individual link state constrained Instead of deriving all the routing paths from v_l to v_o fulfilling the constraint, we apply the k-shortest-paths algorithm [4] to only derive k routing paths that best fulfill the constraint.
- Case: group link state constrained Apply the Prim's algorithm to construct the minimum spanning tree until further growth of the tree will not fulfill the constraint. Backward the construction of the tree k steps with each step representing an optional solution.
- Case: group server state constrained Apply the greedy algorithm to include as many servers as possible until further inclusion will not fulfill the constraint. Backward the greedy algorithm k steps with each step representing an optional solution.

As we can see, the main technique we applied for approximation is to introduce a parameter k to limit the search scope. We call this parameter degree of approximation. The greater k is, the more accurate it is, and the less approximation is.

4 Performance Analysis

In this section, we conduct simulation to evaluate the performance of our algorithm. The performance metric is the average processing time for deriving the set of directories as well as the routing paths to them fulfilling the QoS requirements. We compare the performance of our algorithm with that of two source-based QoS-aware multicast protocols: CST [5] and SunQ [6], both of which deal with the combination of delay-constraint and least-cost QoS requirements and belong to the NP-complete problem identified in [3]. We intentionally post the same two QoS-requirements to our algorithm in order to discover how good our algorithm performs in the NP-complete situation comparing with the benchmark protocols.

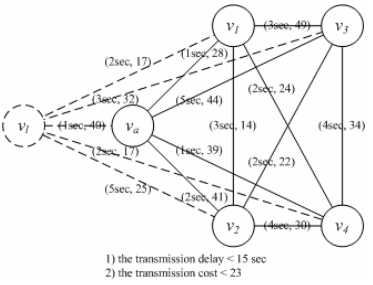


Fig. 4. An example of a network and QoS requirements

The experimental settings are as follows. The program simulates a number of fully-connected directories. For each link between two directories, a pair of integers (X, Y) are generated for identifying the transmission delay and the transmission cost, respectively. The two QoS requirements are 1) the transmission delay < Z seconds and 2) the transmission cost is minimum, where Z is also a generated integer. Fig. 4 shows an example of a generated network as well as an example of generated QoS requirements. In our simulation, the transmission delay of a link (X) is randomly generated between 1 second and 5 seconds, the transmission cost of a link (Y) is randomly generated between 10 and 50, and the transmission cost of a routing path (Z) is randomly generated between 10 seconds and 30 seconds. The test-bed computer is ThinkPad x60 with 1.83 GHz Intel Core Duo T4200 CPU and 1 GB RAM. Each experiment is conducted 1000 times to compute the average.

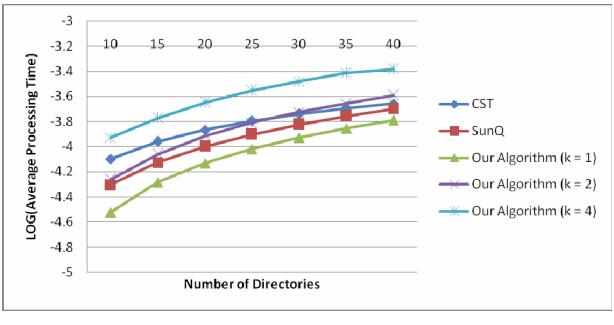


Fig. 5. Performance analysis

Fig. 5 shows the performance results with the number of directories as the x-axis and the logarithm of the average processing time as the y-axis. It is easy to discover that the average processing time of both our algorithm and the two benchmark protocols increase as the number of directories increase. However, this increment is slower than a linear one, which means all of them can effectively reduce the NP-complete problem to polynomial with certain approximation. The second observation is that as the degree of approximation of our algorithm decreases, i.e., k increases, the average processing time increases because more routing paths are derived from those fulfilling the delay-constrained QoS requirement. Although theoretically our algorithm is a

general solution to QoS-aware routing and cannot make the optimization with respect to the dealing QoS requirements, we can always maintain the performance of our algorithm by simply adjusting the parameter k.

5 Conclusion

In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. With the defined classification scheme for QoS requirements, the service query dissemination algorithm performs a routine procedure by sequentially processing each QoS requirement. In this way, our algorithm can deal with whatever combination of QoS requirements and can eventually transform them into a set of directories as well as the routing paths to them. With slight modification, our algorithm can also be applied to other systems supporting QoS-aware routing. Finally, the performance analysis shows a sound result of our algorithm.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 60673123 and the National Hi-Tech Research and Development 863 Program of China under Grant No. 2006AA01Z231.

References

1. McGrath, R.E.: Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing. UIUCDCS-R-99-2132, Department of Computer Science, University of Illinois Urbana-Champaign (March 2000)
2. Chen, S.G.: Routing Support for Providing Guaranteed End-to-end Quality-of-service. Ph.D. Thesis, Department of Computer Science, University of Illinois Urbana-Champaign (May 1999)
3. Chen, S.G., Nahrstedt, K.: An Overview of Quality-of-service Routing for the Next Generation High-speed Networks: Problems and Solutions. IEEE Network 12(6), 64–79 (1998)
4. Macgegor, M.H., Grover, W.D.: Optimized k-shortest-paths Algorithm for Facility Restoration. Software Practice and Experience 24(9), 823–828 (1994)
5. Kompella, V.P., Pasquale, J.C., Polyzos, G.C.: Multicast Routing for Multimedia Communication. IEEE/ACM Transactions on Networking 1(3), 286–292 (1993)
6. Sun, Q., Langendorfer, H.: A New Distributed Routing Algorithm with End-to-end Delay Guarantee. In: Workshop on Protocols for Multimedia Systems (October 1995)

User Preference Based Service Discovery*

Jongwoo Sung, Dongman Lee, and Daeyoung Kim

Information and Communications University, Korea
{jwsung,dlee,kimd}@icu.ac.kr

Abstract. Existing service discovery protocols are designed to take less consideration of different service qualities. Consequently, after the client discovers the services using specific service type information, selecting the best qualifying one among identical services is left to the user or application developers. This paper proposes a quality of service discovery framework which enables a client to discover and select the qualifying service by considering different service qualities based on the weighted preference of the user. We implement and evaluate the prototype system performance and compare it with standard UPnP protocol.

Keywords: Quality of service discovery, user preference, UPnP.

1 Introduction

In a spontaneous network [1] where multiple services and devices cooperate with each other without any user involvement, discovering a qualifying service among different quality services is important. A service discovery protocol allows a service client to discover a qualifying service on a network consisting of heterogeneous devices and services. To discover a service in the network, UPnP (Universal Plug and Play) [2] uses service types and service description while Jini [3] uses service interface and additional static attributes. SLP (Service Location Protocol) [4, 5] adopts string-based service attributes which come with query operators like AND, OR and EQUAL.

However, existing service discovery protocols take less consideration of the quality of a service: if there exist more than one service qualifying to a user's service request, they simply choose one of them based on either the order of appearance or randomly. Consequently, after a client discovers target services using a specific service type or through interface information, selecting the best qualifying one from the discovered services is left to the user or application developers.

Each service instance can be characterized by attributes consisting of a functional service type (cf. printer) and describing characteristics (cf. location) [6]. For service

* This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement)(IITA-2006-C1090-0603-0047) and the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government(MOST) (No. R0A-2007-000-10038-0).

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 587–597, 2007.

discovery of fine granularity, various service attributes as well as service type should be taken into consideration. However, such service discovery is not a simple problem for two main reasons. First, because services are described by the collection of heterogeneous attributes, it is difficult to compare them using simple string operations. Second, since qualified services are scattered across a network(s), comparison among them may be time consuming.

In this paper, we propose a new service discovery scheme which discovers and selects a service that most satisfies the user's quality requirement among qualified service candidates. The key features of the proposed scheme are as follows: 1) it does not depend on any central server for discovery; 2) to compare heterogeneous service qualities efficiently, a scoring algorithm exploiting the user preference table and service descriptions is introduced; and 3) it uses a response collision avoidance mechanism which removes unnecessary responses, keeping service responses from imploding the client.

2 Related Work

Service discovery is used to locate available services by service type; while service selection is concerned with a particular service among services of the same type [7]. In this sense, many established service discovery protocols such as UPnP, JinI and SLP are closer to service discovery than service selection. These discovery protocols find multiple target services of a specified type, while the selection of one from the discovered candidates is left to the user or software developer.

There have been some researches that have enhanced existing service discovery protocols to have better gratuity. The researches in [6] and [7] aim to expand Jini lookup services in order to utilize a variety of service attributes. Researchers in [12] and [13] propose attributes-based extensions for easy service selection based on SLP, which allows the service list to be returned in a sorted order so that the client can easily select the most qualifying service from the candidates.

Quality of service and service discovery are considered together in [8], but that research focuses only on QoS negotiation. The research in [9] integrates network sensitivity into the service discovery process; while [10] describes QoS-aware service discovery for mobile ad-hoc networks. Context or preference service discovery are also proposed in [11, 21, 23].

3 Design Considerations

For designing an efficient service discovery protocol which allows discovery and selection from identical services with different quality of service, the following should be considered:

3.1 Decentralized Architecture

A centralized system fundamentally implies administrative setup and pre-knowledge of discovery environments. Unlike this, the proposed system design does not depend on any central systems. We assume that all services on the network spontaneously

cooperate for providing an answer to a quality of service discovery query. Specifically, each device (service provider) calculates their conformance score based on user defined preferences by themselves.

3.2 Qualifying Service Selection

In order to compare one service quality with others it is required to normalize service characteristics. In addition, an effective way to express user preferences must be created. For this, the service description table and the user preference table are provided for the service provider and the client, respectively. A user can set his preferences in the preference description table, and each preference is represented by a weighted significance value for scoring. Then a service score, which is calculated by the server itself, indicates the degree of service qualification against the user preference.

3.3 Network Implosion Avoidance

Multicast is a popular tool for service discovery when there is no centralized server. Multicast query-multicast response and multicast query-multiple unicast response can overrun the client to the point of implosion. Because this may degrade the overall performance of the system, proper service discovery has to alleviate this problem. We use multicast request-multicast response, and our scoring based jitter delay allows minimum network response messages to transfer.

4 Architecture

The proposed architecture uses an aggressive service discovery approach in which a client sends a service discovery query to service devices instead of waiting for services to announce their existence. The proposed quality of service discovery consists of three tightly coupled phases: 1) sending quality of service discovery queries which describe expected service attributes by a client; 2) service score calculation on each device; and 3) score-based service response. The following sub-sections explain each phase in more detail.

4.1 Quality of Service Discovery Query

To find the most qualifying service, a user needs to describe the expected service attributes which the client is looking for. In the similar way, available services can be described with multiple attributes and their value pairs by servers. User preference based quality of service discovery problem can be conceptually summarized; there are scattered services which have different featured attributes on the networks, and a client describes expected service attributes, which are sent to network and compared to each services. Comparisons of available service's attributes and expected service's attributes are conceptually shown in Fig 1.

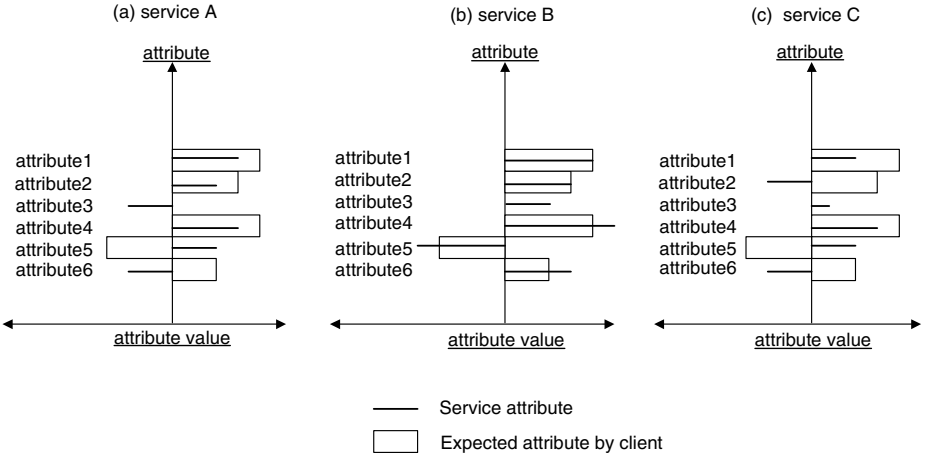


Fig. 1. Conceptual service selection process (Service A. and Service C. have several attributes which do not meet the expectation of client while service B. show best matching to client expectation)

Preference description table = a list of { service attributes, attribute value, significance, arithmetic operators }

Fig. 2. Favorite and preference description table

The *preference description table* in Fig 2 is a list of **attributes**, **its value**, **priroity** and **arithmetic operators** maintained by a client.

A property "**Arithmetic operators"** (>,<.=) explain attribute value condition (for example, delay time is lower than 3 min). Operations may be expanded to include richer and more complex ones as in [19] and [21]. The "**priority**" property expresses the significance of its attribute compared with other attributes. For the illustration purpose five different priority levels (Compulsory, Important, Normal, Trivial, Reject) are defined. Four levels fall into positive categories while the other (reject) level has negative connotations. We explain *preference description table* with a simple printer example in Fig. 3.

In this example, six **attributes**, **value**, **priority** and **arithmetic operators** are defined to express expected user preferences.

After user sets categorized ***priority*** level to each attribute according its importance, it is necessary to change nominal attributes with a numeric value for calculations. Two strategies are used for this quantification. First, we assign high numeric value to more important attributes. Second, we assume that a client who request high priority attributes cannot be satisfied with less priority attributes even though all low priority attributes are qualified. This assumption can be explained with this example. Suppose that there is a service which does not meets "*service type* (e.g., printer)" **attribute** which is ranked as a nominal "*compulsory*" **priority** level. Then, there is the least possibility for a user to choose it regardless of other good attributes.

Attribute name	priority	Arithmetic operator	Value
Service Type	Compulsory	EQ(=)	Printer
Usability (Waiting time)	Important	LT 10(>=)	10 min
Accessibility (location)	Important	EQ (=)	R507
Color support	Trivial	GT 256 (>=)	16 Bit
Pay printer	Reject	EQ(=)	No
Need authorization	Reject	EQ (=)	Yes

Fig. 3. Preference description table

We use the function of S(n) to denote n^{th} (n>=1) priority value with priority ordering from 1(Compulsory) to 5(Reject). Then Fig 4 explains a simple mechanism to calculate n^{th} (n>=1) priority value.

$$S(n) = \sum_{1}^{n} A(n-1) \times S(n-1) + 1, (n >= 1), A(0) = S(0) = 0$$

Fig. 4. Preference quantification

where *A(n)* is the number of **attributes** which have n^{th} **priority** level. The n^{th} **priority** value S(n) can be calculated by adding guard score "1" to sum of all low level attribute's **priority** value between 0 to (n-1). The calculation process starts from low **priority** level (n=1), and it continued to high **priority** (n) based on lower **priority** value.

Negative **priority** level diminishes the chances of a service that meets the negative significance to be chosen. For example, if the printer service has an **attribute** of "pay printer" which is ranked in the "reject" **priority** level, it would be the last candidate regardless of other good **attributes**.

The *preference description table* may be constructed automatically with GUI based program and written in an XML format or simple text. A client sends a service discovery query including this *preference description table* via a multicast channel. Here, scoring calculation may be done by servers to alleviate the processing overhead on limited client devices.

4.2 Service Conformance Score Calculation

Like the preference description table of the service discovery client, participating services maintain their own *service description table* which consists of **service attributes** and their **value** pair.

Once services on multicast channels receive multicast query messages from a client, they find a *preference description table in query messages*. To calculate *conformance score* initially it is set to zero. If an **attribute value**, which describes a service state in the *service description table*, meets an attribute condition in the *preference table*, a corresponding **priority** value is accumulated to the service score. For the simplicities purpose, we only use true or false judgment for each attribute attributes. The *conformance score* is achieved by Fig.5.

Given a priority value S=$\{ s_1 \cdots s_n \}$ where attribute list of preference description table A={ $a_1 \cdots a_n$ }, and a attribute list of service description table B={ $b_1 \cdots b_n$ }, a conformance score C is defined as the sum of priority value where $A_n = B_n$.

$$c = \sum S_n \text{ such that } (A_n = B_n)$$

Fig. 5. conformance score calculation

As a result, a device with a high *conformance score* is considered as a more qualifying service compared to those with low scores.

4.3 Multicast Response and Service Selection

Services return back a discovery response message including a *conformance score* to a client. If multiple response messages from services are sent to a client in a short period of time, they can implode the client [4,15,18,22]. To overcome this response implosion problem, some established service discovery protocols (e.g. UPnP) uses jitter delay, which all service have to wait random period of time before sending their responses.

Our response collision avoidance mechanism is also based on jitter delay, but we assign different jitter delay to services according to their calculated *conformance score*. The jitter calculation algorithm is shown in Fig. 6.

$$PerfectScore = \sum_{n=1}^{m} S(n) \times T(n)$$

$$DelayJitter = (\log_2^{PerfectScore - ConformanceScore})^n \times Scale$$

Fig. 6. Jitter calculations

where m is the number of different **priority** level (we use 4 because there are four priority levels from a trivial to a compulsory), S(n) is n^{th} priority value and T(n) is the n^{th} number of attributes. *PerfectScore* is the calculated score when a service has all **attributes** expected by a client. *DelayJitter* is calculated to give high delay to a low score matching service. Because a linear function gives same interval delay regardless of their *conformance score*, the total delay time may be impractically long. We use logarithms based jitter calculation so that relatively good qualified services can have enough delay difference to be distinguished while less qualified services have small delay difference among different services. As a result, it is assured that multiple qualifying services can have long jitter delay enough to distinguish it with responses from similar services, while less important services may pass over jitter delay quickly to reduce maximum delay time. Variables *n* and *Scale* are used to characterize calculations in real implementations.

The service which has expired jitter delay sends respond messages to a client, and automatically, the first arriving service is the most qualifying service to the client's preference because it can be thought of as having the highest score according to our jitter calculation.

Throughout this process because all service responds to a client, it would result in a waste of precious network bandwidth and a client resource. To avoid this inefficiency we utilize multicast protocol for service discovery responses. This services which receive multicast response message from any other service stop to delay for responding., and simply cancel remaining process.

As a result of our service discovery process, the multicast request and multicast response mechanism alleviates the network implosion problem effectively, and our scored based delay Jitter response assures the right selection of the most qualifying service quickly.

5 Implementation

We implement the proposed system, and compare its performance with the standard UPnP protocol. The testing environment consists of six desktop computers which have Pentium 4 CPU 3GHz and 1GB RAM and 100Mbps Ethernet. We implemented preference description tables and service description tables in client side and server side respectively. We programmed the preference quantification, conformance score calculation and jitter calculation mechanism as explained before.

The UPnP system is implemented based on Intel UPnP [17], and we modify both the UPnP device (services) and the UPnP control point (client). Most modifications are applied into the network layer in SSDP (Simple Service Discovery Protocol) codes of UPnP protocol stacks. For clarity purposes, we disabled periodic service notification announcements and ignoring cache control. Because originally UPnP does not differentiate the quality of services if they are identical type, we modified the UPnP client to accept continuous responses even after it receive first service.

6 Performance Evaluation

In the experimental we made virtual printer example which have 25 attributes. We defined 5 significance levels from 1(compulsory) to 5(reject), and we set the identical number of attributes which have each significance level. We set *n*=1 and *scale*=1 for

jitter calculation. Because performance of proposed service discovery depends on distributions of services, we assume services (service attributes) are uniformly distributed. We use only one service requestor for simple comparison purposes, and increase the number of devices from 0 to 100 with 5 steps.

We evaluate three factors: network traffic, time delay and accuracy. In our evaluation we compare prototype system's performance with standard UPnP. Because UPnP does not support quality of service discovery, we use UPnP's operations (GET/SET) for retrieving service descriptions from services. In UPnP, total network bandwidth is the sum of service discovery request/response message, communication overhead for service descriptions transfer from services. Similarly, service delay is the sum of service discovery request/response delay, communication delay for service descriptions and descriptions sorting delay.

Network traffic is measured by the network capturing tool, Etherreal [16]. The total request and response message sizes are monitored and the result of network bandwidth is shown in Fig. 7. The bandwidth of service discovery increases as the number of services increases.

As a result UPnP bandwidth will increase linearly, but our prototype shows more stable results.

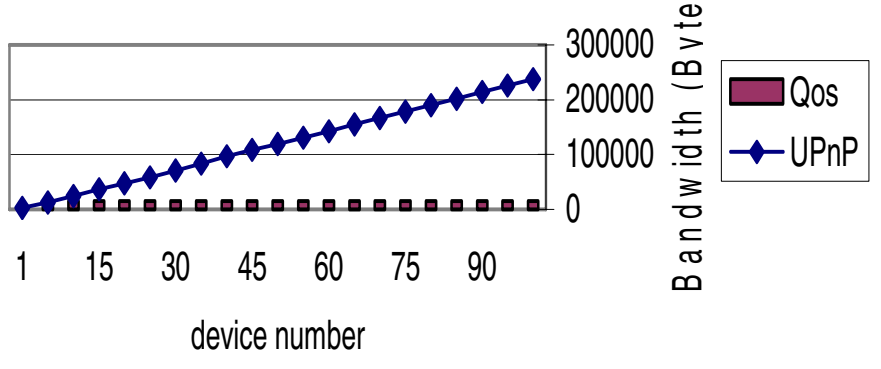


Fig. 7. Network bandwidth

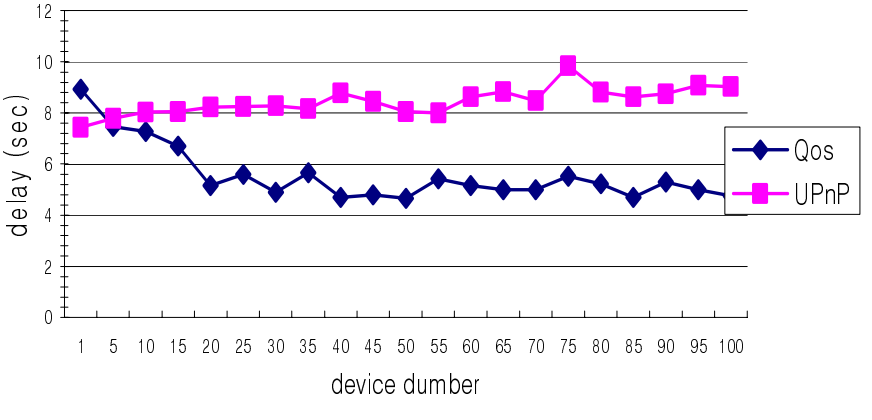


Fig. 8. Service response time delay

Because the proposed service discovery model consists of one multicast request and one multicast response, the network bandwidth is always the same regardless of device numbers.

Service discovery delay is the total amount of time it takes from the client's request to the discovery of one qualifying service. Fig. 8 shows the time delay of the proposed system. UPnP has an MX(default is 3sec) value on its multicast request query header indicating the maximum waiting time. UPnP sends service discovery query two times. On the other hand, the time delay of the proposed system shows fast response time. In addition, because the increased device numbers give high probability of good services overall response time is increased.

Because the client needs to communicate with multiple services at the same time, it is easy to experience network errors. We measure error rates (number of actual messages - expected message number)/ number of expected messages *100. In Fig 9, UPnP have more packet errors when number of device is increased while our system shown little error rate. This may come from packet collisions, and when devices are too crowded, our delay jitter scheme failed to successfully differentiate delay difference among services.

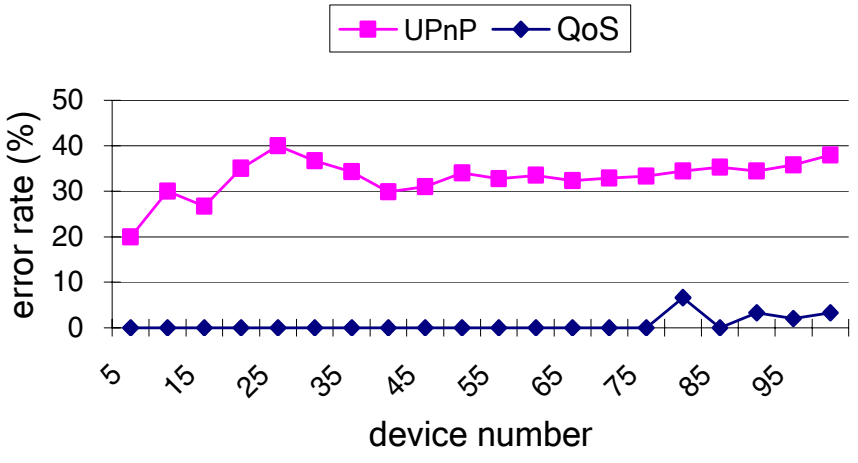


Fig. 9. Service discovery response error rate

In our experimental a client can composite service with maximum 1290 score (when the client sets five attributes for each priorities), and worst case waiting for service discovery response are 10.33 seconds. Because we use weighted delay calculations according to a service's good qualities, the most qualifying service is assured to have at least 1 second delay difference with next qualifying service. This prohibits services are miss-chosen due to simultaneous responses by other good services. However, this long delay difference affects overall response time, we diminish delay gap between less qualifying services. This is because considering a few good services are more important than taking care of many less qualifying services. However, this may cause the least qualifying service to have small delay (0.001 seconds) gap, which can lead response duplications problems.

6 Conclusion

In this paper, we have proposed a new service discovery based on the user's QoS requirement which finds the service based on preferred service quality. To minimize network traffic and client overhead, service description comparison processing is moved from the client side to service devices. The proposed service discovery consists of three tightly coupled phases: 1) preference advertising by the client; 2) candidate service score calculations at each device; and 3) qualifying service response.

To prove the feasibility of our system, we compare the performance of our system against standard UPnP. To the best of our knowledge, we have uniquely categorized some service discovery mechanisms in the point of quality of service discovery. We will study better jitter control to minimize duplicate multicast responses as our future work.

References

[1] Preu, S., Cap, C.H.: Overview of Spontaneous Networking - Evolving Concepts and Technologies. In: Future Services for Networked Devices (FuSeNetD) Workshop, November 8-9, Heidelberg, Germany (1999)
[2] UPnP(Universal Plug and Play) forum, http://upnp.org/
[3] Jini, http://jini.org/
[4] Guttman, E.: Service location protocol: Automatic discovery of IP network services. IEEE Internet Computing 3(4), 71–80 (1999)
[5] Guttman, E., Perkins, C.: RFC2608, Service Location Protocol, Version2 (June 1999)
[6] Moller, M.B, Jorgensen, B.N.: Enhancing Jini's Lookup Service using XML-based Service Templates. IEEE tech-nology of Object-Oriented Language and Systems, 19–31 (2001)
[7] Lee, C., Helal, S.: Context attributes: an approach to enable context-awareness for service discovery. In: IEEE Symposium on Applications and the Internet (January 2003)
[8] Sung, M., Lee, G.-y.: A Qos-enabled Service Discovery and Delivery Scheme for Home Networks. In: IEEE International Conference on Local Computer Networks(LCN 2003) (2003)
[9] Huang, A.-C., Steenkiste, P.: Network-Sensitive Service Discovery. In: USITS (2003)
[10] Liu, J., Issarny, V.: Qos-aware Service Location in Mobile Ad-Hoc Networks. In: IEEE International conference on Mobile Data Management(MDM 2004) (January 2004)
[11] Egashira, R., Enomote, A., Suda, T.: Distributed and Adaptive Discovery Using Preference. In: Proc. of the Workshop on Service Oriented Computing in the IEEE SAINT Symposium (2004)
[12] Zhao, W., Schulzrinne, H., Guttman, E., Bisdikian, C., Jerome, W.: Select and Sort Extensions for the Service Location Protocol (SLP), Internet experimental RFC 3421 (November 2002)
[13] Zhao, W., Schulzrinne, H.: Improving SLP efficiency and extendability by using global attributes and preference filters. In: International Conference on Computer Communications and Networks (ICCCN 2002), Miami, Florida (October 2002)
[14] Goland, Y.Y., et al.: Simple Service Discovery Protocol /1.0, Internet Draft, draft-cai-ssdp-v1-03.txt (Work in Progress)

[15] Mills, K., Dabrowski, C.: Adaptive jitter control for UPnP M-search. In: IEEE International Conference on Communications, May 11-15, 2003, vol. 2, pp. 1008–1013 (2003)
[16] www.ethereal.com/
[17] Intel software for UPnP Technology http://www.intel.com/technology/upnp/
[18] Jaikaeo, C., Srisathapornphat, C., Shen, C.-C.: Diagnosis of Sensor Networks. In: IEEE International Conference on Communications (ICC 2001) (2001)
[19] Howes, T.: A String Representation of LDAP Search Filters, RFC 2254 (December 1997)
[20] Thomson, G., Richmond, M., Terzis, S., Nixon, P.A.: An Approach to Dynamic Context Discovery and Composition. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) UbiComp 2003. LNCS, vol. 2864, Springer, Heidelberg (2003)
[21] Handziski, V., Frank, C., Karl, H.: Service Discovery in Wireless Sensor Networks. Technical Report TKN-04-006, Telecommunication Networks Group, Technische Universitat Berlin (March 2004)
[22] Barbeau, M.: Bandwidth usage analysis of service location protocol. In: Proceedings of the 2000 International Conference on Parallel Processing Workshops (August 2000)
[23] Chen, G., Kotz, D.: Context-Sensitive Resource Discovery. In: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, pp. 243–252 (March 2003)

An Optimal Distribution of Data Reduction in Sensor Networks with Hierarchical Caching

Ying Li[1], M.V. Ramakrishna[1], and Seng W. Loke[2]

[1] Caulfield School of Information Technology, Monash University, Australia
{Ying.Li,Medahalli.Ramakrishna}@infotech.monash.edu.au
[2] Department of Computer Science and Computer Engineering
La Trobe University, Australia
s.loke@latrobe.edu.au

Abstract. Reducing data transmission to conserve energy and provide cost-efficient query answers in sensor networks is the topic of this paper. We consider a sensor network logically organized into a tree structure. The sensors at the lowest level cache raw data, and data is stored with greater degree of compression as we move up the tree. Thus a query answered using data at a lower level has less error but requires more energy. We have proposed a model for accounting the cost of resources consumed and the error in the result obtained. We have formulated an optimization problem for the trade-off between the error cost and the resource cost of answering queries. Its solution enables us to determine the optimal distribution of data reduction at each level. We have presented numerical solutions for some sample data, illustrating the practicality of the scheme.

Keywords: wireless sensor networks, energy-efficient, optimization.

1 Introduction

Conserving energy in sensor nodes is one of the crucial problems in wireless sensor networks. This is specially true for a large sensor network deployed in inaccessible fields. Data transmission is the major energy consumer in wireless sensor networks [1]. To reduce the amount of data transmitted, we investigate a data reduction scheme for efficiently answering queries in a hierarchical data caching structure of large sensor networks while meeting user accuracy requirements. In the caching structure, data at different levels is with a corresponding degree of compression [2]. Sensors at the leaf level cache raw data, compress the data and transmit the compressed data to the intermediate upper level. As we move up the levels, data is compressed to greater extents, but is cached for longer duration. The root node receives user queries, routes the query to an appropriate level and returns the result obtained from lower levels. Caching of the compressed data enables to answer greater historical queries efficiently as some queries can be answered at higher levels instead of being sent to the lower levels. This caching also reduce data transmission in the network since not all the raw

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 598–609, 2007.

data is sent to upper levels. These gains are at the cost of query result accuracy. Queries answered at higher levels will get larger error in the results.

The rest of the paper is organized as follows. The cost model is proposed to describe the cost of query answering in Section 2. Section 3 presents an optimal distribution scheme for data reduction by optimizing the trade-off between energy consumption and user error tolerance. Section 4 reports the experimental results which indicate that our scheme can be adapted depending on the user requests and some network parameters. We present related work in Section 5 and conclude our work in Section 6.

2 Cost Model

Caching data hierarchically with different compression degrees can reduce data transmission. While queries answered with data at higher level get larger error in results. To quantitatively analyze the trade-off between energy consumption and result accuracy, we define two costs, *transmission cost* and *error cost*, to represent the cost for query answering. As energy consumption on computing in a sensor node is orders of magnitude lower than the consumption on transmission, we ignore computation cost in this paper [1].

We define the energy consumption for data transmission as *transmission cost*, denoted by C_T. Let c_t denote the unit cost for data transmission, which can be quantified as cents per bit in practice. Let B_t denote the total amount of data transmitted. Thus, C_T is given by $c_t B_t$. As the transmission for queries and query results is much less than the one for sensor data, and data compression techniques will not be applied on query results, we ignore it in this paper. Besides the transmission cost, we define *error cost* to represent cost generated by errors, which is denoted by C_E. When queries are answered at upper levels where compressed data are cached, the errors are generated in the results. We use relative error as a measure of this error, which is defined as:

$$\frac{|result\ obtained - exact\ result|}{exact\ result}$$

Let c_e denote the unit error cost, which can be quantified as cents per relative error in practice. Let E denote the average relative error on all query answers. Thus C_E is given by $c_e E$. When queries are answered in the caching structure, the transmission cost and error cost will change with varying compression degree of data. We use *cumulative reduction ratio* to represent compression degree on the raw data, denoted by R, which is defined as the ratio between the compressed data size and raw data size. *Reduction ratio* is used to represent the data compression degree between two adjacent levels in the hierarchical structure and the definition is given in Section 3. When the cumulative reduction ratio increases, more data are transmitted to upper levels, the transmission cost increases. On the other hand, the increase in cumulative reduction ratio leads to less errors in query results, thus the error cost decreases. The relationship between the the costs and cumulative reduction ratios are illustrated in Fig. 1. In this paper, we aim to find a data reduction scheme with minimum total cost for query

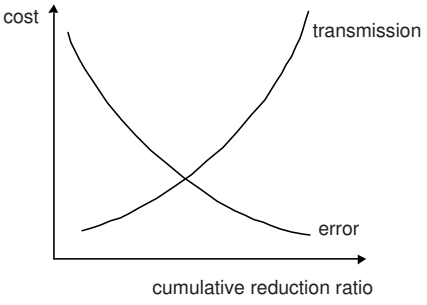


Fig. 1. Relationship between Transmission Cost and Error Cost

answering. Let C denote the total cost, i.e. the sum of transmission cost and error cost, which is given by:

$$C = C_T + C_E = c_t B_t + c_e E.$$

3 Optimal Distribution of Data Reduction

Given a group of queries to be answered in the hierarchical caching structure, we concern the optimal compression degree for data at each level to keep the overall cost minimum. For instance, there is a three-level caching structure. The lowest level is level 0, and the root node is at level 2. The number of queries to be answered at level 0, 1, 2 are 500, 300, 100 respectively. Thus the total cost of answering this group of queries is $C = c_t(B_1+B_2)+c_e(500e_0+300e_1+100e_2)/900$, where B_1 and B_2 are the amount of data transmitted from level 0 to 1 and from level 1 to 2 respectively, e_0, e_1, e_2 are the average relative error in query results at level 0 to 2 respectively. If the number of queries to be answered at level 0, 1, 2 changes to be 100, 400, 400 respectively, in order to obtain minimum overall cost, it is expected that e_1 and e_2 decrease to make the error cost low. The decrease in e_1 and e_2 need the increase in B_1 and B_2, thus the transmission cost increases. Hence, we want to find the best average relative error and the amount of data to be transmitted at each level to get minimum overall cost of query answering. This is essentially the problem of finding the data compression degree at each level. To solve this problem we formulate an optimization problem in this section to give a general statement.

We first consider the data transmission cost. Suppose there are l levels in the hierarchical structure, numbered as 0 to $l-1$ from bottom to upper levels. At level 0, each sensor node produces raw data at the rate of b_0. At a certain upper level i, a sensor receives compressed data from its children at the rate of b_i. Each sensor at level i has n_{i-1} children. We then define *reduction ratio* at level i as the ratio between the data received in a node at level i and the one received in all its children at level $i-1$. Let r_i denoted the reduction ratio, which is given by:

$$r_i = \frac{b_i}{n_{i-1} b_{i-1}}.$$

Defining *reduction ratio* enables us to determine the compression degrees among the levels. In fact, the cumulative reduction ratio R can be calculated by the reduction ratios. Let R_i denote the compression ratio of the data at level i, there is $R_i = \prod_{j=1}^{i} r_j$. Let N_0 be the total number of sensors at level 0. Then the amount of data received at level i, denoted by B_i, is given by:

$B_i = N_0 b_0 R_i = N_0 b_0 \prod_{j=1}^{i} r_j \quad i \in [1, l-1]$

Thus the total amount of data transmitted is:

$B_t = \sum_{i=1}^{l-1} B_i = N_0 b_0 \sum_{i=1}^{l-1} \prod_{j=1}^{i} r_j \quad i \in [1, l-1]$

The total transmission cost is:

$$C_T = c_t B_t = c_t N_0 b_0 \sum_{i=1}^{l-1} \prod_{j=1}^{i} r_j \tag{1}$$

We next consider the error cost of query answering. Let e_i denote the average relative error of a query result when the query is answered at level i. Let q_i denote the number of queries answered at level i. Given a group of queries, the average relative error E of all the queries is:

$E = \frac{e_0 q_0 + e_1 q_1 + \ldots + e_{l-1} q_{l-1}}{q_0 + q_1 + \ldots + q_{l-1}} = \frac{\sum_{i=0}^{l-1} e_i q_i}{\sum_{i=0}^{l-1} q_i}$

In this equation, $\sum_{i=0}^{l-1} q_i$ is the total number of queries posed at the hierarchical caching structure. For simplicity, let q denote the total number of queries, i.e. $q = \sum_{i=0}^{l-1} q_i$. e_0 is 0 as the raw data is cached at level 0. The error e_i is due to the compressed data and is a function of cumulative reduction ratio. In general, e_i can be represented as:

$$e_i = g(R_i) \tag{2}$$

With different compression techniques applied on the sensor data, this function will be different. Further description on this function is given later in this section. Hence, the total error cost is given by:

$$C_E = c_e \frac{\sum_{i=1}^{l-1} e_i q_i}{\sum_{i=0}^{l-1} q_i} = c_e \frac{\sum_{i=1}^{l-1} q_i g(\prod_{j=1}^{i} r_j)}{\sum_{i=0}^{l-1} q_i} \tag{3}$$

Given the costs of transmission and error above, we want to find the optimal reduction ratio at each level to keep the overall cost minimum. Thus, this problem is formulated as an optimization problem:

Minimize

$$C = c_t N_0 b_0 \sum_{i=1}^{l-1} \prod_{j=1}^{i} r_j + c_e \frac{\sum_{i=1}^{l-1} (q_i g(\prod_{j=1}^{i} r_j))}{\sum_{i=0}^{l-1} q_i} \quad 1 \le i \le l-1 \tag{4a}$$

This problem is under following constraints:

– The errors in query results meet user specified accuracy requirement, i.e.

$$0 < g(\prod_{j=1}^{i} r_j) < T_{ei} \tag{4b}$$

where T_{ei} is the threshold of error tolerance at level i, which is decided by the user accuracy requirement. The discussion on user accuracy requirement and T_{ei} is given later in this section.

– reduction ratio at each level is between 0 and 1.

$$0 < r_i < 1 \tag{4c}$$

– As one of the characters of the hierarchical caching structure, older historical data is cached at higher levels than the data at lower levels. Let t_i be the time span for data cached at level i, there is:

$$0 < t_{i-1} < t_i \tag{4d}$$

Let M_i be the memory space for caching in a sensor node at level i, the time span for data cached at this node is given by:

$t_i = \frac{M_i}{b_i} = \frac{M_i}{b_0 \prod_{k=0}^{i-1} n_k \prod_{j=1}^{i} r_j}$.

Description of e_i and T_{ei}

As mentioned in Equation(2), the average error in query results at level i is the function of ratio R_i. This function is determined by the compression technique used. In our previous work, we proposed data approximation algorithms for sensor data [3]. Using some real life data sets, we also analyzed the relationship between the data cumulative reduction ratio and the error generated from the approximated data. Based on our experiment results, we set the error function as following to further illustrate the optimization problem in the next section.

$$e_i = g(R_i) = \frac{k(1 - R_i)}{R_i} \qquad (k > 0) \tag{5}$$

where k is the coefficient of the function. With different compression algorithms or on different data sets, the value of k will be different. We set k as 2 in this paper. In this function, if R_i is 0, then $e = \infty$. This means that when data is totally compressed to size 0, the queries are not able to be answered. If $R_i = 1$, $e = 0$. This indicates that when the raw data is sent to the upper levels, there will be no error for query answering. If $R_i > 1$, the value of e is less than zero which is meaningless. Thus, it complies with our requirement that the value of R_i should be between 0 and 1. The graph of this function is illustrated in Fig. 2.

We use the idea presented by Ganesan *et al* to define the relationship between user accuracy requirement and the error with the compressed data [4]. Users generally expect less errors for queries referring to recent data. If queries refer to the older data, larger error is tolerable. With data cached at the hierarchical caching structure, the errors are represented by a step function. Fig. 3 illustrates the relationship between the user requirements and the errors, where $z(t)$ represents user requirements, $f(t)$ represents the errors obtained. The values of $f(t)$ should be less than $z(t)$ to meet user requirements. For instance, within $(t_1, t_2]$, $f(t_1)$ should be less than $z(t_1)$ to keep $f(t)$ over $(t_1, t_2]$ satisfy the user requirements. Thus $z(t_1)$ is threshold of error tolerance for level 2, which is defined as T_{e2}. To

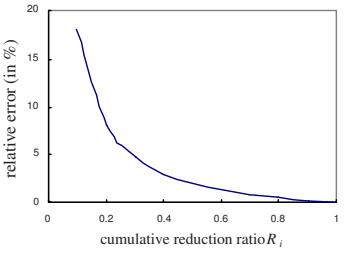


Fig. 2. Error Function $g(R_i)$

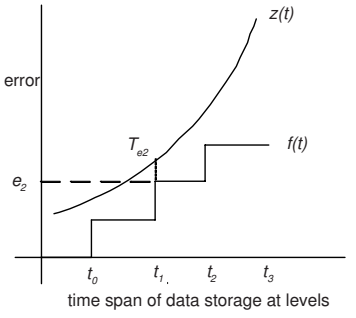


Fig. 3. User Error Requirement $z(t)$ and Result Error $f(t)$

answer queries with data at level 2, there is $e_2 < T_{e2} = z(t_1)$. In general, the error threshold is given by:

$T_{ei} = z(t_{i-1})$.

4 Solution of the Optimization Problem

It appears that the optimization problem in Equation(4) is too difficult to solve in general. However, we can solve it numerically for a given set of parameters. In the following, we use two sets of parameters to illustrate the capability and usefulness of the proposed optimization technique.

4.1 Sample Problem with Two Levels

We consider a sensor network with two-level caching. At level 0, there are N_0 sensors and each sensor produces data at the rate of b_0. The raw data is compressed and sent to level 1 at reduction ratio r_1. For this case, the problem described by Equation(4) can be simplified as following:

Minimize

$$C = c_t b_0 N_0 r_1 + k c_e \frac{q_1}{q_0+q_1} \frac{1-r_1}{r_1} \tag{6a}$$

Subject to:

$$0 < k\frac{1-r_1}{r_1} < z(t_0); \tag{6b}$$

$$0 < r_1 < 1; \tag{6c}$$

$$0 < t_0 < t_1. \tag{6d}$$

For simplicity, we assume the user specified accuracy requirement is a linear function, i.e. $z(t) = t$. The time span for data at level 0 is t_0, which equals M_0/b_0. Thus, the constraint(6b) can be rewritten as:

$0 < k\frac{1-r_1}{r_1} < \frac{M_0}{b_0}$

To obtain the minimum value of C, let $C^{'} = 0$. Then we can get:

$$r_1 = \sqrt{kc_e \frac{q_1}{q_0+q_1}(c_t b_0 N_0)^{-1}}$$

To illustrate the nature of the solution, we use sample values in the equations above. Some of them are based on the typical values presented in the literature [5,6].

$c_t = 10^{-6}¢/b$ $c_e = 1¢/\%$ $b_0 = 10Kb/s$
$N_0 = 500$ $M_0 = 512KB$ $q_0 = 10,\ q_1 = 20,\ q = 30$

The optimal reduction ratio for this case is $r_1^* = 0.516$.

Fig. 4 indicates the effect of ratio r_1 on the transmission and error cost. We observe that, in accordance with Equation(1) and (3) respectively, the transmission cost increases linearly and the error cost reduces dramatically with r_1. Furthermore, the optimal reduction ratio will be different with varying network parameters, such as c_t and c_e. In Fig. 4(a), the transmission cost changes slower than the error cost over the reduction ratio 0.1 to 0.9. Increased the unit transmission cost c_t to $3 \times 10^{-6}¢/b$, we get another curve of transmission cost illustrated in Fig. 4(b), where the transmission cost increases rapidly. The optimal reduction ratio then decreases, which indicates that the data is sent to level 1 at higher compression degree when the unit transmission cost increases.

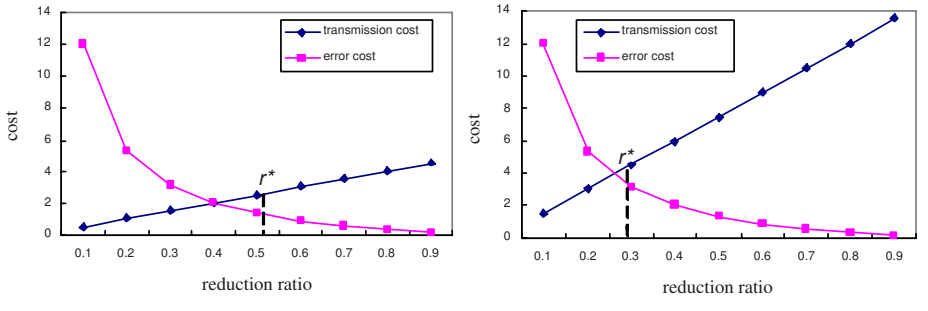


Fig. 4. Cost Trade-off for the 2-level Caching Structure ($b_0 = 10Kb/s, c_e = 1¢/\%, N_0 = 500, k = 2, q_0 = 10, q_1 = 20$)

Except the network parameters, the time span of data requested by queries also affects the optimal reduction ratio r_1^*. This is illustrated in Fig. 5. Given a group of queries, when there are more queries referring historical data at level 1, i.e. q_1/q increases, the optimal reduction ratio increases. This enables more data send to level 1, so that those queries get less errors in results. We also plot another curve by changing the value of the error function coefficient k. Since the increase in k causes the rapid increase in errors, r_1^* will increase to allow more data send to level 1. This is verified by the experiment results reported in Fig. 5, where r_1^* increases rapidly with $k = 3$ comparing the one with $k = 2$.

2-level case is a special example as it only has one reduction ratio parameter r_1. As it can clearly show the effect on costs from the change of reduction ratio,

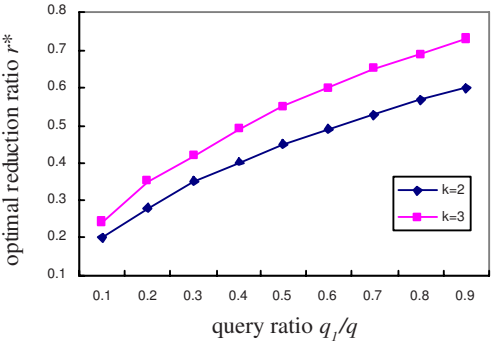


Fig. 5. Reduction Ratios as A Function of q_1/q ($c_t = 3 \times 10^{-6}$¢$/b, b_0 = 10Kb/s, c_e = 1$¢$/\%, N_0 = 500$)

we discuss it in a separate section. In the following, an example of multi-level is used to analyze the effect on reduction ratios from the change of data requests and network parameters.

4.2 Sample Problem with Four Levels

We use 4-level caching structure in a sensor network as an example to illustrate the multi-level problem. The transmission cost in Equation(1) can be simplified as:

$$C_T = c_t b_0 N_0 \sum_{i=1}^{4-1} \prod_{j=1}^{i} r_j = c_t b_0 N_0 (r_1 + r_1 r_2 + r_1 r_2 r_3)$$

Same error function e_i and user requirement function $z(t)$ as the ones in the 2-level case are used here. Then the error cost is:

$$C_E = c_e \sum_{i=1}^{4-1} (q_i g(\prod_{j=1}^{i} r_j)) / \sum_{i=0}^{4-1} q_i = \frac{k c_e}{q} (q_1 \frac{1-r_1}{r_1} + q_2 \frac{1-r_1 r_2}{r_1 r_2} + q_3 \frac{1-r_3}{r_3})$$

The threshold of error tolerances on level 1 to level 3 is $T_{ei} = z(t_{i-1}) = t_{i-1}, i \in [1, 3]$. Suppose all the nodes have same number of children n. Then Equation(4) is simplified as:

Minimize:

$$C = c_t b_0 N_0 (r_1 + r_1 r_2 + r_1 r_2 r_3) + \frac{k c_e}{q} (q_1 \frac{1-r_1}{r_1} + q_2 \frac{1-r_1 r_2}{r_1 r_2} + q_3 \frac{1-r_1 r_2 r_3}{r_1 r_2 r_3})$$

Subject to:

$0 < k\frac{1-r_1}{r_1} < \frac{M_0}{b_0}$ $\quad 0 < k\frac{1-r_1 r_2}{r_2} < \frac{M_1}{n b_0}$ $\quad 0 < k\frac{1-r_1 r_2 r_3}{r_3} < \frac{M_2}{n^2 b_0}$

$0 < r_1 < 1$ $\quad 0 < r_2 < 1$ $\quad 0 < r_3 < 1$

$0 < t_0 < t_1 < t_2 < t_3$

Suppose the relevant parameters are set as:

$c_t = 1 \times 10^{-6}$¢$/b$ $\quad c_e = 1$¢$/1\%$ $\quad b_0 = 10Kb/s$

$n = 8$ $\quad N_0 = 512$ $\quad k = 2$

$M_0 = M_1 = M_2 = 512KB$ $\quad q_0 = 10, q_1 = 500, q_2 = 100, q_3 = 80$

This nonlinear constraint problem can be solved using Matlab tool and the solution is:

$r_1^* = 0.50, r_2^* = 0.58, r_3^* = 0.89$

Next, we consider the effect of the user queries and network parameters on the reduction ratios. The results are reported in Fig. 6. Fig. 6(a) illustrates the effect on the reduction ratios when the percentage of q_1 increases. r_1^* increases along with the increase in q_1/q. This enables more data sent to level 1 so that less error generated in answering queries at this level. r_2^* and r_3^* decrease because of the decrease in the percentage of q_2 and q_3. Fig. 6(b) reports the effect of the increase in percentage of q_2 on reduction ratios. r_2^* increases along with the increase in q_2/q. This enables more data sent to level 2 so that less error generated at this level. When q_2/q is less than 50%, its rise leads to the slight decrease in r_1^* because of the decrease in $q1/q$. When q_2/q is over 50%, its effect on r_1^* turns to be determinant, thus r_1^* starts to rise. Because of the decrease in $q3/q$ and the increase in r_2^*, r_3^* rapidly drops. Fig. 6(c) shows the effect of the increase in $q3/q$. The effect on r_1^* is similar with the one in Fig. 6(b). When q_3/q is less than 50%, r_2^* shows a gradual increase as the consequence of the interaction between the slight decrease in r_1^* and rapid increase in r_3^*. When q_3/q is over 50%, its effect turns to be determinant. Thus r_2^* approximately reaches 1 to allow more data to be received at level 2 and to be sent to level 3. In a summary, Fig. 6 indicates that the change in the percentage of queries at certain level has direct effect on the reduction ratio at the corresponding level and also strong effect on the immediate upper level. Furthermore, the results show that more queries are answered at a certain level, more data is sent and less errors will be generated at this level.

Unit transmission cost c_t and unit error cost c_e also have effect on the reduction ratios. Fig. 7(a) illustrates the effect of c_t on r_i^*, where $log(c_t)$ is used as x-axis value for better display. When c_t increases from $10^{-6}¢/b$, r_1^* firstly decreases while r_2^* and r_3^* keep constant. When c_t increases up to $3.16 * 10^{-5}¢/b$ (i.e. $log(c_t) = -4.5$), r_1^* stops decreasing and r_2^* starts to decrease. The similar process is taken place on r_2^* and r_3^*. When c_t reaches $3.16 * 10^{-3}¢/b$ (i.e. $log(c_t) = -2.5$), r_2^* stops decreasing and r_3^* starts to rapid decrease. The decrease process among r_i^* indicates that the increase in c_t gradually affects the reduction ratios from lower level. This is due to the total amount of data transmitted at lower levels is larger than the one at higher levels. Fig. 7(b) illustrates the effect of c_e on r_i^*. The reduction ratios in the hierarchical structure shows the same change order as the ones in Fig. 7(a), but with different change direction. The increase process among r_i^* indicates that the c_e also gradually affects the reduction ratios from lower levels. The first increase of r_1^* enables more data to be sent from lower level, and alleviates total error cost.

In summary, these results show the effect of reduction ratio on the cost of query answering, and the effect of user requests and network parameters on the distribution of data reduction. We observe that the scheme is adaptive to keep the overall cost minimum. Further, in this scheme, the increase in the percentage

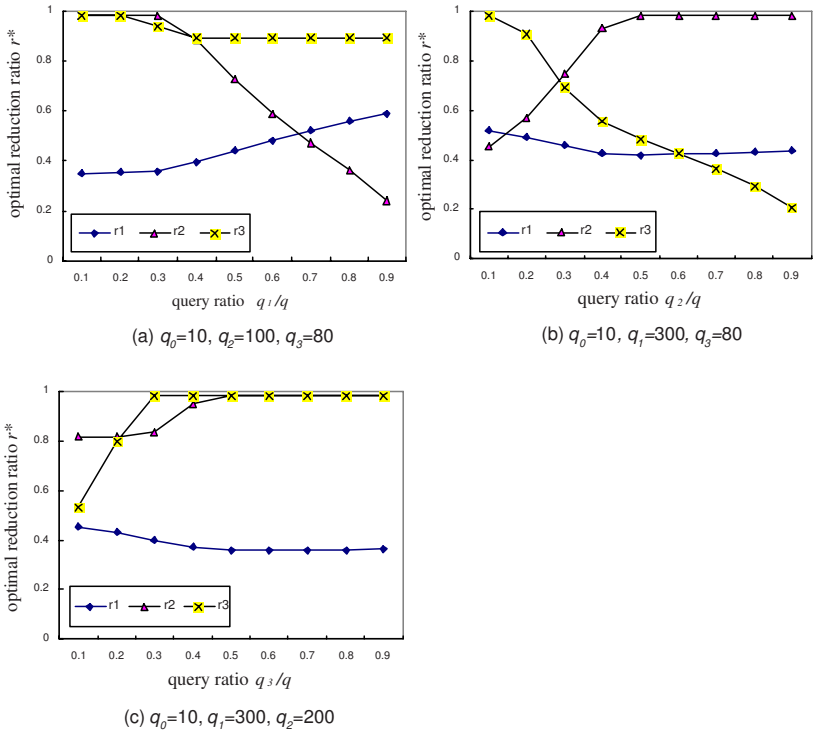


Fig. 6. Reduction Ratios as A Function of $q_1/q, q_2/q, q_3/q$ ($c_t = 10^{-6}¢/b, b_0 = 10Kb/s, c_e = 1¢/1\%, n = 8, N_0 = 512, k = 2, M_0 = M_1 = M_2 = 512KB$)

of queries posed at a certain level leads to the increase in the reduction ratio at this level, i.e., less compression is applied on the data sent to this level.

5 Related Work

A wide range of methods have been proposed to reduce data transmission to conserve energy in wireless sensor networks. These can be roughly divided into three categories: data routing, data compression, data prediction. Some researchers investigate to route the sensor data efficiently. Meliou *et al* present an algorithm to compute the optimal communication path for data transmission [7]. An analytical model and a heuristic algorithm are proposed in [8] to construct an energy efficient routing for real time data aggregation gathering. Exploiting the correlation character in sensor data, some researchers have devoted to sensor data compression or approximation to reduce data transmission. Exploiting spatial correlation in sensor data, a distributed wavelet compression algorithm is proposed in [9]. Lazaridis *et al* represent sensor data in an approximate format by dividing the time series into segments [10]. Some researchers investigate to reduce data transmission by predicting data in sink node. For example, an ARIMA

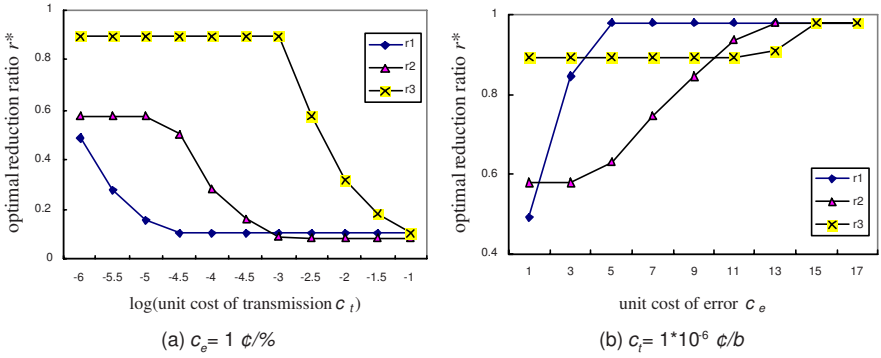


Fig. 7. Effect of Unit Cost on Reduction Ratio ($b_0 = 10Kb/s, N_0 = 512, k = 2, M_0 = M_1 = M_2 = 512KB, q_0 = 10, q_1 = 300, q_2 = 100, q_3 = 80$)

model is used to predict data in sink node in [11]. Although these work consider the trade-off between energy consumption and data quality, they neither pay attention on the cost of query answering nor put the error cost in overall cost for optimizing their solutions.

Our work is similar to that of Ganesan *et al*, where the compressed data is stored in a hierarchical structure [4]. Their work focuses on optimizing the memory usage, i.e. how long the data can be cached at each level, to satisfy user accuracy requirements given the limited total memory capacity. Our work focuses on optimizing the data reduction ratios while keep the overall cost to a minimum.

6 Conclusion

Given a group of queries to be answered with hierarchically cached sensor data, different data reduction schemes lead to different costs of data transmission and error. In this paper, we provided a technique for determining the optimal strategy for data compression to minimize the energy consumed while meeting the user requirement. We used example data drawn from the literature to illustrate the practicality of the technique presented. The results show that the optimal data reduction scheme can adaptively change according to network parameters and user requirements.

References

1. Pottie, G.J., Kaiser, W.J.: Wireless integrated network sensors. Communications of the ACM 43(5), 51–58 (2000)
2. Li, Y., Ramakrishna, M., Loke, S.W.: Approximate query answering in sensor networks with hierarchically distributed caching. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications, Vienna, Austria, pp. 281–285. IEEE Computer Society Press, Los Alamitos (2006)

3. Li, Y., Loke, S.W., Ramakrishna, M.: Energy-saving data approximation for data and queries in sensor networks. In: Wen, G., Komaki, S., Fan, P., Landrac, G. (eds.) 2006 6th International Conference on ITS Telecommunications Proceeding, Chengdu, China, pp. 782–785. IEEE Computer Society Press, Los Alamitos (2006)
4. Ganesan, D., Greenstein, B., Estrin, D., Heidemann, J., Govindan, R.: Multiresolution storage and search in sensor networks. ACM Transactions on Storage 1(3), 277 (2005)
5. MICAmote: http://www.xbow.com
6. Kumar, R., Tsiatsis, V., Srivastava, M.B.: Computation hierarchy for in-network processing. In: Proceedings of the 2nd ACM international conference on Wireless Sensor Networks and Applications, pp. 68–77. ACM Press, New York (2003)
7. Meliou, A., Chu, D., Guestrin, C., Hellerstein, J., Hong, W.: Data gathering tours in sensor networks. In: The Fifth International Conference on Information Processing in Sensor Networks, pp. 43–50. IEEE Press, Los Alamitos (2006)
8. Hu, Y., Yu, N., Jia, X.: Energy efficient real time data aggregation in wireless sensor networks. In: International Wireless Communications and Mobile Computing Conference, Vancouver, Canada, pp. 803–808 (2006)
9. Ciancio, A., Ortega, A.: A distributed wavelet compression algorithm for wireless sensor networks using lifting. In: Preceeding of International Conference on on Acoustics, Speech and Signal Processing, pp. 825–828. IEEE Press, Los Alamitos (2004)
10. Lazaridis, I., Mehrotra, S.: Capturing sensor-generated time series with quality guarantees. In: Proceedings of the International Conference on Data Engineering, pp. 429–440. IEEE Press, Los Alamitos (2003)
11. Liu, C., Wu, K., Tsao, M.: Energy efficient information collection with the arima model in wireless sensor networks. In: Proceedings of IEEE Global Telecommunications Conference (2005)

MOFBAN: A Lightweight Modular Framework for Body Area Networks

Benoît Latré, Eli De Poorter, Ingrid Moerman, and Piet Demeester

Ghent University - IBBT vzw- IMEC vzw
Department of Information Technology (INTEC)
Gaston Crommenlaan 8, bus 201,B-9050 Gent, Belgium
Tel.: +32 9 331 49 00, Fax: +32 9 331 48 99
benoit.latre@intec.UGent.be

Abstract. The increasing use of wireless networks, the constant miniaturization of electrical devices and the growing interest for remote health monitoring has led to the development of wireless on-body networks or WBANs. The research on communication in this type of network is still at it's infancy. The first communication protocols are being proposed, but a general architecture that can be used to integrate the protocols easily is still lacking. However, such an architecture could trigger the development of new protocols and ease the use of WBANs. In this paper, we present a lightweight modular framework for body area networks (MOFBAN). A modular structure is used which allows for a higher flexibility and improved energy efficiency. The paper first investigates the challenges and requirements needed for sending messages in a WBAN. Further, we discuss how this framework can be used when designing new protocols by defining the different components of the framework.

1 Introduction

Recent advancements in electronics have enabled the development of small and intelligent (bio) medical sensors which can be worn on or implanted in the human body. However, the use of wires to extract data is becoming too cumbersome due to the multitude of sensors. As a solution, the sensors placed on and inside the body are equipped with a wireless interface which enables an easier application [1]. Doing so, a patient is no longer compelled to stay in a hospital. The health care is becoming mobile. This is referred to as m-health [2]. For this purpose, a new type of network is defined: a wireless on-body network or a Wireless Body Area Network (WBAN) [3,4]. This type of network communicates wirelessly and consists of several small and mobile devices close to, attached to or implanted into the human body. Interaction with the user or other persons is usually handled by a personal device, e.g. a PDA or a smartphone which acts as a sink. Generally speaking, one can distinguish two types of devices: sensors and actuators. The sensors are used to measure certain parameters of the human body, either externally or internally. Examples include measuring the heartbeat, body temperature or recording a prolonged ECG. The actuators or actors on the

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 610–622, 2007.

other hand take some specific actions according to the data they receive from the sensors or through interaction with the user, e.g. an actuator equipped with a built-in reservoir and pump for administering the correct dose of insulin to diabetics based on the measurements of glucose level. These systems reduce the enormous costs of patients in hospitals as monitoring can occur real-time and over a longer period of time, even at home [5].

Only emerging in recent years, the concept of a Wireless Body Area Network can be seen as fairly new. The main research nowadays focuses on the development of new radio interfaces and sensors. The devices are becoming tinier, even less than 1 cm^3 [6] and the energy consumption is decreasing [7]. The IEEE 802.15 Working Group has recently started a Study Group for WBANs [8]. Current implementations of WBANs generally assume a star topology where the sensors or actuators are directly communicating with the personal device. However, recent studies have spoken out for the use of multi hop routing in wireless on-body networks where intermediate sensors may be used as relay devices in order to reach the personal device [9,10]. The main reasons for using such an approach are to increase the reliability and connectivity of the network, the high path loss experienced around the body [11,12] and to even further lower the energy consumption.

The development of new network protocols especially designed for body area networks has been considered in a lesser degree. Nevertheless, the creation of protocols which are adapted and optimized to the specific requirements of WBANs, can even further lower the energy consumption while other requirements, e.g. reliability and delay, are still satisfied. In order to boost the development of such protocols, we propose a lightweight framework for a system architecture usable in body area networks called MOFBAN or MOdular Framework for Body Area Networks. This framework will allow for an easy integration of existing and newly developed protocols and optimizes the energy-efficiency by using its modular structure. It further provides a uniform interface for application and radio designers.

The remainder of this paper is as follows. Section 2 gives an overview of current work in the field of Body Area Networks where we will focus on existing networking protocols for body area networks. The specific characteristics and properties will be discussed in Section 3. Section 4 discusses the modular framework of the proposed protocol. The modules are described in Section 5. In Section 6, the communication in the framework is discussed. And finally, Section 7 offers directions for future research and Section 8 concludes the paper.

2 Related Work

Protocols for WBAN can be divided in *intra-body communication* and *extra-body communication*, see Figure 1. The former controls the information handling between the sensors or actuators and the personal device [13], the latter ensures communication between the personal device and an external network [7,14,15,16]. Doing so, the medical data of the patient at home can be consulted by a doctor.

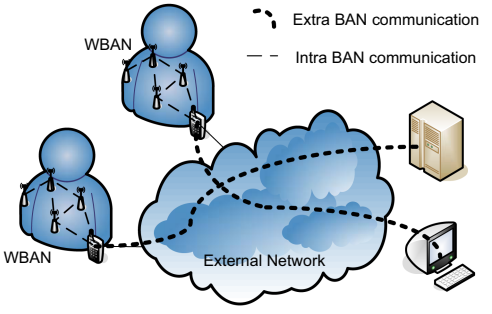


Fig. 1. Example of intra-body and extra-body communication in a WBAN

This paper deals with intra-body communication. Developing efficient routing protocols in WBANs is a non trivial task because of the specific characteristics of a wireless environment. First of all, the available bandwidth is limited, shared and can vary due to fading, noise and interference, so the protocol's amount of control information should be limited. Secondly, the nodes that form the network can be very heterogeneous in terms of available energy or computing power. A lot of research has already been performed in the area of sensor networks [17]. These protocols can not be used in a intra-body network as is as these protocols are mainly developed for large networks whereas the WBAN only has a limited number of devices. Besides, sensor networks assume one-way-communication between the sensor and the sink and more reliability is wanted in WBANs. In current implementations, these networks consider a star topology where the sensors are directly (and of course wirelessly) connected to a personal device [18]. Hence, the routing aspects of these protocols are very minimal as only direct communication is to be contemplated.

The strict separation of the protocol stack has proved to be a good solution for wired networks, but is not suitable for wireless networks [19]. Hence, a cross-layer approach is considered as the way to go in developing protocols for wireless networks and sensor networks. By using a cross layer architecture, optimization can be done at several layers at once, it is possible to achieve a global optimization and conflicting optimizations between different layers can avoided. Hence, most recent protocols have opted for a more holistic view using a cross layer approach [20]. A special type of cross layer is the modular approach [21]. This allows for a rich interaction between the building blocks of the protocol, but requires a total new approach which changes the very way protocols are organized. Another approach of a modular sensor network, is to make the sensor nodes themselves modular in hardware. This is done in mPlatform [22], that uses a collection of stackable hardware modules that share a well defined common interface.

3 Properties of a WBAN

Many different types of sensors and actuators are used in a BAN. The main use of all these devices is to be found in the area of health applications. In this view

Table 1. Examples of medical WBAN applications [23,24]

Application	Data Rate	Bandwidth	Accuracy	Reliability
ECG (12 leads)	144 kbps	100-1000 Hz	12 bits	10^{-10}
EMG	320 kbps	0-10,000 Hz	16 bits	10^{-10}
EEG (12 leads)	43.2 kbps	0-150 Hz	12 bits	10^{-10}
Blood saturation	16 bps	0-1 Hz	8 bits	10^{-10}
Glucose mon.	1600 bps	0-50 Hz	16 bits	10^{-10}
Temperature	120 bps	0-1 Hz	8 bits	10^{-10}
Motion sensor	35 kbps	0-500 Hz	12 bits	10^{-3}
Audio	1 Mbps	–	–	10^{-5}
Voice	50-100 kbps	–	–	10^{-3}

a BAN can be utilized to provide interfaces for the disabled, for diagnostics, for drug administration in hospitals, for telemonitoring of human physiological data, as aid for rehabilitation, etc. The devices are worn on the body and therefore should be made as tiny as possible. Consequently, limited space will be available for energy supply. In order to minimize the energy consumption, one can use an extremely low transmit power. Energy scavenging can be used, but this will only deliver small amounts of energy [25].

An important property is that the data consist of medical information. Hence, a high reliability and low delay is required. It is crucial that messages with monitoring information are received by the health care professionals. The reliability can be considered either end to end or on a per link base. Examples of reliability include the guaranteed delivery of data, in-order-delivery, ... Besides, the delivery of the messages should be in reasonable time.

Different types of delivery can be distinguished: *continuous* (data or control information is sent continuously or periodically with small intervals), *demand driven* (data is only sent when needed), *event driven* (data is sent whenever an event occurs, i.e. if a threshold is crossed) or *hybrid* (a combination of the types above). Most applications will have continuous data transmission.

A small overview of health care applications in a BAN can be found in Table 1. The data rate is calculated by means of the sampling rate, the range and the desired accuracy of the measurements [23,24]. The sampling rate is twice the required bandwidth. The number of bits required per measurement is calculated through the range and accuracy:

$$\text{Data rate} \;=\; \text{nr of bits} \cdot 2 \cdot (f_{max} - f_{min}) \tag{1}$$

Looking at Table 1 and the visualization of Fig. 2, we can group the applications into 4 categories:

- Low data rate and low reliability
- Low data rate and high reliability

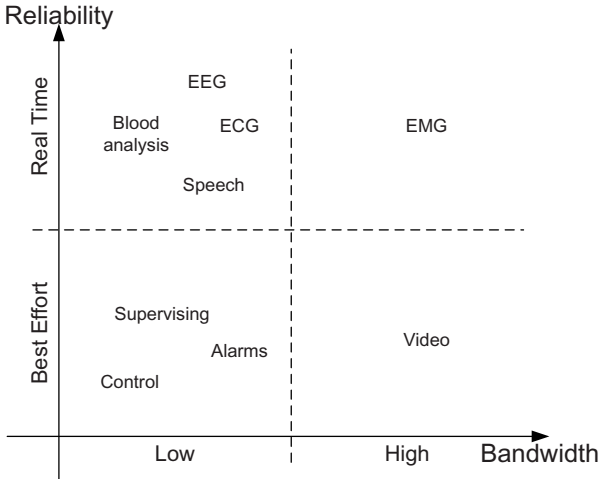


Fig. 2. Traffic analysis in a WBAN

– High data rate and low reliability
– High data rate and high reliability

It is clear that the traffic in a WBAN is quite heterogeneous and newly developed protocols should be capable of coping with this heterogeneity in mind.

In most cases, a WBAN will be set up in a hospital by medical staff, not by network engineers. Consequently, the network should be capable of configuring itself automatically, i.e. self-organizing should be supported. Whenever a node is put on the body and turned on, it joins the network and the routes are set up without any external intervention. When a route fails, a back up path should be set up. The self organizing aspect also includes the problem of addressing the nodes. They can use a preconfigured address given at manufacturing time (e.g. the MAC-address) or an address given at set up time by the network itself. Further, the network should be quickly reconfigurable, i.e. for adding new services.

Another important issue is the network security. The data gathered by the sensors is highly confidential and private. Special efforts such as encryption, key establishment, authentication, ... are needed.

From the analysis above, it is clear that we have a set of different requirements for supporting communication in a BAN. In order to ease the development of new and robust applications, a framework and new protocols are needed. These should consider the energy efficiency, reliability, the delay, the different QoS-levels and data rates, the ease of use of the system and the security. In the following, we present a framework especially designed for body area networks that will take care of these requirements.

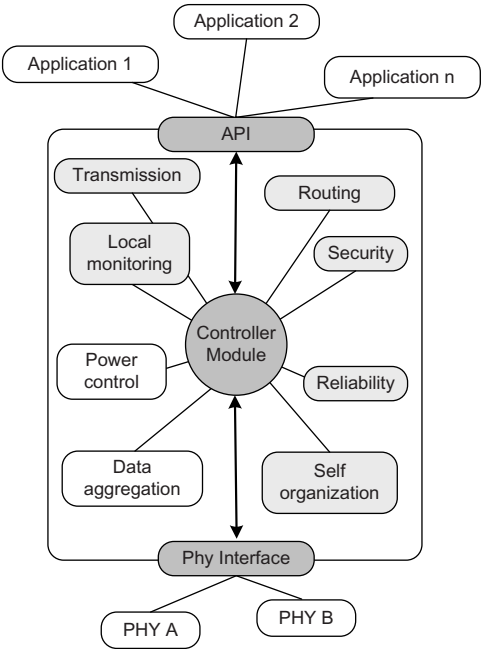


Fig. 3. The modular framework for BAN or MOFBAN

4 Modular Framework

In this section, we present MOFBAN, a lightweight MOdular Framework for Body Area Networks. The framework uses a modular design instead of the normal layered approach. This means that all the functionalities needed are implemented as software modules [21]. The use of modules allows for a more flexible solution as some functionalities can be changed, added or removed more easily, simply by altering the corresponding module. These modules can be used and altered more easily compared to a general cross layer approach where all the functionalities are implemented in one layer.

Developing protocols in a modular way requires a new approach in designing network protocols. In a layered design, the interfaces between the protocols residing on different layers are well-defined, i.e. in the TCP/IP stack. This is no longer the case in a modular framework. One has to make sure the modules are called in the appropriate order and that the exchange of parameters between the modules is standardized. In MOFBAN, the interaction between the different modules is handled by a controller module that is responsible for addressing the appropriate function at the right time. The controller module further acts as a general access to a storage space or data bank for the parameters that can be used in the network.

An example of the MOFBAN framework can be found in Figure 3. The most important element of the framework is the middle part. It contains the different

modules which implement the desired functionalities and holds the controller module. Further, two interfaces are provided. At the top, the application interface eases the use of the framework for application designers. The application layer is handled separately in order to facilitate the use of different applications (such as sensing, video,...) and the addition of extra services/applications. It is necessary to define a sort of generic interface between applications and the framework (see section 6). At the bottom, the physical interface provides a uniform access to the physical layer. The framework has a simple architecture and the use of modules avoids the duplication of functionality, making the framework lightweight.

In a Body Area Network, two types of devices can be defined: a sensor/ actuator node and the personal device. In these two types, the MOFBAN framework is implemented.

- The *Sensor/Actuator nodes* are the regular nodes in the network. The specific use of the node is determined by the application designer who uses the application interface to activate the required functions. A node can be a ECG-sensor or an actuator. Another example is that the node can be used as a relay device. These devices are merely used for relaying data, not for sensing. The question of which modules needs to be used is determined by the application designer. The sensor nodes can have different implementations between themselves. Doing so, the network can support the different requirement of heterogeneity of the network.
- The *Personal device* acts as a gateway between the BAN and other networks. Therefore, the personal device needs to support the normal IP-stack. It is an IP-capable device and it takes care of the conversion between the modular protocol stack and the layered OSI protocol stack. This device generally has two or more physical interfaces: one to connect with the WBAN and the other to connect with an external network. It further has a larger energy supply and more computation power.

5 Modules

The modules of MOFBAN take care of the networking functionality in general, such as routing, medium access, reliability, self organization,... Three types of modules can be distinguished: the *controller* module and the *required* and *optional* modules. The required modules are essential to have a proper working BAN and implement the basic networking functionalities. When a more functional BAN is needed, i.e. a BAN supporting a certain level of reliability, security or power control, extra optional modules are added. It is up to the user of the network, i.e. the application designer, to define what is needed.

5.1 Controller Module

The controller module is the most important module. It is responsible for handling the correct functioning of the framework and handles all incoming requests

from the physical and application layer. It consists of two major parts: a scheduler that calls the other modules at the appropriate time and a passage to the database that acts as a data-repository accessible by the other modules.

It is of utmost importance to keep the information between the different modules consistent. This is regulated by the controller module. It defines the interaction or communication process between the modules. For this purpose the controller module uses an additional database module. If a module wants to access or pass information, this is communicated to the controller that gets or stores the information in the database. The controller module is responsible for solving and controlling interdependencies between different modules. A uniform data structure is used by all the modules.

The controller is responsible for calling the appropriate module. This can occur packet-based or periodically. When a packet is received by the framework, the controller intercepts it. The data part is removed and temporally stored in the database. Based on the header information, the correct module is activated by the scheduler. When the module has finished, it passes the control back to the scheduler which activates the next module. The controller also holds several timers. Upon expiration of a timer, the appropriate module is activated.

5.2 Required Modules

The required modules are essential to have a proper working WBAN. In the following, an overview of the required modules is given:

- The *Information Module* stores all the information from the network and the modules. It is the database of the controller. The controller module acts as a gateway to the database and controls the access.
- The *Transmission Module* regulates the transmitting of data on the medium and handles the channel access to the medium. The implementation in this required module is very basic: a simple CSMA with collision avoidance.
- The *Routing Module* is responsible for setting up a path toward the personal device or other destinations. This can be done using a weight function, number of hops, ... The routing module provides the next hop to the transmission module via the controller module. It can use information about the network, e.g. from the QoS module or from the information module. Different implementations can be made for this module. If a new protocol is used, a new routing module can be added easily. Even multiple routing modules can exist in the same node. It is the responsibility of the controller to activate the correct routing module.
- The *Local Monitoring Module* monitors the network parameters such as the link quality, received signal strength, remaining battery power of the node, the number of neighbors, ... Stated otherwise, this module retrieves the information of the physical layer and other layers that needs to be shared among the other modules. The information is stored in the information module.
- The *Self Organization Module* is responsible for the automatic set up of the network and for maintaining the network. It is activated by the controller module when the node starts up and at certain periods of time.

5.3 Optional Modules

The second type of modules are the optional ones. These modules are used to enhance the functionality of the network. Some examples to illustrate the realm of possibilities:

- The *Advanced Transmission Module* introduces a more sophisticated channel access where slots are used.
- The *ACK-Module* is used for sending acknowledgments. It stores a copy of the packet sent and a timer is started in the controller. If no ACK is received when the timer expire, the transmission module is activated by the scheduler. The module is also activated when a packet is received and it is required to send an ACK.
- The *Security and Privacy Module* has as primary goal to authenticate devices in order to protect the Body Area Network from intruders. The second goal is to protect the privacy of sensitive information, i.e. medical data, by encrypting the data and/or encrypting the links between the authenticated devices. In order to obtain this, a private key can be exchanged.
- The *Power Control Module* acts on the transmission power of a node. This can be useful for limiting the number of neighbors and thus influencing the interference between nodes, or for lowering the power consumption and thus lengthen the lifetime of the node. This module not only considers the power control of each node individually, but can also look at the whole (or a large area) of the network. The module needs to work closely with the routing module. The routing module can ask to this module to alter the transmission power (and doing so the network topology) if no appropriate route to the destination can be found.
- The *Reliability Module* can be regarded as an extended version of the transmission module. It adds error control, better retransmission, priority queuing, ... Stated otherwise, this module is responsible for providing QoS and guaranteed delivery. If wanted, one can define multiple modules that each take care of a QoS aspect.
- The *Data Aggregation Module* can be considered as an extension of the routing protocol. The data received by other nodes is aggregated prior to send them further to the personal device. Doing so, fewer transmissions are needed and the aggregated data can consist of fewer bits. Thus, the energy efficiency is increased.
- The *Local Data Processing* module adds the possibility to perform local data processing or in network processing.

6 Communication

When developing a framework, it is important to define how the communication in the framework should take place. This communication can be either within or between the framework and the application or physical layer or between nodes

themselves. In this section, we will briefly describe how the communication is handled in MOFBAN.

The modules in the framework need to communicate with each other. This communication mostly is nothing more than passing parameters to a module or invoking another one. In the framework, this is handled by the controller. The module sends the parameters to the controller which stores it in the information module. By using the controller as gatekeeper to the database, data conflicts between modules can be avoided.

The interaction between the framework and the application interface is provided via a sort of API, usable by the application designers. The API eases the application development as the designers do not need to be aware of the underlying network characteristics. It allows the designers to adjust settings such as the required level of security, the maximum desired delay, the bitrate needed by the application,... The application interface also allows the application designer to select the modules needed in order to meet the characteristics of the network/application. The API informs the controller module which modules needs to be included in the framework.

The properties of the physical layer largely depend on the design of the hardware. In order to have a common interface usable by the framework, a mapping between the proprietary characteristics of the hardware and the more generic properties used by the framework is provided. Examples of such information are bit error rate (BER), path loss, received signal strength, ... Every time a packet has been received, additional transmission information can be communicated to the framework through the interface. The controller module receives this information and stores it in the information module.

A last type of communication can be found between similar modules located at different nodes. For this purpose, the packet header is used. As the framework is modular, i.e. modules can be added, removed or changed, the packet header should be capable of handling the modularity. This can be done by using a modular header structure. Each module can add an extension to the header. When a packet is received by a node, the header is analyzed by the controller module. The controller puts the data of the header in the database, determines which modules need to be activated, starts the first one and passes the required arguments to the module.

7 Future Work

The framework presented in this paper serves as a preliminary proposal for a fully flexible modular architecture for WBANs. Various requirements considered in section 3 are covered, such as coping with heterogeneity, flexibility, ease-of-use, energy efficiency, ... However, several improvements can still be made.

In a first step, we need to properly validate the framework and investigate thoroughly the impact of the framework on the overall performance of the network. The overhead introduced by MOFBAN should be examined. We believe that this overhead will be minimal due to the avoidance of duplication and the

use of the central database. Further we will use the framework to test existing protocols for WBANS such as CICADA [13]. Based on the analysis, we will propose further improvements to have a more energy efficient solution. In a last phase, we envision to put the framework to test in a real-life testbed.

In the communication part, we have introduced the concept of the modular header structure. This structure is beneficial for reducing the overhead between nodes as data can be shared better and no redundant or duplicate data is sent over the network. Future research will specify the use of modular header structure and analyze its performance.

8 Conclusion

In this paper we have proposed a framework that supports the communication protocols in on-body networks. It is designed in such a way that it is more transparent for the application designer. The functionality of the framework is easily adaptable and expandable. This is made possible by the use of modules. Other resulting advantages are:

- Duplication of functionality is avoided and a simple structure is used. The framework can be regarded as being lightweight.
- Heterogeneity and QoS are well supported by using different implementations of the routing module or reliability module.
- Easy to add functionality by simply plugging in a new module in the framework.
- Quickly reconfigurable through the application interface.

Further, we have discussed the communication in the framework. The interaction between the modules is handled by the controller module and information is stored in a common database. Conflicts in the database are avoided by using the controller as gatekeeper. The controller module also takes care of the communication between modules, with the applications and physical layer en between the nodes. It further is responsible for activating the appropriate module at the correct time by using the scheduler. The controller module can be considered as the key element of MOFBAN.

Finally, we are convinced that MOFBAN will proof to be a starting point for the development of new protocols for communication in a WBAN. It will ease the development of new applications and trigger the use of WBANs.

Acknowledgment

This research is partly funded by the Fund for Scientific Research - Flanders (F.W.O.-V., Belgium) project G.0531.05, by The Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) through the contract No. 020152 and a PhD.grant for B. Latré and E. De Poorter and by the IBBT-IM3 project.

References

1. Cypher, D., Chevrollier, N., Montavont, N., Golmie, N.: Prevailing over wires in healthcare environments: benefits and challenges. IEEE Communications Magazine 44(4), 56–63 (2006)
2. Istepanian, R.S.H., Jovanov, E., Zhang, Y.T.: Guest editorial introduction to the special section on m-health: Beyond seamless mobility and global wireless health-care connectivity. Information Technology in Biomedicine, IEEE Transactions on 8(4), 405–414 (2004)
3. Chlamtac, I., Conti, M., Liu, J.J.-N.: Mobile ad hoc networking: imperatives and challenges. Ad Hoc Networks 1(1), 13–64 (2003)
4. Otto, C., Milenkovic, A., Sanders, C., Jovanov, E.: System architecture of a wireless body area sensor network for ubiquitous health monitoring. Journal of Mobile Multimedia 1(4), 307–326 (2006)
5. Park, S., Jayaraman, S.: Enhancing the quality of life through wearable technology. IEEE Engineering in Medicine and Biology Magazine 22(3), 41–48 (2003)
6. Brebels, S., Sanders, S., Winters, C., Webers, T., Vaesen, K., Carchon, G., Gyselinckx, B., De Raedt, W.: 3d sop integration of a BAN sensor node. In: 2005. Proceedings. 55th Electronic Components and Technology Conference, pp. 1602–1606 (May/June 2005)
7. Jovanov, E., Milenkovic, A., Otto, C., de Groen, P.C.: A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. Journal of NeuroEngineering and Rehabilitation 2(1), 16–23 (2005)
8. Ieee 802.15 wpan study group medical body area networks (sg mban)
9. Latré, B., Vermeeren, G., Moerman, I., Martens, L., Demeester, P.: Networking and propagation issues in body area networks. In: 11th Symposium on Communications and Vehicular Technology in the Benelux, SCVT 2004 (November 2004)
10. Braem, B., Latré, B., Moerman, I., Blondia, C., Reusens, E., Joseph, W., Martens, L., Demeester, P.: The need for cooperation and relaying in short-range high path loss sensor networks. In: International Conference on Sensor Technologies and Applications (SENSORCOMM 2007) (accepted, 2007)
11. Reusens, E., Joseph, W., Vermeeren, G., Martens, L., Latré, B., Braem, B., Blondia, C., Moerman, I.: Path-loss models for wireless communication channel along arm and torso: Measurements and simulations. In: IEEE AP-S Internation Symposium 2007 (June 2007)
12. Wegmueller, M.S., Kuhn, A., Froehlich, J., Oberle, M., Felber, N., Kuster, W., Fichtner, N.: An attempt to model the human body as a communication channel. IEEE Transactions on Biomedical Engineering (accepted, 2007)
13. Latré, B., Braem, B., Moerman, I., Blondia, C., Reusens, E., Joseph, W., Demeester, P.: A low-delay protocol for multihop wireless body area networks. In: Mobile and Ubiquitous Systems: Networking & Services, 2007 4th Annual International Conference on, Philadelphia, PA, USA (August 2007)
14. Dokovski, N.T., van Halteren, A.T., Widya, I.A.: Banip: Enabling remote healthcare monitoring with body area networks. In: Guelfi, N., Astesiano, E., Reggio, G. (eds.) FIDJI 2003. LNCS, vol. 2952, pp. 62–72. Springer, Heidelberg (2004)
15. Wac, K.E., Bults, R., van Halteren, A., Konstantas, D., Nicola, V.F.: Measurements-based performance evaluation of 3g wireless networks supporting m-health services. In: Chandra, S., Venkatasubramanian, N. (eds.) Multimedia Computing and Networking 2005. Proceedings of the SPIE, vol. 5680, pp. 176–187 (December 2004)

16. Milenkovic, A., Otto, C., Jovanov, E.: Wireless sensor networks for personal health monitoring: Issues and an implementation. Computer Communications, Wireless Sensor Networks and Wired/Wireless Internet Communications 29(13-14), 2521–2533 (2006)
17. Akkaya, K., Younis, M.: A survey on routing protocols for wireless sensor networks. Ad Hoc Networks 3(3), 325–349 (2005)
18. Ylisaukko-oja, A., Vildjiounaite, E., Mantyjarvi, J.: Five-point acceleration sensing wireless body area network - design and practical experiences. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 184–185. Springer, Heidelberg (2004)
19. Srivastava, V., Motani, M.: Cross-layer design: a survey and the road ahead. Communications Magazine, IEEE 43(12), 112–119 (2005)
20. Melodia, T., Vuran, M., Pompil, D.: The state of the art in cross-layer design for wireless sensor networks. In: Cesana, M., Fratta, L. (eds.) Wireless Systems and Network Architectures in Next Generation Internet. LNCS, vol. 3883, pp. 78–92. Springer, Heidelberg (2006)
21. De Poorter, E., Latré, B., Moerman, I., Demeester, P.: Universal modular framework for sensor networks. In: International Workshop on Theoretical and Algorithmic Aspects of Sensor and Ad-hoc Networks (WTASA 2007), Miami, USA (June 2007)
22. Lymberopoulos, D., Priyantha, N.B., Zhao, F.: mplatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In: IPSN 2007: Proceedings of the 6th international conference on Information processing in sensor networks, pp. 128–137. ACM Press, New York (2007)
23. Penzel, T., Kemp, B., Klosch, G., Schlogl, A., Hasan, J., Varri, A., Korhonen, I.: Acquisition of biomedical signals databases. IEEE Engineering in Medicine and Biology Magazine 20(3), 25–32 (2001)
24. Arnon, S., Bhastekar, D., Kedar, D., Tauber, A.: A comparative study of wireless communication network configurations for medical applications. IEEE [see also IEEE Personal Communications] Wireless Communications 10(1), 56–61 (2003)
25. Paradiso, J.A., Starner, T.: Energy scavenging for mobile and wireless electronics. IEEE Pervasive Computing 04(1), 18–27 (2005)

Performance Analysis for Distributed Classification Fusion Using Soft-Decision Decoding in Wireless Sensor Networks

Jing-Tian Sung[1], Hung-Ta Pai[2], and Bih-Hwang Lee[3]

[1] Dept. of Electrical Engineering
National Taiwan University of Science and Technology
No. 43, Sec. 4, Keelung Rd., Taipei, 106 Taiwan
d9107305@mail.ntust.edu.tw
[2] Graduate Institute of Communication Engineering
National Taipei University
No. 151, University Rd., Sanhsia, Taipei, 237 Taiwan
htpai@mail.ntpu.edu.tw
[3] Dept. of Electrical Engineering
National Taiwan University of Science and Technology
No. 43, Sec. 4, Keelung Rd., Taipei, 106 Taiwan
lee@ccg.ee.ntust.edu.tw

Abstract. Distributed Classification Fusion using Error-Correcting Codes (DCFECC) has recently been proposed for wireless sensor networks. It adopts the Minimum Hamming Distance (MHD) fusion rule and performs much better than traditional classification approaches when the network has faulty sensors. Different fusion rules were proposed later. One of them is Distributed Classification fusion using Soft-decision Decoding (DCSD). The DCSD fusion rule has a considerably lower misclassification probability than the MHD fusion rule. This work analyzes the performance of the DCSD fusion rule. Asymptotic performance approximation of the DCSD fusion rule is derived based on the Central Limit Theorem. Furthermore, an asymptotic upper bound on the misclassification probability is obtained. Finally, numerical simulations are conducted to verify our analysis results.

Keywords: Wireless sensor networks, distributed detection, soft-decision decoding, Central Limit Theorem.

1 Introduction

Wireless sensor networks (WSNs) comprise many tiny, low-cost, battery-powered sensors in a small area. The sensors detect environmental variations and then transmit the detection results to other sensors or a base station. The base station or a sensor, serving as a fusion center, collects all detection results, and determines what phenomenon has occurred [1,2]. The WSN sometimes must be able to function under severe conditions, such as in a battlefield, fireplace or polluted area. The transmission channel, as well as the environmental phenomenon

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 623–634, 2007.

observed by the sensor, is noisy. Furthermore, the observation signal to noise ratio (OSNR) and the channel signal to noise ratio (CSNR) may change quickly and be difficult to estimate accurately. Some sensors may even have unrecognized faults in the harsh environment. Therefore, a fault-tolerant system must be developed to make the received local decisions error-resistant [3,4].

Wang *et al.* [5] proposed Distributed Classification Fusion using Error-Correcting Codes (DCFECC) to solve this problem by combining the distributed detection theory [6] with the concept of error-correcting codes in communication systems [7]. DCFECC with the Minimum Hamming Distance (MHD) fusion rule has a much lower probability of misclassification when some sensors are faulty than the traditional distributed classification method. DCFECC outperforms the method even when CSNR is not correctly estimated. Its performance analysis is given in [8].

Three fusion rules were proposed and compared [9,10] later. One is the maximum *a posteriori* probability (MAP) fusion rule, one is the Minimum Euclean Distance (MED) fusion rule, and the other is Distributed Classification fusion using Soft-decision Decoding (DCSD) fusion rule. The MAP and DCSD fusion rules have a considerably misclassification probability than the MED one. Moreover, the DCSD has a lower computational complexity than the MAP with little performance loss when no faulty sensor appears. If some sensors are defective, the DCSD outperforms the MAP when the misclassification probability is lower than 0.2. Therefore, the DCSD fusion rule is a more practical choice than the other ones. However, its performance analysis have not been provided.

In this work, we analyze the performance of the DCSD fusion rule without assuming no errors in local decisions and wireless channels. Asymptotic performance approximations are obtained by the Central Limit Theorem. Asymptotic upper bounds on the misclassification probability are derived. These results can be utilized for the optimal code matrix design in the future. Computer simulations show the performance approximation is accurate and the upper bound is tight when the misclassification probability is lower than 0.2.

The remainder of this work is organized as follows. Section 2 briefly addresses the distributed detection problem in WSNs and the DCSD fusion rule. The performance analysis of the DCSD fusion rule is derived in Section 3. Section 4 shows simulation results. Concluding remarks and suggestions for future works are given in Section 5.

2 Fault-Tolerant Distributed Detection and DCSD Fusion Rule

Figure 1 depicts a wireless sensor network for distributed detection with N sensors deployed for collecting environment variation data and a fusion center for making a final decision of detections. At the jth sensor, one observation y_j is undertaken for one of phenomena H_i, where $i = 1, 2, \ldots, M$. The observation is normally a real number represented by many bits. Transmitting the real number to the fusion center would consume too much power, so a local decision, u_j, is

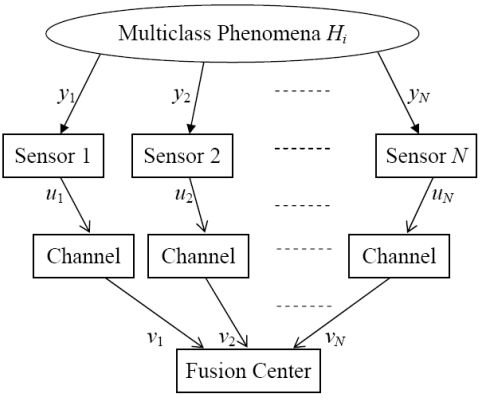


Fig. 1. Structure of a wireless sensor network for distributed detection using N sensors

made instead. For a phenomenon, if only L bits are allowed to send the local decision from the sensor to the fusion center, then the L bits are used to represent the decision.

The DCFECC approach [5] sets $L = 1$, and designs an $M \times N$ code matrix $\mathbf{T}$ not only to correct transmission errors, but also to resist faulty sensors. The application of the code matrix is derived from error-correcting codes. Table 1 lists an example of $\mathbf{T}$, which is the optimal code matrix found through the criterion in [11]. Row i of the matrix represents a codeword $\mathbf{c}_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,N})$ corresponding to hypothesis H_i, and $c_{i,j}$ denotes a 1-bit symbol corresponding to the decision of sensor j. Notably, sensors 1 to 10 have the same decision pattern and sensors 11 to 20 have the same decision pattern. As a result, there are two decision patterns for the code matrix in Table 1.

Table 1. The 4×20 optimal code matrix

H_1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
H_2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
H_3	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
H_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Let v_j be the received local decision at the fusion center, where $v_j \in \{0, 1\}$. A cost function is then defined as

$$C_{\mathbf{v},\mathbf{c}_i} = \begin{cases} 1 - \frac{1}{q}, & \mathbf{c}_i \text{ is one of } q \text{ solutions of} \\ & \arg\min_{\mathbf{c}_k} d_H(\mathbf{v}, \mathbf{c}_k); \\ 1, & \text{else.} \end{cases}$$

Notably, $d_H(\mathbf{v}, \mathbf{c}_k)$ denotes the Hamming distance between a received vector, $\mathbf{v} = (v_1, v_2, \ldots, v_N)$, and a codeword, $\mathbf{c}_k$. Hence, the Bayes risk function [6] represents the probability of misclassification,

$$P_e = \sum_{i,\mathbf{v}} \int_{\mathbf{y}} p(\mathbf{v}, \mathbf{y}, H_i) C_{\mathbf{v},\mathbf{c}_i} d\mathbf{y}, \tag{1}$$

where $\mathbf{y} = (y_1, y_2, ..., y_N)$. Set $\mathbf{u} = (u_1, u_2, \ldots, u_N)$, and make the following assumptions:

Assumption 1: Observations at all sensors are conditionally independent, i.e.,

$$p(\mathbf{y}|H_i) = p(y_1, y_2, \ldots, y_N|H_i) = \prod_{j=1}^{N} p(y_j|H_i).$$

Assumption 2: The jth local decision, u_j, only depends on the jth observation, y_j.

Assumption 3: The jth received local decision, v_j, only depends on the jth local decision, u_j.

Equation (1) can then be recast as

$$\begin{aligned} P_e = \sum_{i,\mathbf{u},\mathbf{v}-v_j} \int_{\mathbf{y}} & P(H_i)[P(\mathbf{v}_{j=1}|\mathbf{u})p(\mathbf{u}|\mathbf{y})p(\mathbf{y}|H_i)C_{\mathbf{v}_{j=1},\mathbf{c}_i} \\ & + P(\mathbf{v}_{j=0}|\mathbf{u})p(\mathbf{u}|\mathbf{y})p(\mathbf{y}|H_i)C_{\mathbf{v}_{j=0},\mathbf{c}_i}]d\mathbf{y}, \end{aligned}$$

where $\mathbf{v}_{j=b_v} = (v_1, \ldots, v_{j-1}, b_v, v_{j+1}, \ldots, v_N)$, $b_v \in \{0, 1\}$, and $\mathbf{v} - v_j$ represents the elements of $\mathbf{v}$ except v_j.

The DCSD is applied as follows. Set $\mathbf{u} = (u_1, u_2, \ldots, u_N)$. The local decision $\mathbf{u}$ is transmitted for the final decision to the fusion center. The received data at the fusion center are $\tilde{\mathbf{v}} = (\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_N)$, where

$$\tilde{v}_j = (-1)^{u_j} \sqrt{\frac{E_s}{L}} + n_j. \tag{2}$$

Notice that E_s is the total transmission energy per sensor, and n_j is the additive white Gaussian noise (AWGN) with the two-sided power spectral density $N_0/2$. The received data are decoded as hypothesis i if

$$p(\tilde{\mathbf{v}}|\mathbf{c}_i) \geq p(\tilde{\mathbf{v}}|\mathbf{c}_k) \text{ for all } \mathbf{c}_k, \text{ where } k = 1, \ldots, M. \tag{3}$$

For simplicity, let $L = 1$. Since $\tilde{v}_j$ does not depend on $c_{i,j}$ given u_j, and according to Assumptions 2 and 3, (3) can be rewritten as

$$\prod_{j=1}^{N} \sum_{b_u=0}^{1} p(\tilde{v}_j|u_j = b_u) p(u_j = b_u|c_{i,j}) \geq \prod_{j=1}^{N} \sum_{b_u=0}^{1} p(\tilde{v}_j|u_j = b_u) p(u_j = b_u|c_{k,j}),$$

$$\Rightarrow \sum_{j=1}^{N} \ln \frac{\sum_{b_u=0}^{1} p(\tilde{v}_j|u_j = b_u) p(u_j = b_u|c_{i,j})}{\sum_{b_u=0}^{1} p(\tilde{v}_j|u_j = b_u) p(u_j = b_u|c_{k,j})} \geq 0. \tag{4}$$

Because $c_{i,j}$ and $c_{k,j}$ are binary, the bit logarithm-likelihood ratio of the received data at the fusion center can be defined as

$$\lambda_j = \ln \frac{\sum_{b_u=0}^{1} p\left(\tilde{v}_j | u_j = b_u\right) p\left(u_j = b_u | c_{i,j} = 0\right)}{\sum_{b_u=0}^{1} p\left(\tilde{v}_j | u_j = b_u\right) p\left(u_j = b_u | c_{k,j} = 1\right)}. \tag{5}$$

(4) is then equivalent to

$$\sum_{j=1}^{N} \left[\lambda_j - (-1)^{c_{i,j}}\right]^2 \leq \sum_{j=1}^{N} \left[\lambda_j - (-1)^{c_{k,j}}\right]^2. \tag{6}$$

3 Performance Analysis

Assume that the wireless channel between the fusion center and the sensor is influenced by AWGN with zero mean and variance σ_c^2. Namely,

$$p\left(\tilde{v}_j | u_j = b_u\right) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left\{-\frac{(\tilde{v}_j - (-1)^{b_u})^2}{2\sigma_c^2}\right\}. \tag{7}$$

For simplicity, let

$$\begin{aligned} P_{j,0|0} &= p\left(u_j = 0 | c_{k,j} = 0\right) \\ P_{j,1|1} &= p\left(u_j = 1 | c_{k,j} = 1\right). \end{aligned} \tag{8}$$

Substituting (7) and (8) into (5), we can rewritte logarithm-likelihood ratio as

$$\begin{aligned} \lambda_j &= \ln \frac{\exp\left\{\frac{\tilde{v}_j}{\sigma_c^2}\right\} P_{j,0|0} + \exp\left\{-\frac{\tilde{v}_j}{\sigma_c^2}\right\} \left(1 - P_{j,0|0}\right)}{\exp\left\{\frac{\tilde{v}_j}{\sigma_c^2}\right\} \left(1 - P_{j,1|1}\right) + \exp\left\{-\frac{\tilde{v}_j}{\sigma_c^2}\right\} P_{j,1|1}} \\ &= \ln \frac{\exp\left\{\frac{2\tilde{v}_j}{\sigma_c^2}\right\} P_{j,0|0} + \left(1 - P_{j,0|0}\right)}{\exp\left\{\frac{2\tilde{v}_j}{\sigma_c^2}\right\} \left(1 - P_{j,1|1}\right) + P_{j,1|1}}. \end{aligned} \tag{9}$$

Thus, the Cumulative Density Function (CDF) of λ_j can be expressed as

$$\begin{aligned} \Pr(\lambda_j < x | c_{i,j}) &= \Pr\left\{ \ln \frac{\exp\left\{\frac{2\tilde{v}_j}{\sigma_c^2}\right\} P_{j,0|0} + \left(1 - P_{j,0|0}\right)}{\exp\left\{\frac{2\tilde{v}_j}{\sigma_c^2}\right\} \left(1 - P_{j,1|1}\right) + P_{j,1|1}} < x \middle| c_{i,j} \right\} \\ &= \Pr\left\{ \tilde{v}_j < \frac{\sigma_c^2}{2} \ln \frac{e^x P_{j,1|1} + P_{j,0|0} - 1}{P_{j,0|0} + e^x (P_{j,1|1} - 1)} \middle| c_{i,j} \right\}. \end{aligned} \tag{10}$$

We denote

$$\zeta_j(x) = \frac{\sigma_c^2}{2} \ln\left(\frac{e^x P_{j,1|1} + P_{j,0|0} - 1}{P_{j,0|0} + e^x(P_{j,1|1} - 1)}\right). \tag{11}$$

Because the Probability Density Function (PDF) of $\tilde{v}_j$ can be represented by

$$\begin{aligned} f_{\tilde{v}_j}(x|c_{i,j}) = &\frac{P_{j,c_{i,j}|c_{i,j}}}{\sqrt{2\pi\sigma_c^2}} \exp\left\{-\frac{x-(-1)^{c_{i,j}}}{2\sigma_c^2}\right\} \\ &+\frac{(1-P_{j,c_{i,j}|c_{i,j}})}{\sqrt{2\pi\sigma_c^2}} \exp\left\{-\frac{x-(-1)^{(1-c_{i,j})}}{2\sigma_c^2}\right\}, \end{aligned} \tag{12}$$

where $P_{j,c_{i,j}|c_{i,j}}$ represents the probability of correct local decision for the sensor j, (10) can be rewritten as

$$\begin{aligned} &\Pr(\tilde{v}_j < \zeta_j(x)|c_{i,j}) \\ &= \int_{-\infty}^{\zeta_j(x)} f_{\tilde{v}_j}(x|c_{i,j})dx \\ &= P_{j,c_{i,j}|c_{i,j}} \times \Phi\left(\frac{\zeta_j(x)-(-1)^{c_{i,j}}}{\sigma_c}\right) \\ &\quad +(1-P_{j,c_{i,j}|c_{i,j}}) \times \Phi\left(\frac{\zeta_j(x)-(-1)^{c_{i,j}}}{\sigma_c}\right), \end{aligned} \tag{13}$$

where $\Phi(\cdot)$ is the CDF of a random variable with normal distribution, i.e.,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left\{-\frac{x^2}{2}\right\} dx.$$

Therefore, the PDF of λ_j can be given by

$$\begin{aligned} &f_{\lambda_j}(x|c_{i,j}) = \frac{d}{dx}\int_{-\infty}^{\zeta_j(x)} f_{\tilde{v}_j}(t|c_{i,j})dt \\ &= \frac{1}{2}\left[\frac{P_{j,c_{i,j}|c_{i,j}}}{\sqrt{2\pi\sigma_c^2}} \exp\left\{-\frac{\zeta(x)-(-1)^{c_{i,j}}}{2\sigma_c^2}\right\}\right. \\ &\quad \left. + \frac{(1-P_{j,c_{i,j}|c_{i,j}})}{\sqrt{2\pi\sigma_c^2}} \exp\left\{-\frac{\zeta(x)-(-1)^{(1-c_{i,j})}}{2\sigma_c^2}\right\}\right] \\ &\quad \times \frac{\sigma_c^2 e^x(P_{j,1|1}+P_{j,0|0})}{(e^x P_{j,1|1}+P_{j,0|0}-1)(e^x P_{j,1|1}+P_{j0|0}-e^x)}. \end{aligned} \tag{14}$$

The mean and the variance of λ_j can be found as

$$\mu_{\lambda_j} = \int_{-\infty}^{\infty} x f_{\lambda_j}(x|c_{i,j})dx, \tag{15}$$

$$\sigma_{\lambda_j}^2 = \int_{-\infty}^{\infty} (x - \mu_{\lambda_j})^2 f_{\lambda_j}(x|c_{i,j})dx, \tag{16}$$

respectively.

The misclassification probability at the fusion center can be expressed by

$$P_e = \sum_{i=1}^{M} \Pr(H_i) \Pr\left(\bigcup_{k=1,k\neq i}^{M} \tilde{H}_k \Big| H_i \right). \tag{17}$$

When the DCSD fusion rule is employed, the misclassification probability given that H_i occurs can be derived as follows.

$$\begin{aligned} \Pr(\tilde{H}_k|H_i) &= \Pr\left(\sum_{j=1}^{N} [\lambda_j - (-1)^{c_{i,j}}]^2 \geq \sum_{j=1}^{N} [\lambda_j - (-1)^{c_{k,j}}]^2 \Big| H_i \right) \\ &= \Pr\left(\sum_{j=1}^{N} \lambda_j \left(c_{k,j} - c_{i,j}\right) \leq 0 \Big| H_i \right), \end{aligned} \tag{18}$$

where $k \neq i$. Let S be the number of the decision patterns for a code matrix and $\Omega(\ell), \ell = 1, 2, ..., S$, be the set of sensors with the same decision pattern ℓ. For example, $\Omega(1) = \{1, 2, ..10\}$ and $\Omega(2) = \{11, 12, ..20\}$ for the code matrix in Table 1. Since the sensors with the same decision pattern operate identically, the means and the variances of λ_j, where $j \in \Omega(\ell)$, have no difference and can be denoted as μ_ℓ and σ_ℓ^2, respectively. Moreover, define $d_H^{(\ell)}(\mathbf{c}_k, \mathbf{c}_i)$ as the partial Hamming distance between $\mathbf{c}_k$ and $\mathbf{c}_i$ at the set ℓ. For example, $d_H^{(1)}(\mathbf{c}_1, \mathbf{c}_2) = 0$ and $d_H^{(2)}(\mathbf{c}_1, \mathbf{c}_3) = 10$ for the code matrix in Table 1. The sensor sets $\Omega(\ell)$ satisfying $d_H^{(\ell)}(\mathbf{c}_i, \mathbf{c}_k) \neq 0$ are employed to differentiate H_k from H_i at the fusion center. When the information from two or more sensor sets are utilized for the final decision, the Hamming distance between $\mathbf{c}_i$ and $\mathbf{c}_k$ is large. Because of the large Hamming distance, the probability of misclassification is small. Therefore, when $\|\{\ell : d_H^{(\ell)}(\mathbf{c}_i, \mathbf{c}_k) \neq 0\}\| \geq 2$, (18) can be rewritten and approximated as

$$\begin{aligned} \Pr(\tilde{H}_k|H_i) &= \Pr\left(\sum_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i, \mathbf{c}_k) \neq 0\}} \sum_{j \in \Omega(\ell)} \lambda_j \left(c_{k,j} - c_{i,j}\right) \leq 0 \Big| H_i \right) \\ &\approx \prod_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i, \mathbf{c}_k) \neq 0\}} \Pr\left(\sum_{j \in \Omega(\ell)} \lambda_j \left(c_{k,j} - c_{i,j}\right) \leq 0 \Big| H_i \right). \end{aligned} \tag{19}$$

The following corollary can be obtained based on the Central Limit Theorem.

Corollary 1. *If $d_H^{(\ell)}(\mathbf{c}_k, \mathbf{c}_i)$ is sufficiently large, the misclassification probability can be approximated as*

$$\Pr(\tilde{H}_k|H_i) \approx \prod_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i,\mathbf{c}_k)\neq 0\}} \Phi\left(-\frac{\sqrt{d_H^{(\ell)}(\mathbf{c}_k,\mathbf{c}_i)}\times(c_{k,j}-c_{i,j})\,\mu_\ell}{\sigma_\ell}\right). \tag{20}$$

If the size of the code matrix is large, it is difficult to calculate the approximation according to (20). Since the probability of the union in (17) can be approximated as

$$\sum_{k=1,k\neq i}^{M} \Pr\left(\tilde{H}_k|H_i\right), \tag{21}$$

we can obtain the following approximation.

Corollary 2. *If $d_H^{(\ell)}(\mathbf{c}_k, \mathbf{c}_i)$ is sufficiently large, the misclassification probability can be approximated as*

$$P_e \approx \sum_{i=1}^{M}\sum_{k=1,k\neq i}^{M} \Pr(H_j)\times \prod_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i,\mathbf{c}_k)\neq 0\}} \Phi\left(-\frac{\sqrt{d_H^{(\ell)}(\mathbf{c}_k,\mathbf{c}_i)}\times(c_{k,j}-c_{i,j})\,\mu_\ell}{\sigma_\ell}\right).$$

Define

$$P_e^* = \sum_{i=1}^{M}\sum_{k=1,k\neq i}^{M} \Pr(H_j)\times \prod_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i,\mathbf{c}_k)\neq 0\}} \Phi\left(-\frac{\sqrt{d_H^{(\ell)}(\mathbf{c}_k,\mathbf{c}_i)}\times(c_{k,j}-c_{i,j})\,\mu_\ell}{\sigma_\ell}\right).$$

Next, we propose a corollary to derive the upper bound of the approximation.

Corollary 3. *For all ℓ and all pair $\{i,k\}$, if*

$$\Phi\left(-\frac{\sqrt{d_H^{(\ell)}(\mathbf{c}_k,\mathbf{c}_i)}\times(c_{k,j}-c_{i,j})\,\mu_\ell}{\sigma_\ell}\right) \le \Pr\left(\sum_{j\in\Omega(\ell)}(c_{k,j}-c_{i,j})\,\tilde{v}_j \le 0\right), \tag{22}$$

then

$$P_e^* \le \sum_{i=1}^{M}\sum_{k=1,k\neq i}^{M} \Pr(H_i)\times \prod_{\{\ell: d_H^{(\ell)}(\mathbf{c}_i,\mathbf{c}_k)\neq 0\}} \Pr\left(\sum_{j\in\Omega(\ell)}(c_{k,j}-c_{i,j})\,\tilde{v}_j \le 0\right). \tag{23}$$

From (12), the characteristic function of $\tilde{v}_j$ is

$$\varphi(z) = P_{j,c_{i,j}|c_{i,j}} \exp\left\{\mathbf{j}\,(c_{k,j} - c_{i,j})\,z - \frac{\sigma_c^2 z^2}{2}\right\} + (1 - P_{j,c_{i,j}|c_{i,j}}) \exp\left\{-\mathbf{j}\,(c_{k,j} - c_{i,j})\,z - \frac{\sigma_c^2 z^2}{2}\right\} \tag{24}$$

and the characteristic function of a random variable which is the summation of $\tilde{v}_j, j = 1, 2, ..., n$, is

$$\varphi^n(z) = \sum_{t=0}^{n} \binom{n}{t} (P_{j,c_{i,j}|c_{i,j}})^t (1 - P_{j,c_{i,j}|c_{i,j}})^{(n-t)} \times \exp\left\{\mathbf{j}(2t - n)\,(c_{k,j} - c_{i,j}) - \frac{z^2}{2}(n\sigma_c^2)\right\}, \tag{25}$$

where $\mathbf{j} = \sqrt{-1}$. Then, the PDF of a random variable which is the summation of $\tilde{v}_j, j = 1, 2, ..., n$, is

$$f_{\sum \tilde{v}_j}(x) = \frac{1}{\sqrt{2\pi n\sigma_c^2}} \sum_{t=0}^{n} \binom{n}{t} (P_{j,c_{i,j}|c_{i,j}})^t (1 - P_{j,c_{i,j}|c_{i,j}})^{(n-t)} \times \exp\left\{-\frac{(x - (2t - n)\,(c_{k,j} - c_{i,j}))^2}{2n\sigma_c^2}\right\}. \tag{26}$$

Let $w_\ell = d_H^{(\ell)}(\mathbf{c}_k, \mathbf{c}_i)$. According to Corollary 3, when (22) holds, the upper bound can be expressed as

$$P_e^* \le \sum_{i=1}^{M} \sum_{k=1, k\neq i}^{M} \Pr(H_i) \prod_{\{\ell: w_\ell \neq 0\}} \sum_{t=0}^{w_\ell} \binom{w_\ell}{t} (P_{j,c_{i,j}|c_{i,j}})^t \times (1 - P_{j,c_{i,j}|c_{i,j}})^{w_\ell - t} \Phi\left(-\frac{(2t - w_\ell)\,(c_{k,j} - c_{i,j})}{\sqrt{w_\ell}\sigma_c}\right). \tag{27}$$

4 Numerical and Simulation Results

The proposed approximations and the upper bound are verified by simulations with 10^6 Monte Carlo tests. A fusion center and $N = 20$ sensors are deployed to detect and classify four hypotheses H_1, H_2, H_3 and H_4. We also assume that the local observations are interfered by the Gaussian noise with the same standard deviation σ_o and mean $0, 1, 2$, and 3, respectively. In addition, wireless channels are interfered by AWGN and CSNR is $10 \times \log_{10}(E_s/N_0)$. The code matrix in Table 1 was utilized.

The first and second approximations are stated in Corollary 1 and Corollary 2, respectively. The first set of figures shows the approximations and the simulation

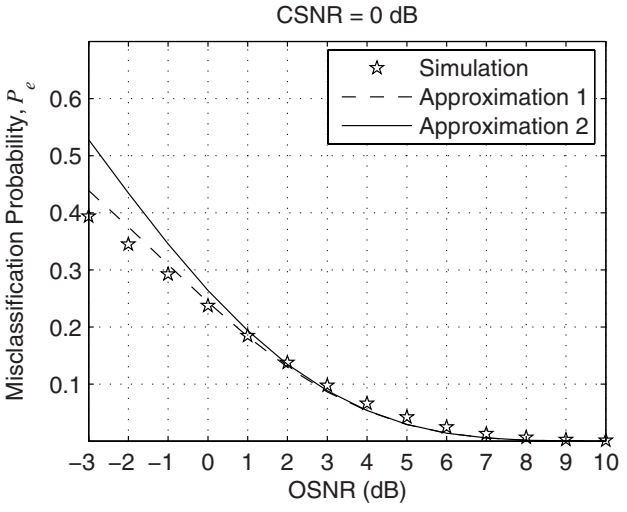


Fig. 2. Proposed approximations and simulation results when CSNR = 0 dB

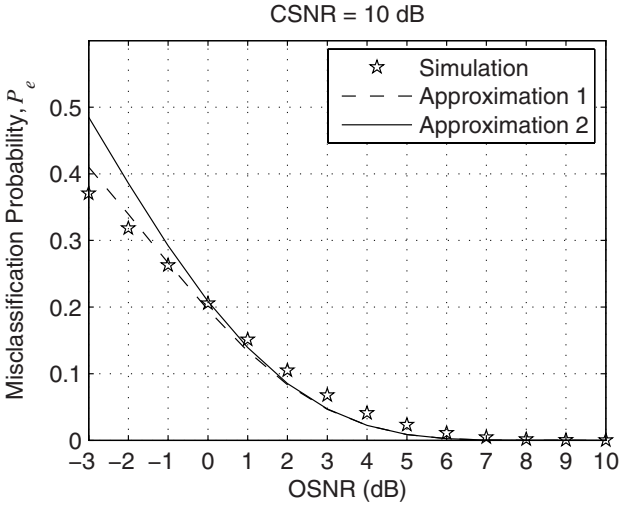


Fig. 3. Proposed approximations and simulation results when CSNR = 10 dB

result when CSNR is set to be 0 and 10, respectively. In this case, M is small and the probability of the union in (17) is obtainable. As shown in Fig. 2 and 3, both approximations are accurate when the misclassification probability is lower than 0.2. The first approximation is better than the second one. However, the computational complexity of the first approximation is higher than the second one, as we pointed in the previous section. When OSNR is low, the probability of

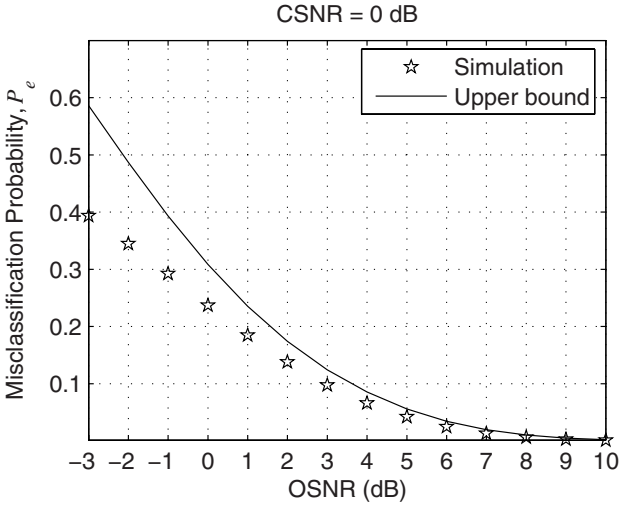


Fig. 4. Proposed upper bounds and simulation results when CSNR = 0 dB

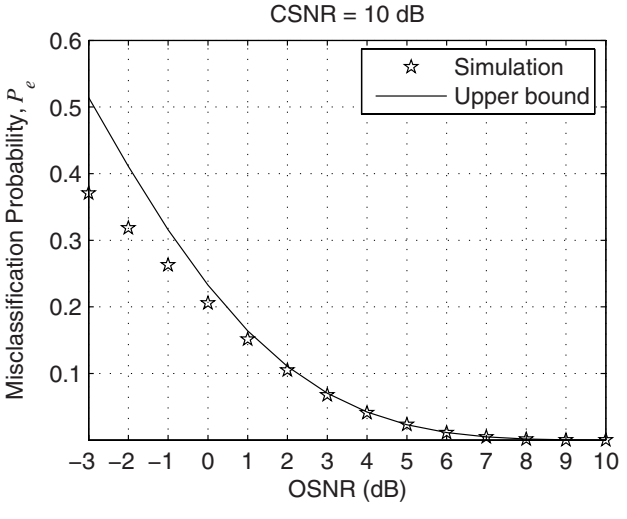


Fig. 5. Proposed upper bounds and simulation results when CSNR = 10 dB

the union in (17) cannot be approximated by (21). Thus, the difference between the approximation and the simulation result is large at −3 dB. Figures 4 and 5 show that the upper bound in (27) is very close to the simulation result when the misclassification probability is lower than 0.2.

5 Conclusions and Future Works

This work analyzes the performance of the distributed detection using the DCSD fusion rule. Two approximations and an upper bound of the misclassification probability are presented. The analysis is based on the Central Limit Theorem. The simulation results showed that the approximation and the upper bound are accurate for the network with only twenty sensors. In the future, we will employ the analysis result to design the optimal code matrix for the DCSD fusion rule.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. IEEE Commun. Mag. 38, 102–114 (2002)
2. Aldosari, S.A., Moura, J.M.F.: Detection in decentralized sensor networks. In: Proc. ICASSP 2004, Montreal, Canada (2004)
3. Meyer, G.G.L., Weinert, H.L.: On the design of fault-tolerant signal detectors. IEEE Trans. Acoust. Speech, Signal Processing 34(4), 973–978 (1986)
4. Reibman, A.R., Nolte, L.W.: Optimal fault-tolerant signal detection. IEEE Trans. Acoust. Speech, Signal Processing 38(1), 179–180 (1990)
5. Wang, T.Y., Han, Y.S., Varshney, P.K., Chen, P.N.: Distributed fault-tolerant classification in wireless sensor networks. IEEE J. Select. Areas Commun. 23(4), 724–734 (2005)
6. Varshney, P.K.: Distributed Detection and Data Fusion. Springer, New York (1997)
7. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. Elsevier, New York (1977)
8. Yao, C., Chen, P.N., Wang, T.Y., Han, Y.S., Varshney, P.K.: Performance analysis and code design for minimum hamming distance fusion in wireless sensor networks. IEEE Trans. Inform. Theory 53(5), 1716–1734 (2007)
9. Chen, P.N., Wang, T.Y., Han, Y.S., Wang, Y.T.: On the design of soft-decision fusion rule for coding approach in wireless sensor networks. In: Int. Conf. on Algorithms, Systems, and Applications, Xian, P. R. China, pp. 140–150 (2006)
10. Wang, T.Y., Han, Y.S., Chen, B., Varshney, P.K.: A combined decision fusion and channel coding scheme for distributed fault-tolerant classification in wireless sensor networks. IEEE Trans. Wireless Commun. 5(7), 1695–1705 (2006)
11. Pai, H.T., Han, Y.S., Sung, J.T.: Two-dimensional coded classification schemes in wireless sensor networks. IEEE Trans. Wireless Commun. (submitted)

Hard Constrained Vertex-Cover Communication Algorithm for WSN

Maytham Safar and Sami Habib

Kuwait University
Computer Engineering Department
P. O. Box 5969 Safat 13060 Kuwait
{maytham,shabib}@eng.kuniv.edu.kw

Abstract. The communication problem is to select a minimal set of placed sensor devices in a service area so that the entire service area is accessible by the minimal set of sensors. Finding the minimal set of sensors is modeled as a vertex-cover problem, where the vertex-cover set facilitates the communications between the sensors in a multi-hop fashion keeping in mind the limited communication range and battery lifespan of all sensors. The vertex-cover is a subset of the coverage set of sensors; therefore, we transform the search space from a continuous domain into a discrete domain. We encoded the vertex-cover problem into the evolutionary domain, where the objective function is to select a minimal set of sensors out of the coverage sensors to act as the vertex-cover set so that its communication range covers all the coverage sensors. The experimental results demonstrate the feasibility of our evolutionary approach in finding minimal vertex cover set, which is less than 37% of total sensors used as communication sensors, in under 14 seconds with 100% coverage of the sensor nodes in wireless sensor network.

Keywords: Wireless sensor network, communication, vertex cover, discrete space, optimization, evolutionary approach.

1 Introduction

The wireless sensor network (WSN) has emerged as a promising platform to monitor an area with minimal human interventions. Advancements in low-power micro-electronic circuits, wireless communications, and operating systems have made WSN into feasible platforms that are used in many applications. Initially, the WSN applications were dominated and funded by the military applications, such as monitoring the activity in a battle field. Now, many civilian applications, such as environmental and habitat monitoring have emerged to benefit from the usage of WSN.

There are two core problems that should be considered by deployment of any wireless sensor networks. These problems are the coverage and communication problems. The coverage problem is to place sensor devices in a service area so that the entire service area is covered. In a previous work [17], we proposed a heuristic model that maps the coverage problem into two sub-problems: floorplan and placement, which are mimicking the placement and integration modules of integrated

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 635–649, 2007.

circuit (IC) into a circuit board. The floorplan problem is to divide the circuit board into well-defined geometric cells, and then the placement problem determines the best cells to place the IC modules into them with minimal total wire connections. A combined optimization of floorplan and placement was coded in an evolutionary approach and found good coverage solutions as defined by the measure of quality of coverage [18].

In this work we focus on the communication problem, which will assume that sensor networks consist of two types of sensor devices. The first type of sensors (coverage sensors) is responsible for sensing/monitoring the surrounding environment, and generates data packets periodically. Those are the sensor devices that we got as a result of applying our evolutionary coverage algorithm. They are also responsible for forwarding the data they receive from other sensors towards a second type of sensors (named communication sensors). Communication sensors are responsible for collecting all the data generated by the coverage sensors. Communication sensors have sufficient processing capability and more power supply that make their communication ranges cover the whole service area. One of the challenges imposed by such sensor networks, is the communication sensor placement problem. We define the communication sensor placement problem as how to select the minimal number of communication sensors out of the set of the coverage sensors while maximizing the communication range of the communication sensors in the service area taking into consideration the traffic intensity distribution in the area. Communication sensors have significant impact on sensor network performance. Despite its significance, results on this problem remain limited, particularly theoretical results that can provide performance guarantee.

In this work, we develop a heuristic algorithm that is based on the vertex cover approach. The vertex cover problem is the optimization problem of finding a vertex cover of minimum size in a graph, where we assume that each vertex cover represents a communication sensor and the covered nodes are the rest of the coverage sensors. Finding the minimum vertex cover is an NP-complete problem. However, by using some heuristics we can obtain a vertex cover set, which is in the worst case at most twice that of the optimal. Our algorithm provides solutions specifying coverage sensors that can be used as communication sensors and minimizes the number of the communication sensors, while providing a satisfactory quality of service to the users. This is accomplished by trying to cover the largest set of the coverage sensors, and hence covering a maximum possible part of the service area. The goodness of a solution depends on how it minimizes the number of the communication sensors while maximizing the communication coverage of the sensors in the service area. In the rest of the paper, we will use the terms base stations and communication sensors interchangeably.

The rest of the paper is organized into five sections. Section 2 describes the related work with respect to the coverage problem and communication problem in wireless sensor networks. Section 3 contains the mathematical formulation of the communication problem. Section 4 describes our evolutionary approach in solving the communication problem. Section 5 illustrates our experimental results generated by the proposed evolutionary methodology. Section 6 contains the conclusion and future directions.

2 Related Work

Recently, many researchers have been investigating and developing deployment strategies that give the optimal base stations placement for guaranteed coverage, connectivity, bandwidth and robustness, taking into consideration numerous factors such as traffic density, channel condition, interference scenario, the number of base stations, …etc. The objectives are either minimizing the number of base stations deployed, minimizing the total cost, minimizing the energy consumption, or maximizing the number of served sensors by a base station, maximizing network lifetime, and maximizing the network utilization.

Therefore, many researchers have focused their efforts on reducing network traffic of these sensor networks [2, 5, 7, 11, 12, 15]. Many other researchers have focused on minimizing the number of base stations [2, 5, 8, 13, 16], where others used a predefined fixed number of base stations [1, 9, 11, 12, 14]. Also, some researchers have focused on maximizing the number of sensors served by a base station [3]. However, most of the above researches have assumed that the number of sensors served by any base station is fixed [1, 2, 5, 9, 12].

The base stations in these applications may be arranged in wired networks, and may in general pose considerable technical problems in data processing, communication, and management. The base stations can also be arranged into wireless networks. This even pose more technical challenges because of their dynamic structures and more constrained energy and bandwidth capabilities. Thus, the base stations placement has been formulated in various ways.

The strategy reported in [1] aimed to find a base station configuration that ensures each user to communicate with a satisfactory signal to-interference ratio (SIR) in a wireless CDMA system. The solution is guaranteed to be optimal and considered coverage, capacity and cost but not interference. The work in [2] is an adaptation to the recent bio-inspired optimization approach, Particle Swarm Optimization (PSO), to form a suitable algorithm that converges with a faster rate than genetic algorithms. Two important factors are considered simultaneously, coverage and economic. Another work in [3] describes an application of combinatorial optimization to the problem of designing cellular mobile telephone wireless networks. The goal of the network design problem is to cover the maximum number of subscribers in an effective and efficient manner. Work [6] focuses on the problem of where to place base stations to yield high capacity and efficiency in term of channel quality and spectral. One of the key objectives is to allow many users to co-exist in a relatively small area while maintaining spectral efficiency, system capacity and channel quality. New dynamic base station selection technique for overlapping cell placement based on robust traffic performance for personal communication systems in fluctuating and heavily tapered traffic is suggested in [7]. The proposed technique improves the blocking probability and carried traffic performance. It enhances the robustness of a system for congested traffic due to moving of the subscribers even if the base station has few channels. The authors in [8] addressed the problem of placing the sensor nodes, relay nodes and base stations in the sensor field such that each point of interest in the sensor field is covered by a subset of sensors of desired cardinality. Several deployment strategies to determine optimal placement of the nodes for guaranteed coverage, connectivity, bandwidth and robustness are considered in this paper.

The authors of paper [9] have studied two phases of installation process, the placement of the base stations and assignment of frequency channels in WLAN networks. They aimed to reduce installation costs, minimize interferences of signals between channels and improve the network throughput. The processes that have been taken to choose the best placement of base stations are to map the demand area by dividing it into small quadrangular pieces of demand points. Next, choose candidate locations that offer low cost of installation and good attendance area. Then signal measurement for the signal level received by each candidate base station on each demand point. Finally, they defined the computational model which is developed using integer linear programming computational model. Limited number of base stations and candidate locations are used. The strategy represented in [10] aimed to find optimization methods for base station placement in wireless applications. The authors suggest that Nelder-Mead method or some other direct search method will be highly effective for many formulations, particularly as reliable problem-specific initialization heuristics are developed. A set of non linear programming models are developed based on the Hooke and Jeeves method, quasi Newton, and conjugate gradient search algorithms to search the optimal location of transmitters to serve specified distributions of receivers. In [12] the authors considered an alternate objective, that is, to determine the base station positions and transmission power levels so as to maximize the minimum throughput among the mobiles, according to their study both of them determine the coverage area and the signal to interference ratio, and hence influence the system capacity. In [13] an approach for automatic base station placement is presented. An optimization strategy forms the core of the automatic process which not only determines the number of base stations and their locations but also base station configurations. It aims at designing a high-quality network that guarantees the system performance; i.e. meets the requirement of the coverage capacity, and interference level, while trying to minimize the required bandwidth and the cost involved in building such a network. The number of base stations and their locations and the transmissions power are defined. In [15] the authors analyzed the problem of automatic base station placement and used a hierarchical approach to solve the problem. A fuzzy expert system was developed to determine the optimal base station parameters. A numerical experiment was made for adjusting the transmitted power to reduce the interference and to distribute traffic equally to the cells so that the frequency cost is minimized. The objective function was based on several weighted factors, such as covered area, interference area, and mean signal path loss. The authors in paper [16] have developed a computer aided planning tool known as POPULAR, which stands for a planning of Pico cellular radio. Planning must take into account the specifics of radio wave propagation at the installation site. POPULAR computes the minimal number of base stations and their locations given a blueprint of the installation site and information about the wall and ceiling materials. The internal technique within POPULAR, depends on the number of assigned test points inside the building to be covered.

In this work, we consider similar formulation of the coverage problem as discussed in [17, 19, 20]. However, we have assumed that the cell size is not fixed, and the service area can be floorplanned in arbitrary ways. Also, we used object-oriented classes to represent chromosomes and their genes. Our evolutionary methodology is attached with a sensor device library with heterogeneous features, such as the radius of coverage (ranging from 1 meter to 50 meters) and cost (ranging from $50 to $1000).

3 Communication Problem Formulation

We are given a two-dimensional service area (A) with width (W) and height (H) as shown in Figure 1. The service area is an obstacle-free. Also, the service area A is already divided into M x N cells, where each cell can possibly contain a sensor device at its center of mass. All the centers of mass represent demand points, which were considered as the candidate locations for the sensor devices for the coverage problem. Thus, a set of placed sensors for the coverage problem, B, is given as an input to the communication problem. Each element in the set B is a tuple, b_i, consisting of six ordered parameters, $b_i = <S_j, C_{MxN}, RC, SC, CR, BL>$. The parameter S_j refers to the sensor identification, which was allocated from the sensor device library S. The parameter C_{MxN} represents the physical cell location of the placed sensor within the service area. The indices M and N refer the column and row numbers respectively of the floorplan of the service area, as shown in Figure 1. The parameter RC indicates the radius of coverage in meters of the placed sensor S_j. The parameter SC refers to the initial installation and deployment cost in Dollars ($) of the placed sensor S_j. The parameter CR refers to the communication radius that the radio signal within the placed sensor (S_j) can reach in meters. The range of CR varies with the consumption of power. The last parameter BL indicates the current battery level of the placed sensor S_j.

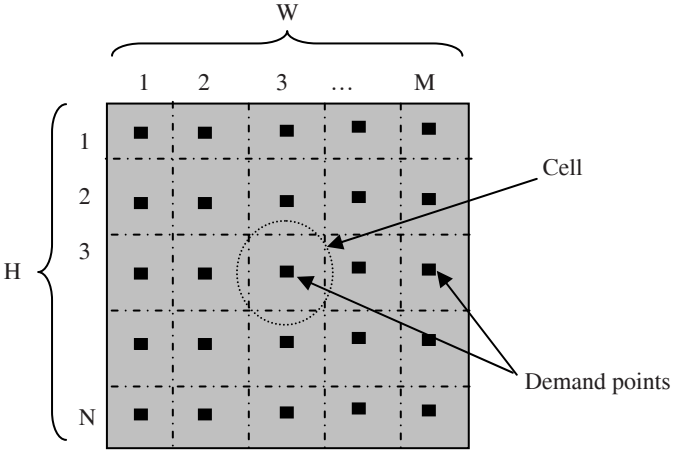


Fig. 1. A service area to be monitored by WSN

Also, the total coverage (TC), which represents the ratio of the total non-overlapping of all placed sensors' radius of coverage over the total area of service area (WxH), is given as an input to the communication problem. The communication problem is to determine a minimal subset C of B ($C \subseteq B$) such that the communication radiuses (CRs) of all selected sensors (vertex covers) within B can reach all other sensors in $\hat{C} = B - C$; moreover, the sensors in $\hat{C}$ should be as far as possible away from the radius of coverage of all selected vertex covers in C, as

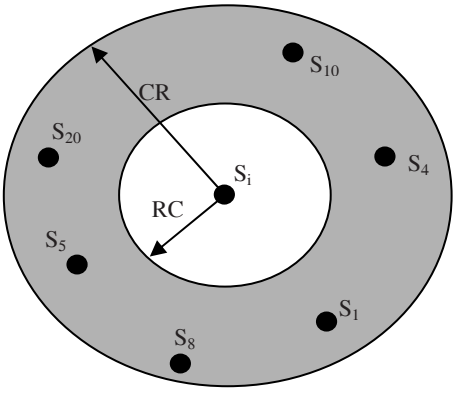


Fig. 2. The relationship between the communication and coverage ranges of s vertex-cover

illustrated in Figure 2. There are three possible relations between the communication radius (CR) and the radius of coverage (RC) of a sensor:

1. CR = RC,
2. CR < RC, and
3. CR > RC.

The first two relations, where the communication radius is equal or less than the radius of coverage respectively, are not considered in this paper. According to the leading company in the development of WSN [21], the relations 1-2 are not considered due to their impractical usage in the field. In this paper, we considered the third relation (CR > RC), where the communication radius is greater than the radius of coverage (sensing range). If a sensor is selected as a vertex cover, then there should be a minimal number of sensors in its sensing range (within the white-circle $\pi \times RC^2$) as illustrated in Figure 2. Also, all the sensors within the shaded circle $((\pi \times CR^2) - (\pi \times RC^2))$ can be bound to the vertex cover S_i. Our objective function is to achieve a minimal vertex cover set as stated in Equation (1). δ_k represents an allocation variable of a vertex cover; δ_k = 1 indicates that a sensor device *k* has been allocated to be used as a vertex cover. This objective function is subject to a set of constrains (2), (3), (4), (5), (6), (7) and (8).

$$\min \sum_{k \in B} \delta_k \tag{1}$$

$$1 \le |C| \le \frac{|B|}{z} \tag{2}$$

$$1 \le \sum_{k \in C} \omega_{k,j} \le |C|, \text{ for given sensor } j \in \hat{C} \tag{3}$$

$$L \leq \sum_{j \in \hat{C}} \omega_{k,j} \leq U, \text{ for given vertex cover } k \in C \tag{4}$$

$$\left(\omega_{k,j} \times C_{w,h}\right) \notin \left(\delta_k \times \left(\pi \times RC_k^2\right)\right), \text{ for } \forall k \in C,\ \forall j \in \hat{C} \tag{5}$$

$$\left(\omega_{k,j} \times C_{w,h}\right) \in \left(\delta_k \times \left(\pi \times CR_k^2 - \pi \times RC_k^2\right)\right), \text{ for } \forall k \in C,\ \forall j \in \hat{C} \tag{6}$$

$$\left(\delta_i \times \left(\pi \times RC_i^2\right)\right) \cap \left(\delta_j \times \left(\pi \times RC_j^2\right)\right) \cong \phi, \text{ where } i \neq j \text{ and } i, j \in C \tag{7}$$

$$\delta, \omega \in \{0,1\} \tag{8}$$

Constraint (2) ensures that the cardinality of the vertex cover set C is bound between one and the cardinality of the entire sensors set B divided by some given value (z). Constraint (3) ensures that each sensor, which is not selected as a vertex cover, must be bound to at least one vertex cover. $\omega_{k,j}$ is a binding variable; $\omega_{k,j}$ = 1 indicates that a sensor j is bound to a vertex cover k. Otherwise, the sensor j is bound to different vertex cover. Constraint (4) ensures that the bound sensors to an allocated vertex cover are restricted between a lower (L) and upper (U) values. The values of L and U are determined by the designers, and also they are used to create a load-balance workload for each vertex cover. Constraint (5) ensures that the number of sensors located within the sensing range of a vertex cover is minimized and cannot be more than the total number of vertex cover sensors. Moreover, Constraint (6) ensures that a sensor is located within the communication range of its vertex cover excluding its sensing range. Constraint (7) ensures that the coverage ranges of two vertex cover sensors are not overlapping, hence, ensures that no vertex cover sensor is located within the sensing range of another vertex cover sensor. Finally, Constraint (8) defines the allocation (δ) and binding (ω) variables as a Boolean.

4 Evolutionary Approach for Solving the Communication Problem

The selection problem of communication sensors requires an enormous computational effort to achieve optimal solutions. Therefore, we have selected the Genetic Algorithm (GA) to search the discrete design space for good solutions. GA uses a population of chromosomes, which represent the candidate solutions, to evolve toward better solutions. Through some genetic operators such as a mutation and crossover, these chromosomes would reach the optimum or near-optimum solutions. The evolution process starts from a population of chromosomes generated by applying the coverage algorithm, and occurs over a number of generations. In each generation, multiple chromosomes are stochastically selected from the current population, modified using different operators to form a new offspring, which becomes the new chromosomes in the next iteration of the algorithm. The basic

structure of GA, as shown in Figure 3, is a powerful search technique that is used to solve many combinatorial problems.

The genetic algorithm starts with an initial population P (t=0) of solutions encoded as chromosomes. Each chromosome is made of a sequence of genes and every gene controls the inheritance of specific attributes of the solution's characteristics. A fitness function measures the quality of the chromosome (number of communication sensors, and number of sensors covered by their sensing and communication ranges). A fit chromosome suggests a better solution. In the evolution process relatively fit chromosomes reproduce new chromosomes and inferior chromosomes die. This process continues until a chromosome with desirable fitness is found. These selected chromosomes, known as parents, are used to reproduce the next generation of chromosomes, known as offspring.

```
Genetic Algorithm:
1 begin
2         t = 0;
3         initialize chromosomes P(t);
4         evaluate chromosomes P(t);
5         while (termination conditions are unsatisfied)
6         begin
7           t = t + 1;
8           select P(t) from P(t-1);
9           mutate some of P(t);
10          crossover some of P(t);
11          evaluate chromosomes P(t);
12        end
13 end
```

Fig. 3. The basic structure of Genetic Algorithm

The evolution process involves two genetic operations namely, mutation and crossover. A mutation operator arbitrarily alters one or more genes of a randomly selected chromosome. The intuition behind the mutation operator is to introduce a missing feature in the population. Our mutation replaces an existed communication sensor device with a new one from the list of coverage sensors.

A crossover operator combines features of two selected chromosomes (parents) to form two similar chromosomes (offspring) by swapping genes of the parent chromosomes. The intuition behind the crossover operator is to exchange information between different potential solutions.

4.1 Chromosome Representation

We represent a solution of communication sensors selection problem as three object-oriented link lists, as shown in Figure 4. The first link-list represents the population, which contains all chromosomes. The second link-list, which is attached with every chromosome class, represents how many sensor devices have been allocated and bound to a chromosome. The third link-list, which is attached with every chromosome class, represents the vertex cover nodes.

We combined all chromosomes in the population of size *P* into one data structure, which comprises of one link list representing all chromosomes and each chromosome has one link list representing all its genes, where each gene symbolizes a sensor device that is allocated as a base station. Two types of nodes are declared as two different classes, where first class represents a chromosome's attributes and the second class represents a gene's features. Also, we maintain a dynamic matrix to illustrate all cells and demand points.

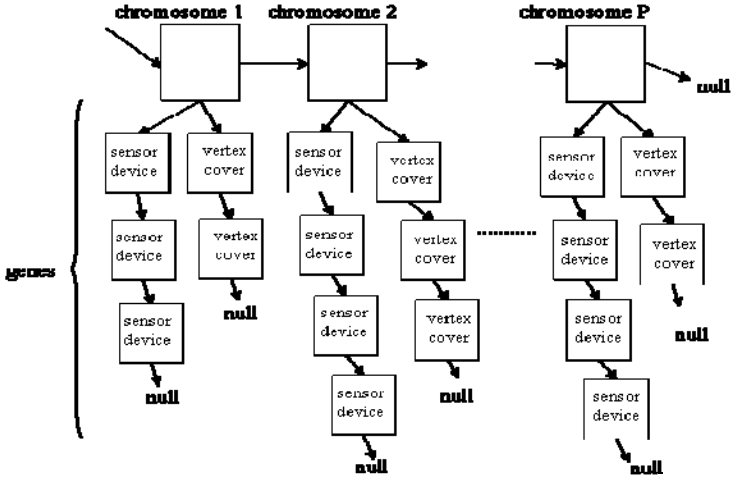


Fig. 4. A population of Chromosomes

5 Experimental Results

To test our evolutionary methodology for the communication problem in wireless sensor networks, we first ran our previously developed code for the coverage problem to find a good solution to the coverage problem of a sensor wireless network. We have coded the coverage within WSN using the evolutionary methodology, which searches for good solutions using JAVA as a programming language. For this experiment, we assigned the population size, the number of generations, the crossover rate and mutation rate to be 100, 1000, 0.45, and 0.25 respectively. The budget threshold *C* is set to $150,000, and we have used a cell size of 30 meters by 30 meters; moreover, we maintained 10 cells by 10 cells as a service area. We choose the solution that consisted of 45 sensors with an average cost of $25,334, and an average coverage ratio of 86.71% of the service area. The coverage ratio represents the total amount of service area that is covered by the sensing range of the sensor devices. Next, we ran our developed methodology for the communication problem on the previous network setup to pick up the least number of the coverage sensors as vertex cover (communication) sensors that would have the maximum communication ratio. The communication ratio represents the total number of coverage sensors that are covered by the communication range of the sensor devices considered as vertex cover (communication) sensors.

For the next experiments, we started by assuming that all the coverage sensors are considered as communication sensors (vertex covers). Hence, 100% of the coverage sensors are covered by the vertex covers, i.e., the communication sensors would have the exact same coverage ratio as the coverage sensors. Then, we applied our evolutionary methodology to reduce the number of required vertex covers, while maintaining the exact same coverage ratio. To achieve such a coverage ratio, we only used the mutation operation that removes or replaces a vertex cover sensor from a solution if and only if the 100% coverage ratio is not affected. In addition, we have ignored the crossover operation, since we cannot guarantee that we end up with a solution that has 100% coverage ratio as we started with. All the experiments are executed on a PC platform and each experimental run for 1000 generations of the communication problem took under 14 seconds. Moreover, in all of our experiments, as the number of generations increased, the behavior of our algorithm changed and then it reaches a plateau after 240 generations. In each of the Figures 5, 6, and 7, we illustrate six curves that tracked the average behavior of the whole population with respect to the number of vertices chosen as vertex covers and how many other vertex covers are in their communication range. This behavior is measured as the number of generations increased.

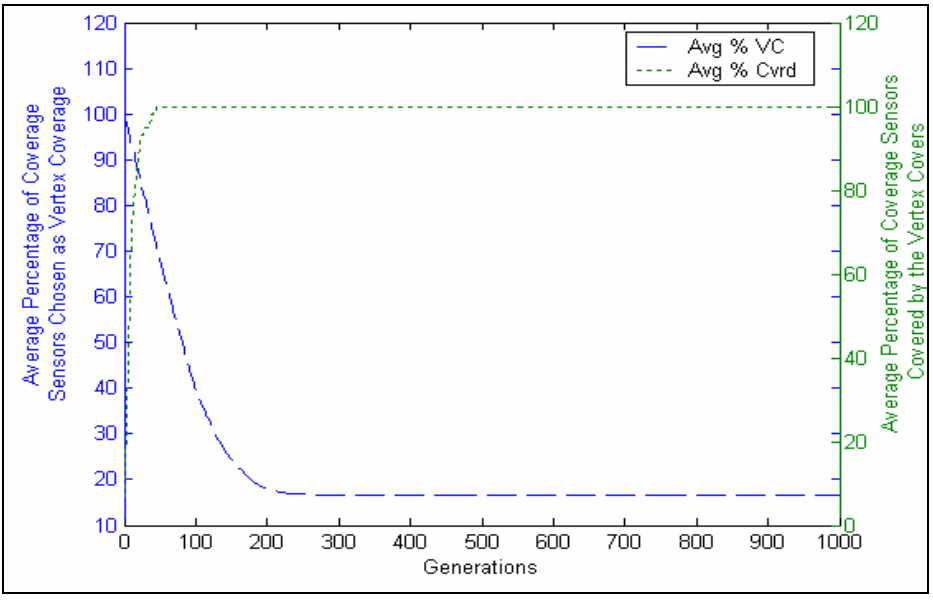


Fig. 5. Relation between the average percentage of coverage sensors chosen as vertex covers, the average percentage of coverage sensors covered by the vertex covers and the number of generations

Figure 5 illustrates two curves that tracked the behavior of the whole population with respect to the average percentage of coverage sensors chosen as vertex covers, and the average percentage of coverage sensors covered by the vertex covers. As the number of generations increased, the average number of coverage nodes chosen as

vertex covers decreased from 100% of the coverage sensors to around 16% of the coverage sensors. In addition, the second curve illustrates that our methodology managed to cover all the coverage sensors by the set of the vertex covers (100% coverage ratio) as the number of generations increased. The average ratio of the number of vertex covers out of the total number of coverage sensors was around 37% and those vertex covers covered 100% of the original 45.

In Figure 6, the curve tracked the behavior of the whole population with respect to the average number of vertex covers in the range of other vertex covers. Since we began with a solution that considers every coverage sensor as a vertex cover, and given the communication ranges of those vertex covers, a large percentage of the vertex cover sensors are covered by the communication ranges of other vertex covers. However, as the number of generations increased, this ratio dropped to almost 0%. Thus, our methodology was able to optimize the solution to a minimum set of vertex covers that cover the whole coverage sensors and at the same time they do not cover each other by their communication ranges (i.e., minimized their intersections.)

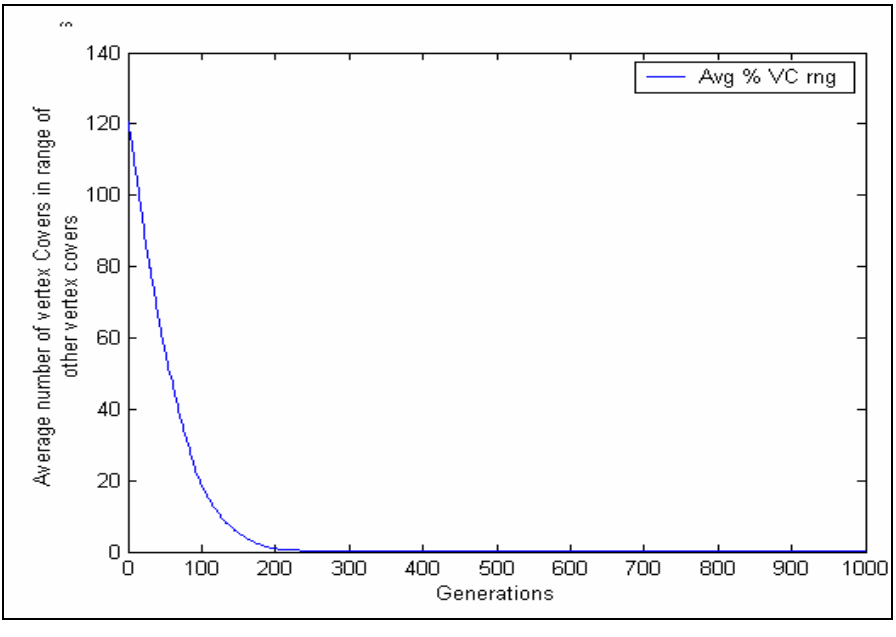


Fig. 6. Relation between the average number of vertex covers in range of other vertex covers and the number of generations

Figure 7 illustrates three curves that track the best, the average, and the worst percentage of sensors chosen as vertex covers as the number of generations increased. All the three curves show a consistent behavior of our methodology in choosing the optimum number of sensors as vertex covers, and that the number of VCs decreases with the increasing number of generations. We concluded from these early experiments that our methodology managed to produce a near optimal number of vertex cover sensors that cover all the coverage sensors, and this was accomplished in less than 14 seconds.

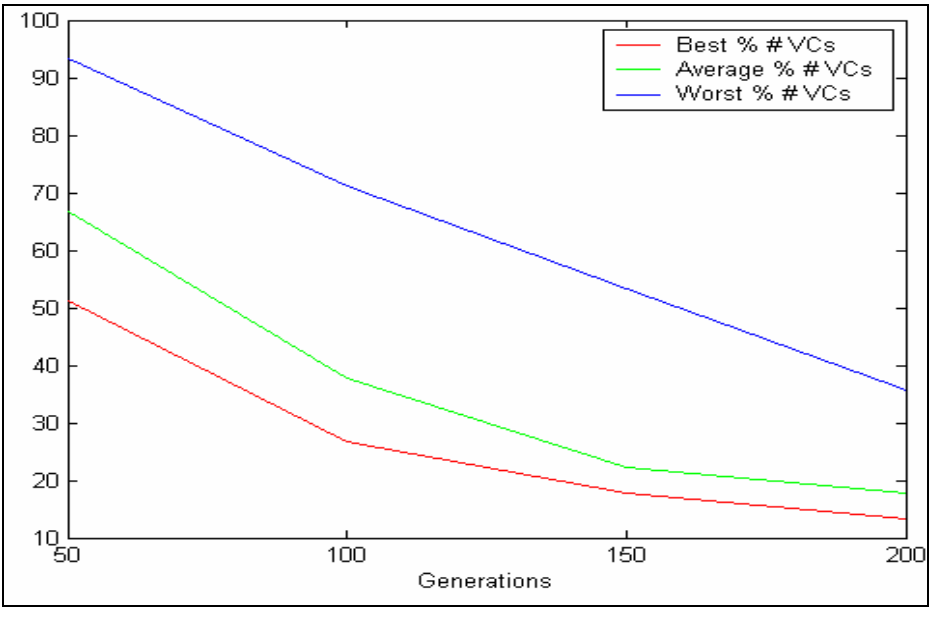


Fig. 7. Relation between the best, average and worst number of vertex-cover sensors and the number of generations

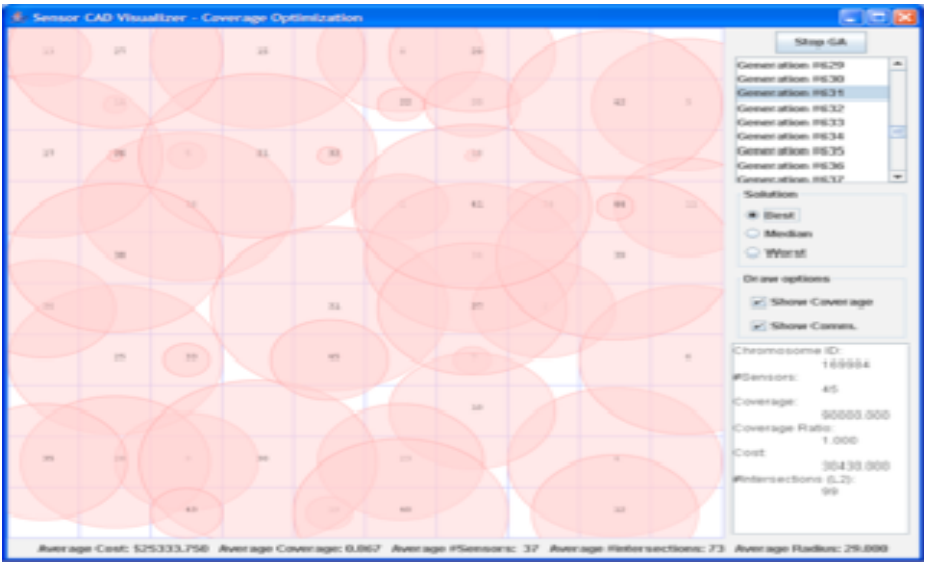


Fig. 8. A snapshot of the Sensor CAD Visualizer, which shows that after 631 generations of applying the coverage GA found 45 sensors as the best coverage at a cost of $30,430 while having a coverage ratio is equal to 99%

Fig. 9. A snapshot of the Sensor CAD Visualizer, which shows that after 200 generations of applying the communication GA found 8 sensors as the best vertex-cover sensors that cover all coverage sensors

Finally, in Figures 8 and 9 we show snapshots of our Sensor CAD Visualizer. It provides a graphical user interface that lets the user control the genetic algorithm, visualizes the solutions graphically, shows how it would look like in real life, and lets you go through different generations that were produced by the GA. Figure 8 shows the result of applying our evolutionary coverage GA, while Figure 9 shows the result of applying our evolutionary communication GA.

6 Conclusion and Future Directions

We have extended our previously proposed model for the coverage problem in the wireless sensor networks by introducing a new model for the communication problem. As with the coverage model, our communication modeling has reduced the solution space into a discrete optimization problem so that it can achieve the maximum communication possible with the least number of the coverage sensors (i.e., vertex covers) and at the same time guarantees that all the coverage sensors are covered by the vertex covers. Our early experiments with our new evolutionary model demonstrate very promising results. We will continue to improve our methodology by trying to solve both the coverage and communication problems simultaneously, and hence try to increase the transmission power during the coverage problem while reducing the energy utilization and reducing the over all cost of constructing the sensor network. Furthermore, we want to run our evolutionary communication GA using both the mutation and crossover operations. We want to study the effect of using both operations on the number of vertex cover sensors and the time required to get such optimal solution.

References

1. Wong, J., Neve, M., Sowerby, K.: Optimisation strategy for wireless communications system planning using linear programming. Electronic Letters 37, 1086–1087 (2001)
2. Yangyang, Z., Chunlin, J., Ping, Y., Manlin, L., Chaojin, W., Guangxing, W.: Particle Swarm Optimization for Base Station Placement in mobile communication. In: IEEE International Conference on Networking, Sensing and Control, Taipei, Taiwan, vol. 1, pp. 428–432 (2004)
3. Hurley, S., Kapp-Rawsley, R.: Towards automatic cell planning. In: The 11th IEEE International Conference on Personal, Indoor and Mobile Radio Communications, London, pp. 1583–1588 (2000)
4. Ishizuka, M., Aida, M.: Performance Study of Node Placement in Sensor Networks. In: IEEE International Conference on Distributed Computing Systems 2004 Workshops Assurance in Distributed Systems and Networks, Tokyo, Japan, pp. 598–603 (2004)
5. Han, J.K., Park, B.S., Choi, Y.S., Park, H.K.: Genetic approach with a new representation for base station placement in mobile communications. In: The proceedings of the 54th IEEE Conference on Vehicular Technology, vol. 4, pp. 2703–2707 (2001)
6. Butterworth, K.S., Sowerby, K.W., Williamson, A.G.: Base station placement for in-building mobile communication systems to yield high capacity and efficiency. IEEE Transactions on Communications 48, 658–669 (2000)
7. Takanashi, H., Rappaport, S.S.: Dynamic base station selection for personal communication systems with distributed control schemes. In: The proceedings of the 47th IEEE Conference on Vehicular Technology, vol. 3, pp. 1787–1791 (1997)
8. Patel, M., Chandrasekaran, R., Venkatesan, S.: Energy efficient sensor, relay and base station placements for coverage, connectivity and routing. In: The 24th IEEE International Performance, Computing, and Communications Conference (IPCCC), pp. 581–586 (2005)
9. Rodrigues, R.C., Mateus, G.R., Loureiro, A.: Optimal base station placement and fixed channel assignment applied to wireless local area network projects. In: IEEE International Conference on Networks, pp. 186–192 (1999)
10. Wright, M.H.: Optimization Methods for Base Station Placement. In: Wireless Applications the proceedings of Vehicular Technology Conference (1998)
11. Park, B.-S., Park, H.-K., Yook, J.-G.: The Determination of Base Station Placement and Transmit Power in an Inhomogeneous Traffic Distribution for Radio Network Planning. In: The proceedings of the 56th IEEE Conference on Vehicular Technology, vol. 4, pp. 2051–2055 (2002)
12. Yang, S.-T., Ephremides, A.: Optimal Network Design: the Base Station Placement Problem. In: The proceedings of the 36th IEEE Conference on Decision and Control, San Diego, CA, USA, vol. 3, pp. 2381–2386 (1997)
13. Huang, X., Behr, U., Wiesbeck, W.: A New Approach to Automatic Base Station Placement in Mobile Network. In: The International Zurich Seminar on Broadband Communications, pp. 301–306 (2000)
14. Lindström, M.: Base Station Placement in Asymmetric TDD Mode Systems in a Manhattan Environment. In: The proceedings of the 59th IEEE Conference on Vehicular Technology, vol. 4, pp. 1968–1972 (2004)
15. Huang, X., Behr, U., Wiesbeck, W.: Automatic base station placement and dimensioning for mobile network planning. In: The proceedings of IEEE 52nd Vehicular Technology Conference (VTC2000-Fall), Boston, Massachusetts, USA, vol. 4, pp. 1544–1549 (2000)

16. Fruhwirth, T., Brisset, P.: Placing base stations in wireless indoor communication networks. IEEE Intelligent Systems 15, 49–53 (2000)
17. Habib, S.: Modeling the Coverage Problem in Wireless Sensor Networks as Floorplanning and Placement Problems. In: The proceedings of the 6th IASTED International Multi-Conference on Wireless and Optical Communications (Wireless Sensor Networks), Banff, AB, Canada (2006)
18. Dasgupta, K., Kukreja, M., Kalpakis, K.: Topology-Aware Placement and Role Assignment for Energy-Efficient Information Gathering in Sensor Networks. In: The proceedings of Eighth IEEE International Symposium on Computers and Communication, Turkey, vol. 1, pp. 341–348 (2003)
19. Quintao, F.P., Mateus, G.R., Nakamura, F.G.: An Evolutive Approach for the Coverage Problem in Wireless Sensor Network. In: The Proceeding of 24th Brazilian Computer Society Congress (2004)
20. Quintao, F.P., Nakamura, F.G., Mateus, G.R.: Evolutionary Algorithm for the Dynamic Coverage Problem Applied to Wireless Sensor Networks Design. In: The proceedings of IEEE Congress on Evolutionary Computations, Edinburgh, UK (2005)
21. Crossbow Technology Inc.

A Selective Push Algorithm for Cooperative Cache Consistency Maintenance over MANETs

Yu Huang[1,2], Beihong Jin[3], Jiannong Cao[4], Guangzhong Sun[5], and Yulin Feng[3]

[1] State Key Laboratory for Novel Software Technology (Nanjing Univ.), Nanjing, China
[2] Dept. of Computer Science and Technology, Nanjing Univ., Nanjing, China
yuhuang@ics.nju.edu.cn
[3] Technology Center of Software Engineering, Institute of Software
Chinese Academy of Sciences, Beijing, China
{jbh,feng}@otcaix.iscas.ac.cn
[4] Internet and Mobile Computing Lab, Dept. Of Computing
Hong Kong Polytechnic Univ, Kowloon, Hong Kong
csjcao@comp.polyu.edu.hk
[5] Dept. of Computer Science and Technology
Univ. of Science and Technology of China, Hefei, China
gzsun@ustc.edu.cn

Abstract. Cooperative caching is an important technique to support efficient data dissemination and sharing in Mobile Ad hoc Networks (MANETs). In order to ensure valid data access, the cache consistency must be maintained properly. Many existing cache consistency maintenance algorithms are stateless, in which the data source node is unaware of the cache status at each caching node. Even though stateless algorithms do not pay the cost for cache status maintenance, they mainly rely on broadcast mechanisms to propagate the data updates, thus lacking cost-effectiveness and scalability. Besides stateless algorithms, stateful algorithms can significantly reduce the consistency maintenance cost by maintaining status of the cached data and selectively propagating the data updates. Stateful algorithms are more effective in MANETs, mainly due to the bandwidth-constrained, unstable and multi-hop wireless communication. In this paper, we propose a stateful cache consistency maintenance algorithm called *Greedy Walk-based Selective Push* (GWSP). In GWSP, the data source node maintains the Time-to-Refresh value and the cache query rate associated with each cache copy. Thus, the data source node propagates the source data update only to caching nodes which are in great need of the update. After recipients of the source data update have been decided, GWSP employs a greedy but efficient strategy to propagate the update among the selected caching nodes. Extensive simulations are conducted to evaluate the performance of GWSP. The evaluation results show that, compared with the widely used *Pull with Dynamic TTR* algorithm, GWSP can save up to 41% traffic overhead and reduce the query latency by up to 85% for cache consistency maintenance in cooperative caching over MANETs.

Keywords: Mobile Ad hoc Networks, Cooperative Caching, Cache Consistency, Stateful, Cache Status Maintenance, Selective Push.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 650–660, 2007.

1 Introduction

Mobile Ad hoc Networks (MANETs) have received considerable attention due to the potential applications in pervasive Internet access, outdoor assemblies and disaster salvage [1, 2, 3, 20]. Ad hoc networks feature in their quick deployment and easy reconfiguration, which makes them ideal in situations where installing an infrastructure is too expensive or too vulnerable. The primary goal of deploying MANETs is to support pervasive and efficient data dissemination and sharing [2, 3, 20]. For example, in a MANET shown in Fig. 1, the mobile hosts close to an access point can directly access the Internet, and hence can serve as gateway nodes. Other mobile hosts access Internet resources via the gateway nodes through multi-hop wireless connections. In another example, several commanding officers and a group of soldiers form a MANET in fulfilling a mission of disaster salvage. The commanding officers can access useful information, such as the geographic information, via the satellites. The useful information and commands from the officers can then be efficiently disseminated via the MANET.

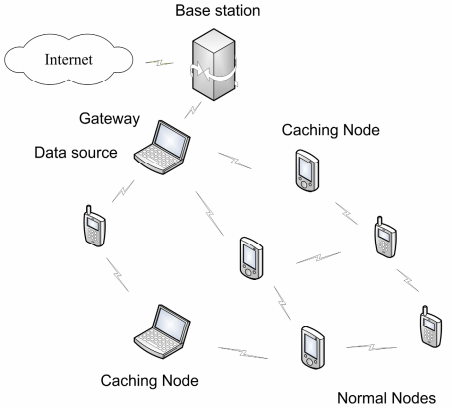


Fig. 1. A MANET with one data source node and multiple caching nodes

However, the limited communication resources (e.g., bandwidth and battery power) and users' mobility make pervasive data dissemination and sharing a challenging task in MANETs. One effective and widely-used method to improve the performance of data access in MANETs is to use *Cooperative Caching*, i.e., to cache frequently accessed data objects at the *data source node* (gateway node) and a group of *caching nodes* [3, 4, 5, 13, 20]. Thus, other mobile users can access the cached data objects nearby, with reduced traffic overhead and query latency.

In order to ensure valid data access, the cache consistency [2, 13, 20, 21], i.e., consistency among the source data owned by the data source node and the cache copies held by the caching nodes, must be maintained properly. Cache consistency maintenance algorithms can be divided into two main categories: *stateful* [17] and *stateless* [15, 16], based on whether the cache status is maintained on the data source node [21]. Existing consistency maintenance algorithms for MANET (e.g., [2, 13,

20]) are mainly stateless, in which the data source node is unaware of status of the cached data. The data source node mainly relies on broadcast mechanisms to propagate the data updates, which inevitably introduces much redundant data update propagation.

Besides stateless algorithms, there also exist stateful consistency maintenance algorithms in the literature. In stateful algorithms, the data source node maintains the cache status (e.g. the Time-to-Refresh value [9]) of each cache copy. Based on the cache status maintained, the data source node can hence selectively propagate the data updates to the caching nodes which are in great need of the updates, i.e., caching nodes whose cache copy will soon expire. Compared with stateless algorithms, stateful ones significantly reduce the consistency maintenance cost by selectively propagate the data updates based on the cache status. We argue that stateful algorithms are more suitable for MANETs, since the power consumption of data transmission (for data update propagation) is much more than that of local computation (for cache status maintenance) on wireless terminals [10]. So far, to the best of our knowledge, no stateful cache consistency maintenance algorithm has been proposed for MANETs.

In this paper, we propose a stateful cache consistency maintenance algorithm called *Greedy Walk-based Selective Push* (GWSP). In GWSP, the data source node maintains the *Time to Refresh* value and the cache query rate associated with each cache copy. Based on the cache status, the data source node can hence make online decisions on which cache copies are in need of the data updates upon each update. When recipients of the data update have been decided, GWSP employs the *greedy walk* mechanism to propagate the data update among the selected caching nodes. Cooperation among the data source node and caching nodes amortizes the cost for data update propagation.

Extensive simulations are conducted to evaluate the performance of GWSP. The evaluation results show that, compared with the stateless *Pull with Dynamic TTR* algorithm, GWSP can save up to 41% traffic overhead and reduce the query latency by up to 85%.

The rest of this paper is organized as follows. Section 2 presents design and analysis of the GWSP algorithm. In Section 3, we present the experimental evaluation. Finally, Section 4 concludes the paper with a summary and the future work.

2 Greedy Walk-Based Selective Push

The *Greedy Walk-based Selective Push* (GWSP) algorithm adopts a widely accepted system model [2, 13, 20], in which each data object is associated with a single node that can update the *source data*. This node is referred to as the *data source node*. Each data object can be cached by a collection of nodes called the *caching nodes*. The data copies held by the caching nodes are called the *cache copies*[1]. There are two basic mechanisms for cache consistency maintenance: *push* and *pull*. Using push, the data source node informs the caching nodes of data updates. Using pull, the caching node sends a request to the data source node to check the update.

[1] In this paper, we do not consider the issue of caching node membership. This issue should be addressed by the cooperative caching mechanism, based on which we further focus on the issue of cache consistency maintenance.

In designing GWSP, we assume that the source data updates and the cache queries follow the *Poisson Process* of rate λ_u and λ_q respectively [22]. We also assume that the routing protocol employed in the network layer provides the hop count between each pair of nodes, and the hop count of data transmission is used to measure the consistency maintenance cost [3].

2.1 Providing Delta Consistency by Pull with TTR

GWSP provides Delta Consistency based on the *Pull with TTR algorithm.* In *Pull with TTR*, each cache copy is associated with a timeout value *Time to Refresh* (TTR). The initial value of TTR is set to δ. When TTR is valid (TTR > 0), the caching node can directly serve cache queries. When TTR expires, the caching node first pulls the data source node to update the cache copy and to renew TTR to δ. Then the caching node can directly serve cache queries. By associating a TTR value with each cache copy, GWSP guarantees that deviation between the source data and the cache copy will not be over δ, thus ensuring Delta Consistency [2, 21].

Although the *Pull with TTR* algorithm guarantees Delta Consistency, it is not cost-effective, mainly due to the round–trip consistency maintenance cost imposed by the pull mechanism. Therefore, GWSP employs a stateful selective push mechanism to save consistency maintenance cost.

2.2 Selective Push

Using push, the data source node informs the caching nodes of data updates, which only imposes one-way consistency maintenance cost (traffic overhead, query latency etc.). However, if the data source node is unaware of the cache status of each cache copy, there exist redundant data update propagations in the following two cases:

- After the cache copy and the associated TTR are renewed via push, there comes no cache query before the TTR expires;
- Before the TTR expires, there are multiple data updates (only the last data update should make the data source node push the caching nodes);

Thus, the following design principles should be followed in designing GWSP: use the push mechanism to save consistency maintenance cost, but

- Push only if the cache copy is expected to serve queries;
- Push if there are probably no other data updates before TTR expires;

The detailed design of the GWSP algorithm is. Upon the source data update at t_u, we consider the cache status on one caching node, as shown in Fig. 2. The TTR of this cache copy was renewed (via push or pull) at time t_0. Thus, the remaining TTR of this caching node at time t_u is $TTR_{remain} = t_0+\delta-t_u$, and the TTR renewed is $TTR_{renew} = t_u - t_0$. According to the design principles, the data source node pushes one caching node only when the following two requirements are both satisfied:

1. The probability that there is at least one query in period $(t_0+\delta, t_u+\delta)$ is greater than threshold value τ_q;
2. The probability that there is at least one update in period $(t_u, t_0+\delta)$ is less than threshold value τ_u;

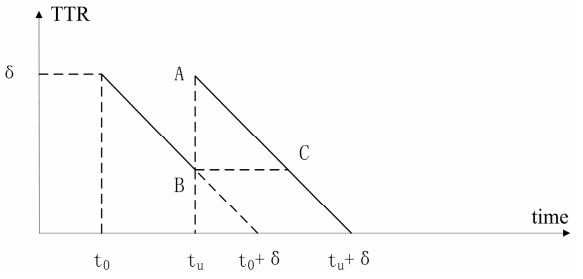


Fig. 2. Cache status on one caching node

For a *Poisson Process* with rate λ, the probability that there are k events in a period of length t is: $P\{t, k\} = (\lambda t)^k e^{-\lambda t} / k!$. Thus, the probability that there is at least one query in period $(t_0+\delta, t_u+\delta)$ is:

$$1 - P\{(t_u+\delta)-(t_0+\delta), 0\} = 1 - P\{t_u - t_0, 0\} = 1 - e^{-\lambda_q (t_u - t_0)}$$

Thus, for *Requirement (1)*, we have that: $1 - e^{-\lambda_q (t_u - t_0)} \geq \tau_q$, which is equivalent to:

$$TTR_{renew} = t_u - t_0 \geq -\log(1 - \tau_q) / \lambda_q \quad (1)$$

Note that each caching node maintains the query rate λ_q for the cache copy. When the caching node pulls the data source node, it will piggyback the value λ_q in the pull message. Thus, the data source node obtains the query rate λ_q of each cache copy. The data source node also maintains the TTR value of each cache copy.

The probability that there is at least one update in period $(t_u, t_0+\delta)$ is:

$$1-P\{t_0+\delta-t_u, 0\} = 1 - e^{-\lambda_u (t_0+\delta-t_u)}$$

Thus, for *Requirement (2)*, we have that $1 - e^{-\lambda_u (t_0+\delta-t_u)} \leq \tau_u$, which is equivalent to:

$$TTR_{remain} = t_0 + \delta - t_u \leq -\log(1 - \tau_u) / \lambda_u \quad (2)$$

Based on inequality (1) and (2), the data source node can make on-line decisions on which caching nodes should receive the PUSH message upon each data update.

2.3 Greedy Walk-Based Data Update Propagation

After the data source node has decided the *Push Set*, i.e., the caching nodes which should receive the data update, it needs to decide how to propagate the data updates among the selected caching nodes. GWSP employs a greedy but efficient strategy *Greedy Walk* to disseminate the PUSH message (which contains the source data update and IDs of the push set nodes) to all caching nodes in the push set. The data source node sends the PUSH message to the nearest caching node in the push set via unicast. The caching node which receives the PUSH message deletes itself from the push set. It acknowledges the PUSH message with a PUSH_ACK message, and then goes on relaying the push message to the nearest caching node in the push set (The path length between two nodes are provided by the routing protocol, according to our assumption). This process is repeated until the push set is empty. The last caching

node will send a PUSH message to the data source node, which initiates the PUSH message propagation process.

If the sender of some PUSH message does not receive the corresponding PUSH_ACK (since either the PUSH message or the PUSH_ACK message is lost), it will wait for at most τ_{ACK} seconds. Then, it will send the PUSH message to the data source node. We require that PUSH messages should be acknowledged, in order to deal with the message loss, which often occurs in MANETs.

When the data source node receives the PUSH_ACK message, it obtains the time span of the PUSH message propagation process. It also knows the caching nodes which have received the PUSH message. The data source node then updates the TTR values of the cache copies as follows. Let t_{push} denote the time span of the PUSH message propagation process. The data source node sets the TTR values of all cache copies which receives the PUSH message to δ - t_{push}. Hence, the data source node can maintain the TTR values of each cache copy more accurately, by taking into account the time delay imposed by the PUSH message propagation process. The pseudo-code of the GWSP algorithm is listed below.

Algorithm 1. GWSP on the data source node

Upon each source data update
// deciding the Push Set
(1) Put all caching nodes satisfying inequality (1) and (2) into the push set;
// propagating the source data updates by Greedy Walk
(2) Send the source data update and the push set information in a PUSH message to the nearest caching node in the push set;

Upon receiving PUSH message
(3) Decide all the caching nodes which have received the PUSH message and the time span of the push process t_{push};
(4) Update the TTR of the caching nodes which received the PUSH message by letting $TTR = \delta - t_{push}$;

Algorithm 2. GWSP on a caching node

Upon receiving a cache query
(1) If (TTR = 0) pull the data source node to update the cache copy and TTR;
(2) Serve the cache query;

Upon receiving a PUSH message
(3) Reply the PUSH message with PUSH_ACK;
(4) Delete itself from the push set;
(5) If(size of the push set > 0) relay the PUSH message to the nearest caching node in the push set;
(6) Else send the PUSH message to the data source node;

Upon receiving a PUSH_ACK message
(7) If(no PUSH_ACK is received within time τ_{ACK})
send the PUSH message to the data source node;

2.4 Analysis of the Greedy Walk Strategy

Finding the optimal path for propagating the PUSH message can be easily proven to be an instance of the *Traveling Salesman Problem* (TSP). TSP is NP-Complete and there is no even approximate algorithm with constant approximation ratio for it [6]. Thus the simple greedy walk strategy is adopted in GWSP, which is inspired by the *Anycast* mechanism. Moreover, the greedy walk strategy is also easy to implement in the distributed and asynchronous MANET environment. We also find that GWSP is load-balanced. The data source node and the caching nodes cooperate to propagate the data updates and hence amortize the overhead. The load balance of GWSP makes it suitable for the resource-constrained mobile terminals.

3 Experimental Evaluation

3.1 Experimental Methodology and Configurations

We have conducted simulations to evaluate the performance of GWSP. In the experiments, we first compare *greedy walk* with the optimal solution. We adopt a brutal force search strategy to obtain the optimal solution. The network we consider should contain a suitable number of nodes, so that computation of the optimal solution does not become exceedingly cumbersome. Then we finely tune the number of caching nodes and the cache query rate, which have great impact on GWSP. We study the cost-effectiveness of GWSP based on extensive performance comparison with the *Pull with Dynamic TTR* (named as DynTTR in short) algorithm [11, 12, 13, 20]. The following performance metrics are used in the evaluation:

- Traffic overhead: number of hops counted for consistency maintenance message propagation;
- Query latency: latency imposed by consistency maintenance. We obtain the query latency based on the number of hops the message needs to be relayed in the MANET;

Detailed experimental configurations are listed in Table 1.

Table 1. Experiment configurations

Network area	200×200 m^2
Size of network	80 m
Transmission range	15 m
Mobility model	Random way point [23]
Average speed	0.5 m/s
Maximum portion of crashed nodes	10%
Pattern of date updates and queries	*Poisson process*
Initial value of TTR (δ)	10 s
τ_u	50%
τ_q	50%
τ_{ACK}	2s

The default configurations of the number of caching nodes and average query interval are as follows:

- number of caching nodes: 20 ;
- average query interval: 10 s ;

We will study the impact of varying these parameters in the following experiments.

3.2 Evaluating the Greedy Walk Strategy

In this experiment, we compare greedy walk with the optimal solution. From the evaluation results (Fig 3), we find that, even compared with the optimal solution, the greedy walk strategy is quite cost-effective, especially when there are a suitable number of caching nodes. As the number of caching nodes increase, the performance of greedy walk gradually degrades. Given the computational complexity of finding the optimal solution, greedy walk is quite cost-effective.

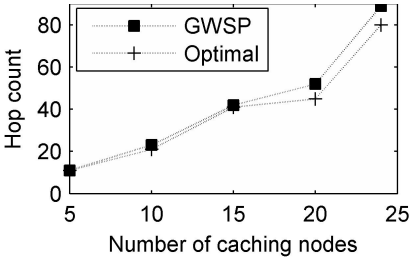


Fig. 3. Cost-effectiveness of greedy walk

3.3 Effects of Tuning the Number of Caching Nodes

In this experiment, we increase the number of caching node from 10 to 20, 30 and 40. We find that GWSP is comparatively more cost-effective when there are a suitable number of caching nodes. When there are 20 and 30 caching nodes, the traffic overhead saved by GWSP is 26% and 21% respectively (Fig. 4). Here, the percentage of traffic overhead saved refers to the ratio of the saved traffic overhead to that imposed by DynTTR. However, when there are 10 and 40 caching nodes, the traffic overhead saved decreases to 15% and 9% respectively (Fig. 4). It is mainly because, when there are only a few caching nodes, little traffic overhead can be saved by selective push in GWSP. On the other hand, when there are many caching nodes, the PUSH & ACK process in GWSP also imposes great traffic overhead.

We also find that the query latency imposed by GWSP is relatively stable, while the latency imposed by DynTTR increases as the number of caching nodes increases (Fig. 5). The query latency saved in GWSP can go up to 85% when there are 30 caching nodes. The selective push mechanism based on the cache status accounts for the better performance of GWSP in terms of query latency.

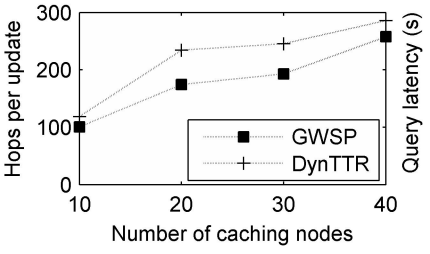


Fig. 4. Traffic overhead

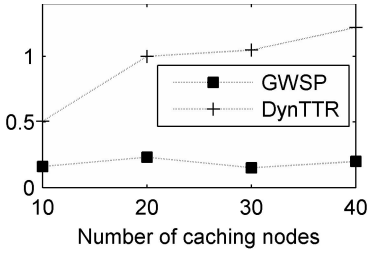


Fig. 5. Query latency

3.4 Effects of Tuning the Cache Query Rate

In this experiment, we tune the cache query rate. We find that, when the cache query rate decreases, the traffic overhead imposed by GWSP gradually decreases, while the traffic overhead imposed by DynTTR decreases much more quickly (Fig. 6). The traffic overhead saved by GWSP decreases from 41% (query interval = 5s) to 19% (query interval = 20s). The query latency imposed by DynTTR decreases similarly. In GWSP, the query latency even increases when there are less frequent queries (query interval = 15), as shown in Fig. 7. The query latency saved in GWSP decreases from 72% (query interval = 5s) to 19% (query interval = 20s). It is mainly because less and less caching nodes are selected in the push set when faced with less and less frequent queries. Thus, the GWSP algorithm gradually approaches the Pull Each Read [14] algorithm, which suffers from round-traffic query latency.

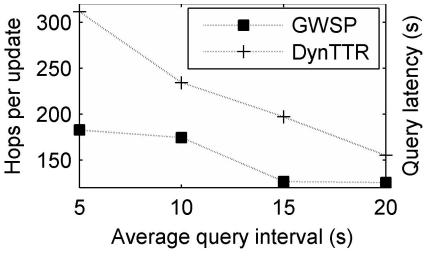


Fig. 6. Traffic overhead

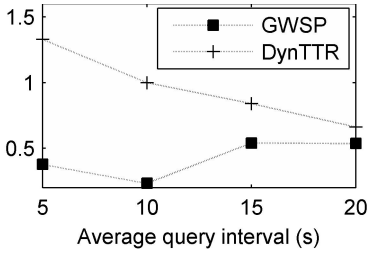


Fig. 7. Query latency

4 Conclusion and Future Work

In this paper, we study how to cost-effectively maintain cache consistency in cooperative caching in MANETs. We focus on stateful cache consistency maintenance algorithms since they can significantly reduce the consistency maintenance cost utilizing the cache status information. We propose the stateful *Greedy walk-based Selective Push* algorithm. In GWSP, the data source node maintains the TTR value and the cache query rate associated with each cache copy. Upon each source data update, the data source node efficiently decides the caching nodes which are in great need of the data updates. GWSP employs the greedy walk

strategy to propagate the data updates among the selected caching nodes. Extensive experiments are conducted to evaluate the performance of GWSP. The evaluation results show that the greedy walk strategy can cost-effectively propagate the data updates in a MANET environment. The GWSP algorithm can save up to 41% traffic overhead and 85% query latency for cache consistency maintenance, compared with the *Pull with Dynamic TTR* algorithm. In our future work, we plan to study how to support different consistency models, besides Delta Consistency, by stateful consistency maintenance algorithms in cooperative caching in MANETs.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant No. 60673123, the National High-Tech Research and Development Program of China under Grant No. 2006AA01Z231, the Hong Kong RGC CERG grant PolyU 5105/05E and Hong Kong PolyU ICRG grant G-YF61.

References

1. Corson, M., Macker, J., Cirincione, G.: Internet-based Mobile Ad Hoc Networksing. In: IEEE Internet Computing, pp. 63–70 (July-August 1999)
2. Cao, J., Zhang, Y., Xie, L., Cao, G.: Consistency of Cooperative Caching in Mobile Peer-to-Peer Systems Over MANET. Intl. J. of Parallel, Emergent, and Distributed Systems 21(3) (2006)
3. Yin, L., Cao, G.: Supporting Cooperative Caching in Ad Hoc Networks. IEEE Transactions on Mobile Computing 5(1) (2006)
4. Lau, W., Kumar, M., Venkatesh, S.: A Cooperative Cache Architecture in Supporting Caching Multimedia Objects in MANETs. In: Proc. Fifth Intl Workshop Wireless Mobile Multimedia (2002)
5. Sailhan, F., Issarny, V.: Cooperative Caching in Ad hoc Networks. In: Chen, M.-S., Chrysanthis, P.K., Sloman, M., Zaslavsky, A. (eds.) MDM 2003. LNCS, vol. 2574, Springer, Heidelberg (2003)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Approximation Algorithms, Introduction to Algorithms, ch. 35. MIT Press, Cambridge (2002)
7. Wang, Z., Das, S.K., Che, H., Kumar, M.: A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments. IEEE Tran. on Parallel and Distributed Systems 15(11) (2004)
8. Tan, K., Cai, J., Ooi, B.C.: An Evaluation of Cache Invalidation Strategies in Wireless Environments. IEEE Tran. on Parallel and Distributed Systems 12(8) (2001)
9. Urgaonkar, B., Ninan, A., Raunak, M., Shenoy, P., Ramamritham, K.: Maintaining Mutual Consistency for Cached Web Objects. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, AZ (April 2001)
10. Ferrigno, L., Marano, S., Paciello, V., Pietrosanto, A.: Balancing Computational and Transmission Power Consumption in Wireless Image Sensor Networks. In: IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems(VECIMS) (2005)

11. Urgaonkar, B., Ninan, A., Raunak, M., Shenoy, P., Ramamritham, K.: Maintaining Mutual Consistency for Cached Web Objects. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, AZ (April 2001)
12. Lan, J., Liu, X., Shenoy, P., Ramamritham, K.: Consistency Maintenance in Peer-to-Peer File Sharing Networks. In: The 3rd IEEE Workshop on Internet Applications (2003)
13. Huang, Y., Cao, J., Jin, B.: A Predictive Approach to Achieving Consistency in Cooperative Caching in MANET. In: Proc. of the 1st Intl. Conf. on Scalable Information Systems, P2PIM workshop section, ACM Press, New York (2006)
14. Howard, J., Kazar, M., et al.: Scale and Performance in a Distributed File System. ACM Trans. on Computer Systems 6(1) (1988)
15. Barbara, D., Imielinksi, T.: Sleeper and Workaholics: Caching Strategy in Mobile Environments. In: Proc. ACM SIGMOD Conf. Management of Data, pp. 1–12 (1994)
16. Cao, G.: A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments. In: Proc. ACM Int'l Conf. Computing and Networking (Mobicom), pp. 200–209 (August 2001)
17. Kahol, A., Khurana, S., Gupta, S.K.S., Srimani, P.K.: A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment. IEEE Trans. Parallel and Distributed Systems 12(7), 686–700 (2001)
18. Yang, B., Hurson, A.R., Jiao, Y.: On the Content Predictability of Cooperative Image Caching in Ad hoc Networks. In: Proc. Of the 7th Intl. Conf. on Mobile Data Management (MDM) (2006)
19. Liu, B., Lee, W., Lee, D.L.: Distributed Caching of Multi-dimensional Data in Mobile Environments. In: Proc. Of the 6th Intl. Conf. on Mobile Data Management (MDM) (2005)
20. Huang, Y., Cao, J., Wang, Z., Jin, B., Feng, Y.: Achieving Flexible Cache Consistency for Pervasive Internet Access. In: proc. of the 5th Annual IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom), New York, U.S, pp. 239–250 (2007)
21. Cao, J., Zhang, Y., Xie, L., Cao, G.: Data Consistency for Cooperative Caching in Mobile Environments. IEEE Computer, 60–67 (2007)
22. Hou, Y.T., Pan, J., Li, B., Panwar, S.S.: On Expiration-based Hierarchical Caching Systems. IEEE J. on Selected Areas in Comm. 22(1) (2004)
23. Camp, T., Boleng, J., Davies, V.: A Survey of Mobility Models for Ad Hoc Network Research. Wireless Communications & Mobile Computing (WCMC) 2(5) (2002)

A Constrained Multipath Routing Protocol for Wireless Sensor Networks

Peter K.K. Loh and Y.K. Tan

Nanyag Technological University, School of Computer Engineering,
Nanyang Avenue, Singapore 639798
askkloh@ntu.edu.sg

Abstract. A deployed wireless sensor network must be managed by an efficient and reliable routing protocol to overcome node degradation and failure, RF communication disruptions and limited energy reserves. RF transmissions are inherently broadcast. Existing research proposals for routing protocol designs typically use either *flooding* (routing same packet over all available paths) or *selective broadcasting* (routing packet over a specified path). Research has shown that routing protocols that use selective broadcasting exhibits better performance with lower communication overheads. Flooding-based protocols typically require lower control overheads and exhibit better fault tolerance. This paper presents two variations of a novel routing protocol, called SOS, which uses *constrained broadcasting* (packet routed over small subset of available paths) to adapt to network failures and disruptions. Simulations show that SOS compares favourably with existing selective broadcasting and flooding-based protocols in terms of performance, reliability and energy efficiency.

Keywords: routing protocol, sensor networks, selective broadcasting, constrained broadcasting, flooding.

1 Introduction

A smart environment needs information about its surroundings and its sensory system to operate reliably and efficiently. The importance of a wireless sensor network (WSN) as a sensory system is demonstrated by several recent initiatives [11]. Advances in radio and micro-electro mechanical systems (MEMs) technologies have made sensor nodes more cost effective. Hence, these nodes may be deployed in large numbers in various operating scenarios. Deployment is, however, often under unpredictable and/or inhospitable conditions that may prevent the sensory system from maintaining a stable network configuration to sustain long-term operations [2,8]. To tolerate these conditions, the WSN must be managed by a reliable routing protocol that will maintain its configuration and its operations in the presence of faults [6, 9, 10]. Existing routing protocols typically depend on either *flooding* [13, 15] or *selective broadcasting* [12, 14] to relay data packets. In flooding, each data packet is routed over all available RF links at a node. It is a brute-force approach to increase the chances of packet delivery. In selective broadcasting, only one neighbouring node receives the packet though other neighbours may hear it. The packet is progressively

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 661–670, 2007.

routed along a single path to the hub. It has been demonstrated that selective broadcasting protocols exhibit a performance edge over flooding protocols but the latter can exhibit better fault tolerance in small to moderate-sized networks with lower control overheads [3, 5, 12, 14]. In this paper, we propose a routing protocol that relays data with *constrained broadcasting* (small set of available paths for routing) while meeting the conflicting requirements of low communications and energy costs.

2 Related Work

In this section, we present a concise survey of four routing protocols: the Periodic, Event-Driven and Query-Based protocol (PEQ) [3-4], Gradient-Based Routing protocol (GBR) [14], Efficient and Reliable protocol (EAR) [12], and Gradient Broadcast protocol (GRAB) [13, 15]. These routing protocols are similar in the sense that they make use of path length and/or energy metrics for data dissemination. The first three protocols are based on selective broadcasting while the fourth uses network flooding and serves as a control reference for flooding-type protocols.

The PEQ (Periodic, Event-Driven and Query-Based) routing protocol uses an ACK (acknowledgement)-based scheme to identify node failures or weak signal conditions. To set up route distance information, each network node is initialised with a hop count to the hub. The hub is initialised with a hop count of 0. If a node receives hop information from more than one neighbour, only the lowest value is stored. Each node will only send data to the next node that is of a lower hop value. Hence, a single, shortest path for each node is created to deliver its data to the hub. To tolerate node failures and noisy links, PEQ uses two phases: (i) failure detection at receiver node and (ii) location of a new neighbour. A sender node forwards data to its neighbour and sets a timeout to wait for the ACK. The ACK message will only be received after the neighbour has forwarded the packet. Thus, the sender node on receiving its neighbour's ACK, knows that its neighbour is alive and that the packet has also been forwarded to the next node. However, if no ACK message is being received, the neighbour node is deemed to have failed and the sender node will select another neighbour as its new intermediate destination. To do this, the sender node broadcasts a search message to its neighbours. Neighbours reply with a message containing their hop count and identification. The sender will choose the lowest hop count and the first one to reply becomes the new neighbour node. The sender's routing table will be updated for relaying subsequent packets and the sender node sets its own hop count to be the neighbour node's hop count plus one. This also avoids looping in paths.

GBR distributes traffic evenly among all nodes and prevents non-uniform traffic overloading. The hub will broadcast an interest message that is flooded throughout the network. Each node upon receiving the interest message will record the number of hops taken by the interest message. Each node then knows the number of hops it needs to reach the hub. The *gradient* between two nodes is the difference in their hop counts. A hop gradient is set up from the nodes to the hub and all messages will flow in that direction towards the hub. A node will forward a message to a neighbour with the largest gradient (nearest) to the hub. When there are multiple neighbours with the same gradient to the hub, one is randomly chosen to spread traffic levels more uniformly. When less than 50% of a node's energy level remains, the node lowers its

gradient with respect to its neighbours. This reduces the chance of future packets being routed through it. Any changes in gradient will be progressively updated across the network to maintain routing consistency.

Routing decisions in EAR are based on four metrics: hop-count, distance traveled, energy level of next hop node and link transmission success history. A weighted combination of these metrics is used to relay packets. A "sliding-window" keeps track of the last *N* successful transmissions via a specified RF link to compute success history. An optimal route in EAR is not necessarily the shortest but represents the best combination of distance, energy requirements and link quality. Control packets are minimized by "piggy-backing" routing information onto MAC-layer protocol packets. EAR deals well with communication faults in WSNs with low to moderate traffic volumes. However, with a high-volume of network traffic, the more complex route management incurs an appreciable overhead affecting its performance.

GRAB is designed for reliability by routing redundant copies of messages in a mesh from a source node to the hub. A cost field is set up in the network and the value of a node in the field is the minimum cost to reach the hub from that node. The cost field has a value of 0 starting from the hub and the value at each node increases with the distance from the hub. Messages will flow through the cost field in the direction of decreasing cost, that is, towards the hub. When a node generates a packet, it initialises the header with its cost to the hub and assigns a credit value to that packet before broadcasting it. When neighbouring nodes receive the packet, only those nodes that have a lower cost will enter a decision process of whether to route or drop that packet. A message's credit is consumed at each node on the path to the hub. When a message has enough credit, the node will route the message or else the message will be dropped. By assigning an appropriate amount of credit to each message, duplicate copies of that message will travel in multiple paths from the source node to the hub.

3 Protocol Design Details

This section details the design of the SOS and SOS_d routing protocols. SOS_d is a double path variant of SOS that incorporates limited route redundancy. Both protocols support single or multiple hubs in a multi-hop WSN [16]. SOS and SOS_d can also be easily modified to support either query- or event-detection based WSNs.

3.1 Route Discovery Phase

This phase determines the WSN's configuration and builds appropriate routing tables at each node. A hop-level distribution from the hub is established in a similar manner to PEQ. A node will use its hop-level to identify the next possible neighbour(s) to relay the data. If the hop level received is lower than the receiving node's hop level by more than one, the receiving node updates its own hop level by adding one to the received hop level and retransmitting the 'hop' message with its own hop level to its neighbour(s). Otherwise, there will no updating and the 'hop' message will be discarded. Each node may store up to three possible routes to enhance routing reliability. To minimise the probability that an important node may not be discovered or a link that could have failed due to interference between two neighbouring nodes,

each node transmits the 'hop' message twice, with each message separated by a random delay to prevent collision of messages. The hub, however, only broadcasts the 'hop' message once to avoid energy wastage as in periodic flooding.

3.2 Data Relay Phase

Here, data is routed from each node to the hub and vice versa (Figure 1).

Each node maintains a routing table with up to three different routes to the hub. Each table entry is a 5-tuple: <*hub address*, *neighbour node ID*, *route status*, *times route failed*, *route usage counter*>. *Hub address* is the hub ID in the WSN, *neighbour node ID* is the ID of the intermediate node to which a data packet is to be sent en-route to the hub, *route status* indicates route availability, *times route failed* tracks the number of times the route has failed. The route will be removed from the routing table once a specified maximum number of retries is reached, *route usage counter* tracks the number of times a route has been used. A node will choose the route with the lowest usage. This helps distribute network traffic more uniformly. As a consequence, network lifetime can be increased with probability of node failure due to insufficient energy reduced. Another advantage is that message data packets forwarded along different routes will reduce the chance that a fault in one route causes loss of the entire message. Figure 1 shows node 7 receiving a data packet. In 'Route Table of node 7', although the route to node 8 has the smallest route usage of 1, the route status indicates it has failed. The next suitable route is to node 6, an active route with lowest route usage. After relaying the packet to node 6, the route usage value of this route will increase to 3, same as the route to node 5. The route to node 5 will be chosen the next time round as, with the same usage value, the first node in the routing table will be selected. Thus, the data packet will be routed from nodes 7-6-2-1 to the hub.

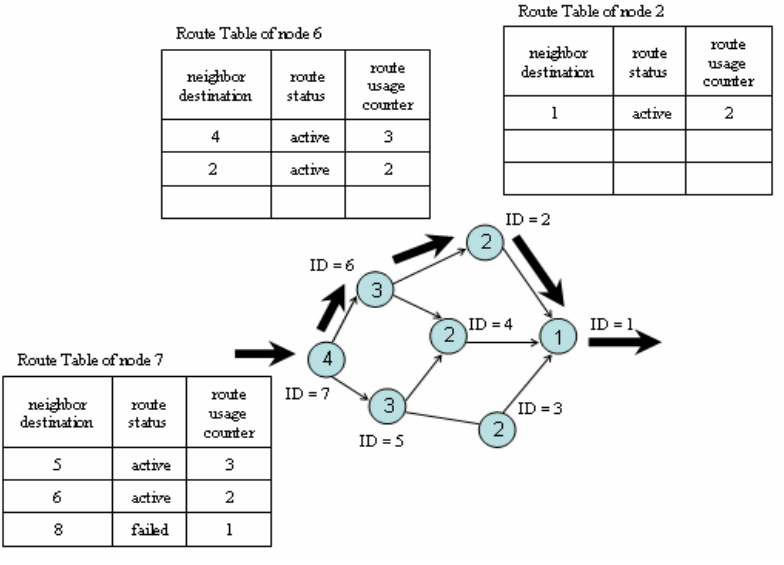


Fig. 1. Illustration of data packet relaying in SOS

3.3 Failure Recovery Phase

Node x relays a data packet to neighbour y by first sending RTS. After a specified number of retries, if no CTS is received from node y, node x updates its routing table by deleting node y's ID, decrements its 'number of routes to hub' value and broadcasts node y's ID in an error message to its neighbours. Neighbours receiving this error message remove the problematic node ID from their routing tables. If at least one route to the hub is left, it will be selected as the alternate route to forward the data packet. Presence of extra routes to the hub will increase the chance of delivery. Node x will then broadcast 'search' messages to neighbouring nodes to replace the problematic node. Node x could also be isolated if all its neighbours have failed. After the search, if no replacement is found, the process will terminate. Neighbours of higher hop level now know that node x has no more routes to the hub and will remove it's ID from their routing tables. A node will respond to a 'search' message when it has at least one route to the hub. Reply message with its hop level and ID is sent after a random delay to minimize collisions with other replies.

3.4 SOS Double Path - SOS_d

SOS_d exploits *constrained broadcasting* to relay data packets. It utilizes an additional message path to increase the probability of bypassing node/link failures and avoiding packet loss while relaying data packets. Additional paths to forward data packets will have a higher chance of ensuring delivery to the hub. However, the energy usage will be higher especially in larger networks. The limitation to a double instead of multiple paths is to avoid excessive redundancy, which can cause significant increases in packet collisions and network congestion, and incur unacceptable energy overheads. Only source nodes send packets to two different neighbours. However, if two data packets from a source reaches the same node *z*, node *z* also relays the packet to two different nodes due to routing control imposed by the route usage counter (Figure 2).

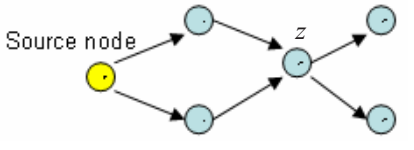


Fig. 2. Illustration of a double path

There are up to three routes to a hub in its routing table.

4 Simulation

UCLA's Global Mobile Simulation Systems Library (GloMoSim) [1] was employed with network nodes modelled after the Xbow MICA2 mote [7]. The MAC protocol was implemented using the distributed coordination function (DCF) of IEEE 802.11. Radio transmission range for each node was 56 meters. Radio model was signal-to-noise (SNR) bounded. Propagation model was ground reflection (two-ray). Frequency

used was 433 MHz and the bandwidth used was 76800 kbps. These settings are fixed throughout each simulation experiment conducted. Each simulation was executed for 60 minutes and the size of data payload was 24 bytes.

4.1 Performance Metrics

Performance of each protocol was analyzed using the following performance metrics.

(a) Packet Delivery Ratio (PDR): proportion of packets successfully delivered.

$$\frac{\text{Total number of data packets received}}{\text{Total number of data packets sent}} \times 100\%$$

(b) Packet Latency: average time for hub to receive data packet from source node.

$$\frac{\sum \text{Individual data packet latency}}{\text{Total number of data packets delivered}}$$

It includes processing times at various layers of each node, route recovery and repair, retransmission, queuing, propagation, transfer times etc.

(c) Energy Consumption: energy consumed per delivered packet.

$$\frac{\text{Total energy consumed} = \sum \text{Individual node's expended energy}}{\text{Total number of packets delivered}}$$

Node energy consumption model in GloMoSim is used, where energy used by a node for each packet transmitted at the radio layer/received at the MAC layer is:

Energy expended (J) = Voltage (V) X Current (A) X Transmission/Receive Duration
= 3.0V X 0.0052A X ((packet size X 8) / bandwidth)

Energy consumed will increase proportionally with the number of packets sent. Hence, energy consumption also contributes to the total overhead of a protocol.

(d) Packet Collisions: the total number of packet collisions that occurred during the simulation. It includes collisions by both control and data packets.

4.2 Non-ideal Conditions

Simulations were carried out for networks of 50 to 500 nodes with 10% and 50% active source nodes. This enables protocol scalability with network size to be evaluated for increasing traffic volume. Active source nodes were randomly chosen and data packet rate at each source node was 1 per 30 seconds. A noise model where every node except the hub takes on a random noise factor between 10% and 50% was used. The noise factor is the probability that a packet received by the node is corrupted or lost in transmission. A fault model where 50% of the nodes, except the source nodes, were randomly selected to fail at a random time within the simulation duration was used. For each network size, results were averaged over 30 runs.

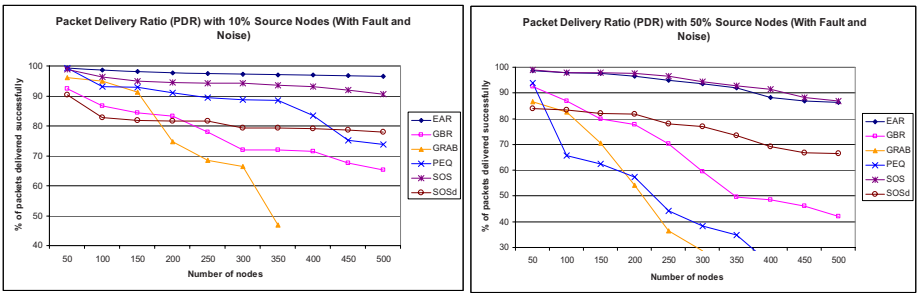


Fig. 3. Packet Delivery Ratio in Non-Ideal Conditions

With 10% source nodes, PDR of PEQ averages above 80% for networks of 400 nodes or less. PEQ is able to maintain a reasonable PDR even at high node failures of 50%. However, PDR drops significantly when active sources increase to 50%. With increased number of active sources, there are more data as well as control packets. PEQ employs an ACK-based scheme that results in an increase in packet collisions - more packets are dropped thus reducing the PDR. A less drastic trend is observed for SOS_d due to the additional path used by each source node. Duplicate data packets transmitted from each source node increases collision rates as in PEQ but performance is increasingly better with more active source nodes as path redundancy in SOS_d is restricted to only source nodes. Figure 3 shows that both SOS and EAR maintain a reasonable PDR above 85%. PDR for GRAB is the lowest for both 10% and 50% active sources. GRAB uses flooding to forward the data to the hub; this generates a significantly large number of data packets in the network and results in over-utilization of bandwidth. Average PDR with 10% is thus better than for 50% active sources where there are considerably more packet collisions and packet loss.

Expectedly, PDR for 50% active sources decreases for all protocols as network size increases. Neighbours within transmission range of a node increases leading to increased collisions. GBR, EAR and SOS protocols make use of a single route for each data packet (selective broadcasting). Reliability of packet delivery is not necessarily directly proportional to the number of multiple paths from source to hub. GRAB and SOS_d achieve fault tolerance by sending duplicate copies of data packets along different routes. However, the PDR of SOS_d for both 10% and 50% source nodes scale better with network size, even exceeding GBR for large networks in excess of 400 nodes. Reconfiguration in GBR does not consider link quality.

GBR maintains relatively low packet latency for 10% active source nodes while it has the lowest packet latency for 50% active sources. In larger networks, GBR's fast and simple random routing has greater chance of locating an active link. However, it does not guarantee that a complete route is available. Also, GBR only updates nodes with network configuration changes when nodes' energy threshold < 50%. Performances of SOS and SOS_d are competitive with EAR. In EAR, the need to continuously process piggybacked control information increases the processing time of data packets resulting in increased latency. SOS informs its neighbours that a change has occurred only when it detects a problem with its intended receiver node. This is more evident with networks in excess of 350 nodes and 50% active nodes.

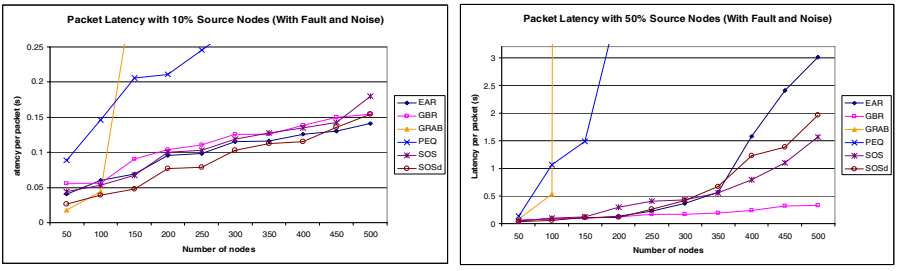


Fig. 4. Packet Latency in Non-Ideal Conditions

For networks < 450 nodes and 10% active sources, SOS_d has lower packet latency than SOS and EAR. A double path allows one data packet to reach the hub first while recovery is in progress in the other failed path. Any network reconfiguration process will have a reduced effect on latency. With 50% active sources, EAR's piggybacking advantage is no longer significant with network sizes > 350 nodes. Here, control overhead becomes considerable and compromises latency. With larger networks, SOS reverses its trend against SOS_d as the chance of avoiding failed paths increases with more neighbours available. Now, double transmissions in SOS_d become overheads.

With PEQ and 50% active sources in larger networks, increased transmissions are accompanied by increased acknowledgements. PEQ has to wait for an ACK message timeout to detect a problem before path repair begins. The next data packet has to wait for PEQ to find a new path before it can be relayed. Thus, PEQ's packet latency is higher due to no immediately available alternative routes when transmission fails. GRAB's use of flooding generates large quantities of redundant packets, resulting in unacceptable levels of collisions (also see Figure 6), severely degrading latencies.

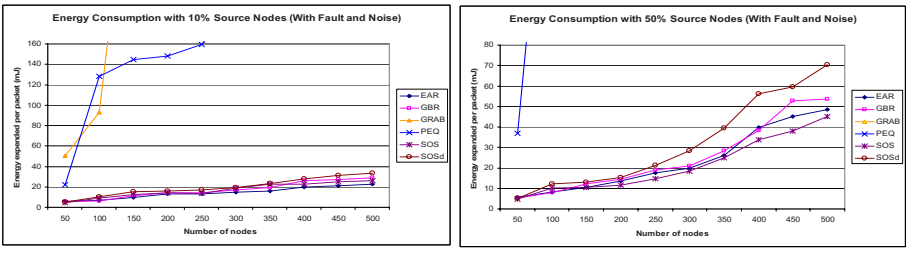


Fig. 5. Energy Consumption in Non-Ideal Conditions

GRAB has the highest energy consumption for both 10% and 50% active sources. This is one of the drawbacks of flooding-based strategies to achieve reliability. During routing, a large amount of redundant data packets are generated (Figure 5).

PEQ's ACKs raise packet transmissions two-fold and reduce its energy efficiency to below that of SOS, SOS_d, EAR and GBR, which are not acknowledgement-based. This is more evident with 50% active sources. Performance of SOS_d may be compromised in a high traffic and noisy environment. Sending duplicate copies of the

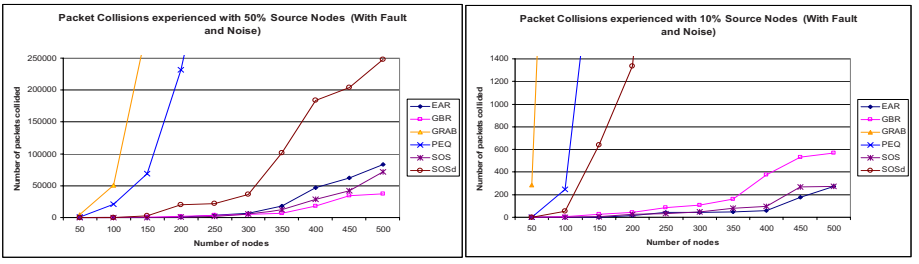


Fig. 6. Packet Collisions in Non-Ideal Conditions

same packet over a second path consumes 100% extra energy and increases traffic loads in a congested network resulting in more collisions and packet loss.

Figure 6 shows that as the number of active sources increase, collisions become a significant performance factor affecting all routing protocols. Extra overhead will be involved in dealing with collisions resulting in less packets delivered successfully. Packet collisions cause unnecessary retransmissions of packets and will impact the latency and energy efficiency of the protocols. PEQ and GRAB, which experience high levels of packet collisions, experience large latencies and high energy consumption figures. With PEQ, increased packet collisions result as a consequence of its ACK-based scheme. With 50% active sources in larger networks, this is more evident allowing SOS to better it.

Both EAR and SOS have lower packet collisions compared to other protocols. EAR uses piggybacked control information to detect node failures. For lower traffic levels, EAR experiences the least number of packet collisions. However, as the traffic levels increase, more failed transmissions are experienced. EAR will send out recovery messages whenever a node detects a problematic neighbour. This results in more control packets exacerbating the collisions. SOS, however, sends out recovery messages only after error detection during transmission. Hence, the situation reverses with SOS experiencing lower packet collisions than EAR with 50% active source nodes. SOS_d experiences higher packet collisions than SOS, EAR and GBR as it exploits an additional path to transmit data packets at each source node. This is moderated to an extent as the path redundancy is restricted to only source nodes.

With 10% active source nodes, GBR experiences more packet collisions than SOS and EAR. The situation reverses with 50% active source nodes. With more active sources transmitting, levels of packet collisions are higher for every protocol. However, this congestion of packet traffic has slightly less effect on GBR due to its stochastic random route selection, enabling it to avoid more collisions with "well-spaced" transmissions. However, a randomly selected path may not lead to successful delivery due to failed nodes as seen in GBR's comparatively low PDR (Figure 3).

5 Conclusions

Several self-reconfiguring routing protocols were evaluated against two variations of SOS. Results show the effectiveness of both protocols' performance. SOS is preferred

over SOS_d in WSNs with moderate to heavy data traffic while SOS_d, with constrained broadcasting, performs better in lightly loaded networks. Results showed that SOS and SOS_d are able to achieve competitive levels of PDR, energy efficiency and packet latency, with comparatively less packet collisions in the former. Limitations identified include poor performance in having unrestricted message path redundancy and the use of acknowledgement-based control. Finally, SOS and SOS_d do not require any changes to the MAC layer to achieve comparable performance to EAR. Practical realization will therefore be easier as only the network layer needs to be implemented.

References

1. Ahuja, R., Bagrodia, R., Bajaj, L., Gerla, M., Takai, M.: GloMoSim: A Scalable Network Simulation Environment. Technical report 990027, UCLA, Computer Science Dept (1999)
2. Akyildiz, I.F., Cayirci, E., Sankarasubramaniam, Y., Su, W.: A Survey on Sensor Networks. IEEE Communications, 102–114 (2002)
3. Bourkerche, A.: Performance Evaluation of Routing Protocols for Ad Hoc Wireless Networks. Mobile Networks and Applications 9(4), 333–342 (2004)
4. Boukerche, A., Werner, R., Pazzi, N., de Araujo, R.B.: A fast and reliable protocol for wireless sensor n/ws in critical conditions monitoring apps. In: MSWiM, pp. 157–164 (2004)
5. Broch, J., Maltz, D.A., Johnson, D.B., Hu, Y.-C., Jetcheva, J.: A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In: Proceedings of ACM/IEEE International, Conference on Mobile Computing and Networking, pp. 85–97 (October 1998)
6. Bulusu, N., Estrin, D., Girod, L., Heidemann, J.: Scalable Coordination for Wireless Sensor Networks: Self-Configuring Localization Systems. In: ISCTA (July 2001)
7. CrossBow MICA 2 motes specification, http://www.xbow.com
8. Estrin, D., Girod, L., Pottie, G., Srivastava, M.: Instrumenting The World With Wireless Sensor Networks. In: Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001), Salt Lake City, Utah, USA (May 2001)
9. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: MobiCOM (August 1999)
10. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next Century Challenges: Mobile Networking for Smart Dust. In: Proc 5th ACM/IEEE Intl Conf Mobile Comp & N/wking, pp. 271–278 (1999)
11. Lewis, F.L.: Wireless Sensor Networks, Smart Environments: Technologies, Protocols, and Applications. John Wiley, New York (2004)
12. Loh, P.K.K.: A Scalable, Efficient and Reliable Routing Protocol for Wireless Sensor Networks. In: Ma, J., Jin, H., Yang, L.T., Tsai, J.J.-P. (eds.) UIC 2006. LNCS, vol. 4159, pp. 409–418. Springer, Heidelberg (2006)
13. Lu, S., Ye, F., Zhang, L., Zhong, G.: GRAdient Broadcast: A Reliable Data Delivery Protocol for Large Scale Sensor Networks. ACM Wireless Networks 11(2) (2005)
14. Schurgers, C., Srivastava, M.B.: Energy Efficient Routing In Wireless Sensor Networks. In: MILCOM (2001)
15. Ye, F., Zhong, G., Lu, S., Zhang, L.: GRAdient Broadcast: A robust data delivery protocol for large scale sensor networks. ACM Wireless Networks 11 (2005)
16. Ye, W., Heidemann, J., Estrin, D.: An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In: INFOCOM (2002)

PerSON: A Framework for Service Overlay Network in Pervasive Environments

Kumaravel Senthivel[1], Swaroop Kalasapur[2], and Mohan Kumar[3]

[1] Fujitsu Network Communications, Richardson, Texas, USA - 75082
[2] Samsung Information Systems America, San Jose, California, USA - 94086
[3] Department of Computer Science and Engineering,
The University of Texas at Arlington, Arlington, Texas, USA - 76019
kumar.senthivel@us.fujitsu.com,
s.kalasapur@samsung.com, kumar@cse.uta.edu

Abstract. With the increase in the number of mobile devices and their network capabilities, users expect transparent access to available services in their pervasive environment. However, heterogeneity and interoperability issues persist in existing mechanisms. In this paper, we present a lightweight framework, PerSON (Service Overlay Network for Pervasive Environments), to abstract the details of service provision and utilization in a pervasive environment. PerSON constructs an ad hoc service overlay network in the pervasive environment based on service requests. PerSON achieves high efficiency by combining service discovery with route discovery. PerSON is especially suitable for resource constrained devices as minimal information about discovered services and the neighboring devices is maintained. In particular, the proposed framework provides the overlay network for the earlier developed middleware for pervasive computing. We also describe the implementation of a prototype for emergency response system (ERS).

1 Introduction and Motivation

In 1988, Mark Weiser envisioned that the computing devices will recede into the background in pervasive computing environments [1]. Even though there has been tremendous progress in many related areas, research is still being done to address many challenges [2]. The most common challenges to creating a pervasive environment are service provisioning and utilization. In a pervasive environment, there are a variety of devices such as desktops, laptops, PDAs, cell phone, etc. These devices may be connected to different physical and logical networks like the Wireless LAN, Bluetooth, etc. The services and applications hosted on different devices should be able to discover and communicate with each other over heterogeneous networks.

In keeping with Mark Wesier's vision PerSON attempts to address two important objectives of pervasive environments: i) A service developer's focus should be on the details of what to provide in the service rather than how to provide the service; and ii) An application developer's interest in the services that can be utilized by the application rather than how to access them.

In this paper, we present a simple framework called PerSON (Service Overlay Network for Pervasive Environments) for constructing a service overlay network.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 671–682, 2007.

PerSON abstracts the complexity of the underlying heterogeneous networks and provides a simple application interface to the user for developing services and applications. We employ distributed query based architecture [3] [4] to discover services and dynamic source routing (DSR) [5] to discover routes across multiple networks. Service discovery is optimized by piggybacking the route messages in the service discovery messages. We also describe the implementation of a prototype for an Emergency Response System (ERS) [6] using the Pervasive Information Community Organization (PICO) [7] architecture.

2 Related Work and Limitations

Over the last few years, significant research has been done in the field of service provision and utilization. Almost all existing frameworks construct a service overlay network to provide additional services over the underlying network. Service Overlay Network (SON), proposed by the authors of [8], enhances the best-effort services in the Internet by providing QoS. An application overlay network, created in [9], detects and recovers from path outages and periods of degraded performance in the Internet within seconds. The peer-to-peer overlay networks like Kaaza and Gnutella provide content sharing. Content Addressable Network (CAN) [10], Chord [11] and Pastry [12] organize the overlay network based on the content attributes to optimize the search.

Recent research initiatives have led to the development of Konark [13] and JXTA [14] [15] that provide support for generic services in a dynamic pervasive environment. Though these frameworks have their own advantages, there are certain limitations when executed in a pervasive environment. XML is used for service description and other data messages in both Konark and JXTA. XML is flexible but not suitable for resource constrained devices. High processing power and huge memory is required to parse the XML messages. JXTA attempts to provide support for such devices by introducing a new version called JXTA for Java 2 Micro Edition (J2ME) or JXME. Konark is dependent on IP multicasting. The routing mechanisms provided by the underlying network are used to route messages. Bridging between different physical or logical networks is not supported. The services in one network cannot be accessed by the devices in a different network. Though JXTA supports different message transport bindings like TCP, HTTP and TLS, it requires special designated devices to discover routes and forward messages.

3 Architecture of PerSON

The services and applications developed using PerSON may be hosted on devices with different computing capabilities. The devices may be connected to different physical or logical networks. A service overlay network is created by the framework to ensure availability, accessibility and utility of services. The services in PerSON can be discovered and utilized by external services and applications, regardless of the topology and complexity of the underlying network.

Suppose three devices (Fig. 1) are connected in two different physical networks. For example, the cell phone and the laptop are connected using RFCOMM in the Bluetooth network, whereas the laptop and the PDA are connected using TCP/IP in wireless LAN. Generally, a service available on the PDA cannot be accessed by an application on the cell phone. If the application and service are developed using PerSON, then they are connected by the service overlay network. The two different networks are bridged by the laptop using PerSON. The underlying network complexity is hidden by the overlay network and service discovery and connectivity are abstracted from applications and services.

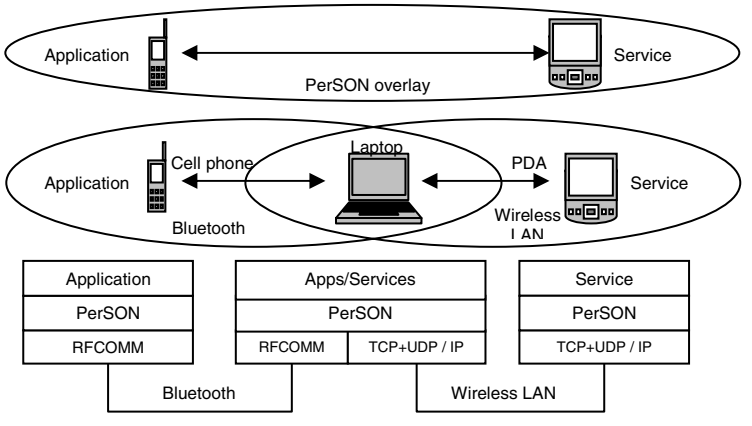


Fig. 1. PerSON Architecture

3.1 PerSON Stack

The PerSON stack (Fig. 2) is implemented by every device that participates in the overlay network. PerSON stack comprises three layers – network, device and service.

3.1.1 Network Layer

The network layer abstracts the various networks that connect the devices. A device is identified by a unique device identifier (DID). The addresses of the physical networks are masked by the DID. The network connections are made using the available transport on each physical network. The unicast and broadcast functions of the network layer are utilized by the device layer to exchange messages with the neighboring devices.

3.1.2 Device Layer

The functions provided by the network layer are utilized by the device layer. The details of service creation, discovery and utilization are abstracted from the services by the device layer. The device component is central to the PerSON architecture. The incoming messages are received by the device and delivered to the service connections, the resolver or the discoverer based on the message type. Whenever a message is received, the device table is updated with the physical address of the

neighboring device from which the message is received. The device is also responsible for choosing the right network connection to send the outgoing messages. The services hosted by a device are registered in the local services. Each service is identified by a unique service identifier (SID). A new service connection is spawned when the device receives a service request. The connection is utilized by the service layer to communicate with the application that requested the service.

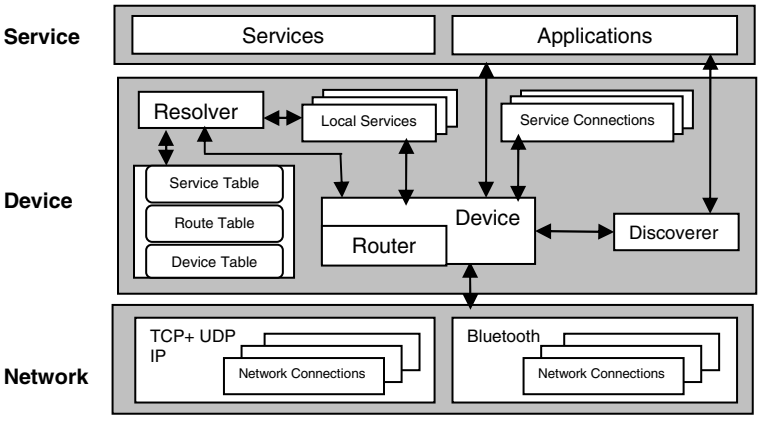


Fig. 2. PerSON Stack

The discoverer is responsible for finding the required services. When an application or service requests the discoverer to find a service, a query message is broadcast to other devices. The query contains the service description in simple text. The discoverer can be overridden to support user defined complex descriptions like XML format description. The scope of the discovery is restricted in terms of hop count.

The received query messages are processed by the resolver. If a matching service is found in the local registry then a result message is sent back by the resolver to the device that requested the service. The result messages received from other devices are also processed by the resolver. The SID of the requested service and the complete route to the device that hosts the service are contained in the result message. The service information is updated in the service table and the route information is updated in the route table. If the discoverer is overridden, then the resolver is also overridden to analyze the query message and search for service description.

The router is only used when the device is connected to multiple networks and the device is willing to act as a bridge between those networks. The router is not required for a device with only one network interface. When a message received is not intended for the device, the router forwards it to the next hop in the route.

3.1.3 Service Layer

The user defined services and applications are included in the service layer. The functions provided by the device layer are utilized to create new services. The applications in the service layer request the device layer to discover other services and

connect to the required services. Once a service connection is established by the device layer, an application and a service can communicate using the connection.

3.2 Physical Network Connections

In a pervasive environment, a device may be connected to more than one network. For example, a device may be connected to two IP networks supporting TCP and UDP transports and to one Bluetooth network using RFCOMM connections. The functionalities of the underlying network are utilized to broadcast query messages and unicast other data messages. IP and Bluetooth networks are supported by the current implementation of PerSON.

In IP networks, the query messages are broadcast using UDP transport. The device responding to the query, first establishes a TCP connection. In Bluetooth networks, when a device enters (or turned on) the network, the neighboring devices are discovered using the Bluetooth discovery mechanism. If a PerSON service is found, then a RFCOMM connection is established to those neighbors. Devices that can alternate between the master and slave modes (of Bluetooth) listen for incoming connections and also connect to other devices. Multiple connections with different devices are possible for such devices. Devices that are not capable of alternating between master and slave modes can have only one connection. Query messages are flooded using multiple unicasts.

3.3 Overlay Network

The overlay network in PerSON is constructed to facilitate service provisioning and utilization. By combining the service discovery with route discovery, high efficiency is achieved. The route information is piggybacked in the service discovery messages, so that additional route discovery messages are not required to discover the route between the service provider and the service consumer.

Distributed query architecture is used to propagate the service discovery queries. In a pervasive environment where only local scalability is considered the flooding mechanism, restricted by the scope is a simple method for service discovery. The devices do not depend on specific service directories to discover the available services. Each device broadcasts a query for the required service in its local network. All other devices in the local network process the query message, but only those devices that support routing to bridge different physical or logical networks will forward the queries.

Consider the network in Fig. 3(a) that comprises five different sub networks. Devices S, A, B, C, D and E belong to network I. Devices A, B, X, and Y belong to network II. Devices E, M, and N belong to network III. Device A and P belong to the network IV. Devices X and R belong to the network V. Only devices A, B and E have two different network interfaces and belong to two different networks. The device S sends a query for a specific service with a scope of 2. The query is broadcast in the local network of device S. The devices A, B, C, D and E in this local network process the query. But only those devices that are willing to route the messages will propagate the query to other networks. For example, B propagates the query and acts as a bridge between networks I and II. E cannot propagate the query because it does not support

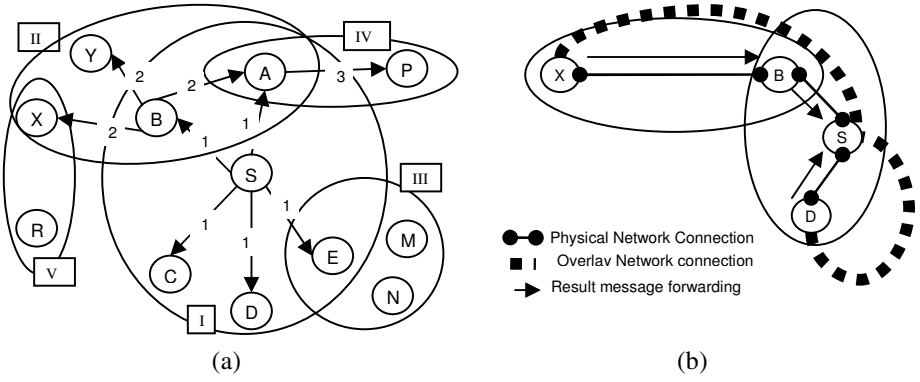


Fig. 3. (a) Query propagation in PerSON, (b) Overlay network in PerSON

routing. Duplicate queries are not propagated. Hence device A propagates the query to network IV only once, even if it receives the same query from devices S and B.

Whenever a query message (Fig. 4 (b)) is broadcast in a network, the DID of the device that propagates the query is added to the route information in the message. The device also specifies the physical network address through which other devices can be connected. The time for which the device can be expected to be available is also added to the message. When device B receives the query message from device S, it knows how to connect to device S if needed and similarly device X knows how to connect to device B. This information is cached in the device table (Fig. 4 (i)) and cleared from cache after the specified time. Each record in the device table contains the DID, available time and the physical address of the neighboring device.

Device X also knows the complete route to device S by reversing the route in the query message. The route information for device S is cached in the route table (Fig. 4 (j)), which is cleared when all the connections to the next hop are broken. The route information can also be cleared when any other device in the route sends a negative acknowledgement specifying that the route is longer valid. Each record in the route table contains the DID of the destination device and the list of all intermediate devices. A device may cache many routes to another device based on the route information specified in the messages it receives.

Suppose the required service is provided by device X, its result message (Fig. 4 (c)) to device S is sent via the resolver of device X. The resolver specifies the complete route in the message. The network layer chooses the first entry in the device table and tries to connect to the device if there are no previous connections. If the connection is successful, then the message is sent, else the network layer tries to connect using the next entry in the device table. If the message cannot be sent an error message is returned to the device layer. All routes that include this device are cleared from cache. The device layer retries to send the message with an alternate route. The message is dropped if there are no more routes to the destination device. When a device such as B receives the message (sent from X to S), it simply routes the message to S via the router in the network layer.

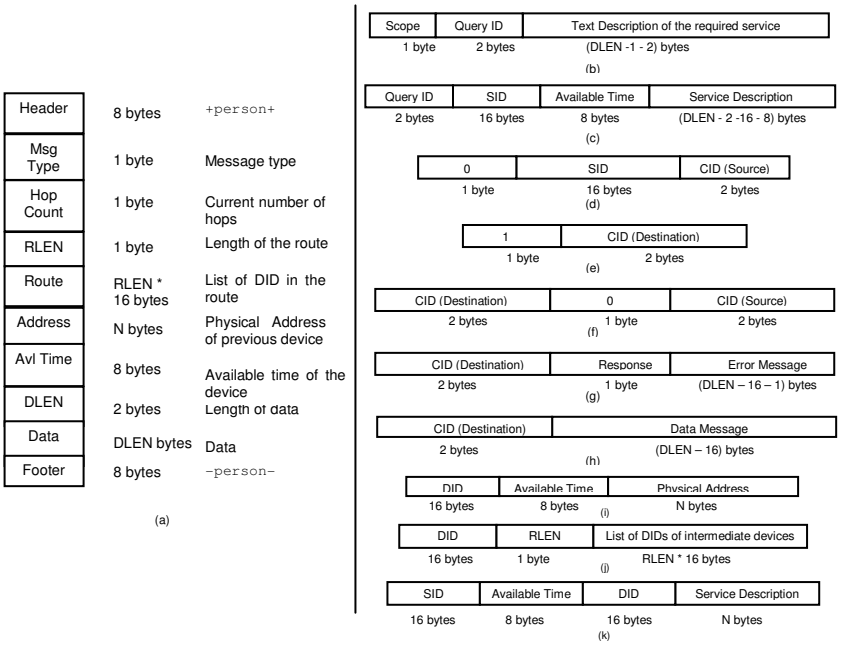


Fig. 4. (a) Message Format, (b) Query, (c) Result, (d) Connect Request, (e) Close Request, (f) Success Response, (g) Error, (h) Data, (i) Device table, (j) Route Table (k) Service Table

Suppose device D also provides the requested service then it sends the result message to device S. The overlay network (Fig. 3(b)) is formed only between the devices X and S and between devices D and S. A device that provides a service gets connected to a device that consumes the service. When the device S receives the result message, it caches the service information in the service table (Fig. 4 (k)) and the reversed route in the route table. Each record in the service table contains the SID, service available time, the DID of the device which provides the service and the service description.

An application on device S requests the device layer to connect to the required service by specifying the SID, The device layer retrieves the DID of the device X that provides the service from the service table. The first available route to device X is obtained from the route table. The device layer then requests the network layer to send the message to the DID of the next hop (B). Devices S and X exchange messages (Fig. 4), through device B, by using the connections previously established during the service discovery process.

4 Prototype Implementation

The PerSON framework is not dependent on a specific development language or operating system. The reference implementation of PerSON is developed using Java. The J2SE version of PerSON can be executed in powerful devices like desktops and

laptops. The J2ME version can be executed in resource constrained devices like PDAs and cell phones. The J2SE version supports both TCP/IP and Bluetooth networks whereas the J2ME version supports only Bluetooth networks. The reference implementation provides simple Java APIs to create, discover and utilize services.

The implementation of PerSON is used to provide the overlay network for the PICO middleware for pervasive computing. The main components of PICO are *devices*, and intelligent agents called *delegents.* PICO creates mission oriented dynamic and static *communities* of delegents that perform tasks for the users and devices.

4.1 PICO Implementation Using PerSON

The device layer and the service layer of PerSON integrate with the device layer and the delegent layer of PICO middleware. When a device is started, the different networks supported by PerSON are initiated. The TCP/IP networks open a UDP server socket and wait for incoming query messages. The Bluetooth networks explore other devices in the vicinity and connect to the devices that implement PerSON. Additional functionalities like the definition of system characteristics are added by the device layer of the PICO framework. The delegent layer of the PICO middleware is similar to the service layer of the PerSON framework.

4.2 Emergency Response System

A prototype for enhanced Emergency Response System (ERS) is implemented by employing services developed using the PICO middleware framework. In the scenario for an ERS, when an automobile on a highway meets with an accident, a crash detector application running on the automobile's computer detects the crash through various sensors. The user may or may not require immediate help based on the intensity of the crash. In order to avoid false alarms the application has to confirm whether the user needs any help. So, it communicates with a UI Service that can inquire the user and confirm the crash. The UI service may be located on the user's watch or PDA or any device that can display the message and get the user's attention. If the user's response is negative or if the user does not respond, the computer assumes that a real crash has occurred. Now the application finds a Dialer Service that can request help from an emergency response center. The dialer service may be located on the user's cell phone or any communication device that can access the Internet. The crash detector application requests the dialer service to place an emergency call with information about the location of the incident and user's personal details for retrieving the available medical history. The emergency response center dispatches an emergency medical services (EMS) vehicle nearest to the location.

In the prototype for the ERS, the crash detector application is executed on a Fujitsu laptop, equipped with a DBT120 Bluetooth dongle driven by Microsoft Bluetooth stack. The UI service is executed on a Sharp Zaurus 5500 PDA with a compact flash wireless LAN card. The Dialer service is executed on the Nokia 6230 cell phone. A light sensor is used to simulate the crash. The laptop and PDA communicate using an ad hoc wireless LAN. The cell phone and the laptop communicate using Bluetooth network.

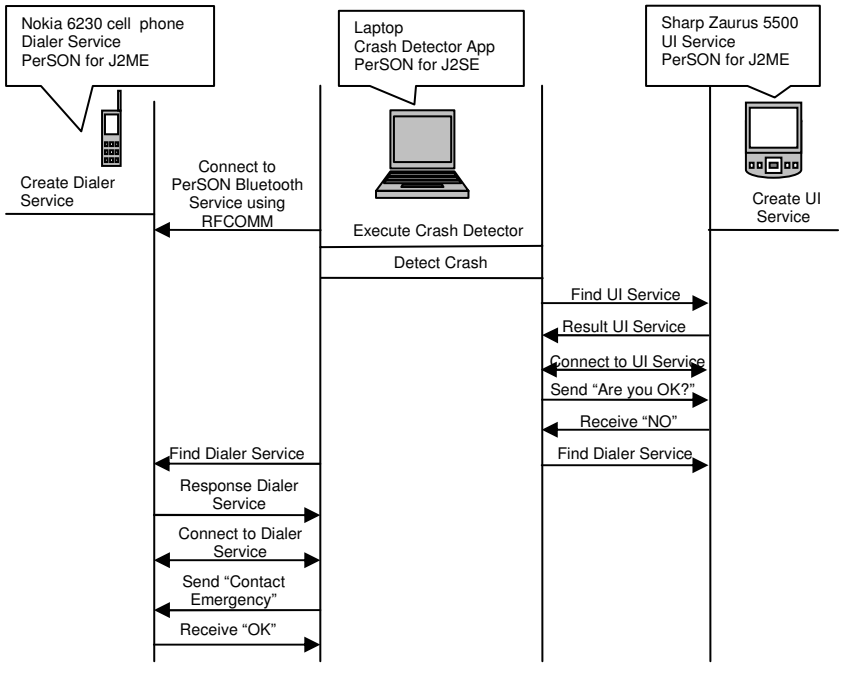


Fig. 5. Execution of ERS services using PerSON

The services and the application are initialized and executed as shown in Fig. 5. The PerSON framework is utilized by the PICO middleware to initialize the Bluetooth network in the cell phone and the laptop. A Bluetooth service of PerSON is created on the cell phone and the laptop. When the Bluetooth network is initiated in the laptop the Bluetooth discovery mechanism discovers the cell phone. The laptop is connected to the "PerSON" service on the cell phone using RFCOMM connection. The dialer service and UI service are created and registered only in the devices in which they are hosted. When the crash detector detects a crash, a query message is broadcast using UDP in the IP network and unicast to the cell phone using the RFCOMM connection. The query message is received by PerSON on PDA and a TCP connection is created between the laptop and the PDA. The response message for the UI service is unicast over the TCP connection. A service connection is established between the crash detector application and the UI service. The data messages are exchanged over this service connection. Similarly, the dialer service is discovered and utilized by the crash detector application.

5 Features of PerSON

5.1 Light-Weight Framework

In PerSON, the directory of available services in the network is distributed. Each device stores only the information about the provided services. Other devices cache the information for the discovered services and the complete route to those services.

Table 1. Comparison of PerSON, JXTA, and Konark

Feature	PerSON	JXTA	Konark
Support for resource constrained devices	Yes. Uses binary messages and is light-weight	Partial support. Uses XML messages for JXTA and binary messages for JXME	No support. Uses XML messages
Support for heterogeneous network	Supports TCP/IP and Bluetooth networks	Current implementation supports only TCP/IP networks. Supports different message transport binding	No support. Depends on IP multicasting
Support for multiple network interfaces	Yes.	Yes	No
Support for dynamic networks	Yes	Yes	Yes
Service discovery	Highly decentralized Only reactive.	Uses distributed service indices. Reactive and proactive	Highly decentralized Reactive and proactive
Service description	Simple Text	XML	XML
Support for service composition	No	No	No
Platform Independence	Yes	Yes	Yes
Scalability	Locally scalable	Scalable over Internet	Depends on the scalability of IP multicasting

Since there is no central repository, PerSON requires less memory to store the minimal service information, compared to the service-coordinator based approaches. PerSON uses simple text to describe services. The memory required to cache the discovered services is considerably reduced. A typical record in the service table requires 16 bytes for the service identifier, 8 bytes to specify the available time, 16 bytes the device identifier and say, 256 bytes for the service description. A total of 296 bytes is required to cache the information of a service. A typical record in the route table for a route with 3 hops requires 16 bytes for the destination device identifier, 1 byte for the length of the route and 32 bytes for the route. A total of 47 bytes is required to cache the route. A record in the device table that stores the information about another device on an IP v4 network requires 16 bytes for the device identifier, 8 bytes for the available time and 6 bytes for the IP address and the port number. A total of 30 bytes is required to cache the device information.

5.2 Heterogeneous Network Connectivity

The reference implementation of PerSON is capable of bridging IP networks and Bluetooth networks. Devices connected to only Bluetooth network can communicate with devices connected only to an IP network using any device that is connected to both networks. The bridging is done at the application layer above the TCP/IP stack and the Bluetooth stack. A device willing to route the messages should include the router component of PerSON stack. The router component simply forwards the message to the next hop in the route.

5.3 Dynamic Service Discovery and Routing

In order to support heterogeneous networks, the messages have to be routed in the overlay network. Dynamic source routing protocol is used in the overlay network for routing messages in the ad hoc environment. The route discovery is merged with service discovery. The route information is piggybacked in the service discovery query messages. A device receiving the query message knows the route to the device that requires the service. Similarly a device receiving the response message knows the route to the service provider. The service and route information are cached in each device. The complete route is specified in each message. The route includes only the list device identifiers of intermediate devices. Each intermediate device is responsible for connecting to the next hop in the route using the physical network connections.

6 Conclusion

PerSON is a framework for constructing service overlay network to facilitate interoperability in pervasive environments. It provides a light-weight abstraction of the complexities of the underlying heterogeneous network to the applications and services. A reference implementation of the framework is developed and used to provide the overlay network for the pervasive computing. The prototype is implemented using the services for an enhanced emergency response system. Future work includes support for service composition and security. Authentication and authorization will be incorporated in future versions of PerSON. The framework will also be evaluated for scalability and throughput.

Acknowledgements. The material presented in this paper is based on the work supported by the National Science Foundation Research Grants STI -0129682 and IIS-0326505.

References

1. Weiser, M.: The Computer for the 21st Century. Scientific American 265(3), 94–104 (1991)
2. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications 8(4), 10–17 (2001)

3. Perkins, C.E., Koodli, R.: Service discovery in on-demand ad hoc networks. IETF Internet Draft (work in progress) (2002)
4. Engelstad, P.E., Zheng, Y.: Evaluation of Service Discovery Architectures for Mobile Ad Hoc Networks. In: Wireless On-demand Network Systems and Services. Proceedings of the Second Annual Conference on. pp. 2–15 (2005)
5. Perkins, C.E., Royer, E.M., Das, S.R., Marina, M.K.: Performance comparison of two on-demand routing protocols for ad hoc networks. IEEE Personal Communications 8(1), 16–28 (2001)
6. Kalasapur, S., Senthivel, K., Kumar, M.: Service Oriented Pervasive Computing for Emergency Response Systems. In: Pervasive Computing and Communications Workshops 2006. Ubicare'06. Proceedings of the Fourth Annual IEEE International Conference on, pp. 517–521 (2006)
7. Kumar, M., Shirazi, B., Das, S.K., Singhal, M., Sung, B.Y., Levine, D.: PICO: A Middleware framework for Pervasive Computing. IEEE Pervasive Computing 2(3), 72–79 (2003)
8. Duan, Z., Zhang, Z.L., Hou, Y.T.: Service overlay networks: SLAs, QoS and bandwidth provisioning. Network Protocols. In: Proceedings of the 10th IEEE International Conference on, pp. 334–343 (2002)
9. Anderson, D., Balakrishnan, H., Kaashoek, M., Morris, R.: Resilient Overlay Network. In: Proceedings of ACM Symp on Operating Systems Principles (2001)
10. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of ACM SIGCOMM, pp. 161–172 (2001)
11. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for Internet applications. Networking. IEEE/ACM Transactions on 11(1), 17–32 (2003)
12. Rowstron, A., Druschel, P.: Pastry: Scalable, Distributed, Object Location and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
13. Choonhwa, L., Helal, A., Desai, N., Verma, V., Arslan, B.: Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services. Systems, Man and Cybernetics. IEEE Transactions on 33(6), 682–696 (2003)
14. Li, G.: JXTA: a network programming environment. IEEE Internet Computing 5(3), 88–95 (2001)
15. Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., Hugly, J., Pouyoul, E., Yeager, B.: Project JXTA 2.0 Super-Peer Virtual Network (2003), http://www.jxta.org/project/ www/docs/JXTA2.0protocols1.pdf

Universal Adaptor: A Novel Approach to Supporting Multi-protocol Service Discovery in Pervasive Computing

Joanna Izabela Siebert[1], Jiannong Cao[1], Yu Zhou[1,2], Miaomiao Wang[1,3], and Vaskar Raychoudhury[1]

[1] Department of Computing,
The Hong Kong Polytechnic University,
Kowloon, Hong Kong
[2] Department of Computer Science and Technology,
Nanjing University,
Nanjing, China
[3] The University of Science and Technology of China,
Hefei, Anhui, China
{csjsiebert,csjcao,csyuzhou,csmmwang,
csvray}@comp.polyu.edu.hk

Abstract. Service discovery is an important and challenging issue in pervasive computing. To date, many service discovery protocols have been proposed and new ones are under development. However, pervasive computing involves applications running in heterogeneous environments and application developers must cope with diversity of network infrastructures, middleware platforms, and hardware devices. There is a need for integrating or bridging these service discovery schemes in order to support the discovery of the services available in different environments. In this paper we study how to provide service discovery across different environments supported by different service discovery systems, which may use standard protocols as well as tailor-made mechanisms. We propose the Universal Adaptor (UA) approach, which consists of two major components: the Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP is the universal set of primitives used by the user to discover the services across different environments, while UAM provides the mapping between the Universal Adaptor Primitives to the primitives used in various service discovery systems. We have developed a prototype of the proposed approach and we describe the implementation issues and the experiment results.

Keywords: Pervasive computing, Service discovery, Heterogeneous environments.

1 Introduction

Along with the advances in pervasive computing technologies, users are no longer constrained to only a single computing environment but can work across different environments, meeting with a diversity of software, hardware devices, and network infrastructures. Heterogeneity of the environments brings significant problems that

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 683–693, 2007.

application developers must cope with. For instance, devices must be able to discover and share services among each other. However, manual configuration may require special skills together with long time to set up. Therefore, service discovery is an important, challenging task in pervasive computing.

Many service discovery protocols have been proposed. Examples include UPnP, SLP, and Jini [1, 2, 3]. They allow automatic detection of hardware devices, software, and other resources in a computing environment along with services offered by the various kinds of entities. Existing service discovery protocols differ in the way service discovery is performed, and no single protocol is suitable for all the environments. Due to the differences in the service discovery approaches implemented, a user working in one environment may not be able to search for the services available in another environment that suits the user's need. To support service discovery across different environments, new techniques are needed to integrate or bridge the service discovery protocols used in a diversity of environments.

Works can be found on providing interoperability of service discovery protocols [11, 12, 13, 14, 15, 16]. In this paper, we study how to provide service discovery across different environments supported by different service discovery systems, which may use standard protocols such as SLP and Jini, as well as tailor-made mechanisms that support multiple protocols within an environment, such as ReMMoC [13].We propose the Universal Adaptor (UA) approach, which consists of two major components: the Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP is the universal set of primitives used by the user to discover the services across different environments, while UAM provides the mapping between the Universal Adaptor Primitives to the primitives used in various service discovery systems. The Uniform Adaptor can be implemented in any environment, independent of the service discovery system used in that environment. This approach enables users to discover available services with no knowledge of the service discovery system adopted in an environment. We have developed a prototype of the proposed approach and we describe the implementation issues and the experiment results.

Comparing with existing approaches, Universal Adaptor provides a simple and flexible solution. It provides only a single set of APIs and supports not only all existing but also future service discovery systems. It is lightweight and easy to implement in diverse infrastructures and to use by users.

The rest of the paper is organized as follows. In Section 2, we present the related work. In Section 3, we present architecture overview of proposed system. Section 4 shows the design of Uniform Adaptor Primitives. Section 5 describes how to realize the Uniform Adaptor Mapping. In section 6, we describe the implementation of a prototype of the proposed UA approach. We conclude the paper in Section 7 with the discussion of our future work.

2 Related Work

Many service discovery protocols have been proposed for different applications in various environments [5, 3, 1, 4, 2]. Survey and comparisons between the existing service discovery protocols can be found in [6, 7, 8, 9, 10]. New service discovery protocols targeting at pervasive environments are also emerging [16, 18, 19]. In this

section, we focus on the works that support interoperability of the service discovery protocols. Several approaches on supporting multi-protocol service discovery have been proposed [11, 12, 13, 14, 15, 16].

All the existing service discovery mechanisms realize the concept of client/server application. Clients are entities that need some functionality (service) and servers are entities existing in the environment and offering this functionality. Service discovery is the framework for connecting clients with services. Existing works towards achieving multi-protocol service discovery use a middleware approach but differ in where the interoperability support is provided. The middleware can be on the service side, client side, or intermediate entity.

An example of providing the support on the service side is INDISS [14] designed for home networks. INDISS provides parsers and composers, which decompose a request from the source service discovery protocol into a set of events and then compose them into message understood by target service discovery protocol.

INDISS can also be deployed on the client side. Another example of middleware on the client side is ReMMoC [13]. In ReMMoC, the client side framework provides the mappings between all the supported service discovery mechanisms. Abstraction of discovery protocols to the generic service discovery is achieved by using a generic API or doing discovery transparently to the client. In this approach, all possible mappings need to be provided at the client side.

Service side support can also be based on service proxies [16]. When a new service appears or disappears in one of the environments, the framework detects it and creates or removes its proxies from other environments. However, this approach requires dynamic service proxies to be implemented for each environment, which increases developer's workload.

Another way of providing interoperability support is to provide an intermediate entity [12, 11]. In Open Service Gateway Initiative OSGi [12] all the connections and communications between the devices are brokered by a Java-based platform. An internal service registry exists in the framework. Interoperability is achieved by providing an API to map the given service discovery protocol to OSGi and vice versa. Supporting new service discovery protocols requires defining new APIs for them. Gateway functionality is also utilized by protocol adapters in the FuegoCore Service broker [11] designed for mobile computing environments. The service broker registers the mappings between its internal template and the external templates used by different service discovery protocols. Extending the FuegoCore service broker to new service discovery protocols requires creating and deploying additional service discovery protocol adapters. INDISS [14], mentioned above, can be deployed as an intermediate entity as well. The intermediate entity approach requires broker to integrate all the adapters into one system. In a network with a large number of service discovery protocols the framework may not be scalable.

Another approach is providing service that discovers services across different environments [15]. In MUSDAC, it registers itself in all the environments, so clients can use whatever protocol to discover it. However, clients must have the knowledge about MUSDAC and the process of discovering the service has high processing requirements.

The centralized approach has the scalability problem. In the future pervasive environment, an unlimited number of service discovery mechanisms will emerge.

Providing all possible translations in one middleware would result in a very heavy system. Our work reported in this paper is based on the analysis of the following requirements on the interoperability system for pervasive computing:

- no change should be imposed on the existing service discovery mechanisms
- no change should be imposed on the services registered in domains
- no functionality of the environment should be compromised
- the system should be lightweight, scalable, and extendable.
- support both standard and tailor-made service discovery mechanisms

Our approach addresses all of these requirements. In the following sections we describe design of the system.

3 System Model and Architecture

The aim of our study is to support interoperability between existing as well as yet unknown service discovery systems.

In this section, we describe the system model of the proposed UA approach. We define an *Environment* as a Service Discovery Domain, where a native service discovery system is able to find a specified resources if they are available in the domain. A native service discovery system can take the form of an existing service discovery protocol, e.g SLP, Jini. Moreover, it can be a tailor-made mechanism, e.g. ReMMoC which supports the interoperability of service discovery within an environment.

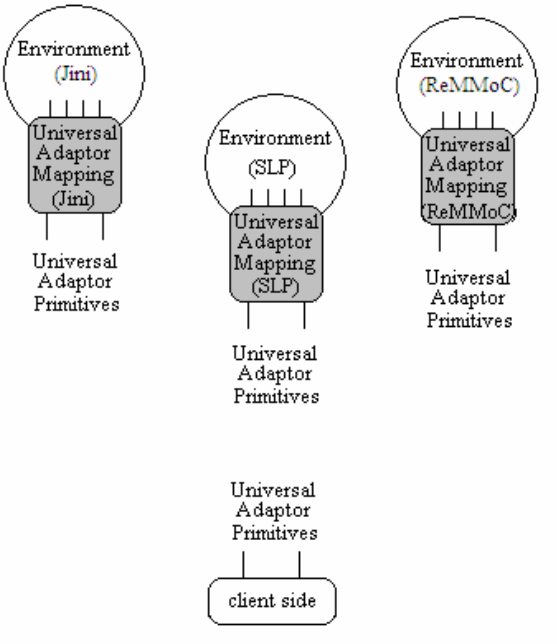


Fig. 1. The system model

Services in an environment can be provided by hardware devices, software, and other entities. Examples of software services can be weather forecast, stock quotes, and language translation. Hardware devices can provide services directly or via another device. For example, the printing service can be provided by a Jini enabled printer, or by a printer that is attached to computer running Jini software.

Figure 1 shows the major entities in the UA system. There are two major components: The Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP acts as uniform interface to the client. The client makes use of UAP to discover and access services. The main function of UAM, on the other hand, is to provide the mapping from UAP to the specific primitives of service discovery protocols (SDP).

The Universal Adaptor (UA) is installed in each environment. On the client side, the UA provides Universal Adaptor Primitives (UAP) for the communication between the client and UA. On the service side, UA is a component tailored to the environment, mapping the UA primitives to those of the native service discovery system and performing service discovery on behalf of the client.

4 Universal Adaptor Primitives (UAP)

We studied existing service discovery protocols and observed that all the service discovery systems provide the same functionality for end users and differ only in the underlying models and operations. We can abstract the common characteristics of existing approaches and to provide a universal set of primitives that enable service discovery across diverse environments.

The format of the proposed primitives is the following:

[Return_Values] Primitive_Name [Parameters]

Primitive_Name represents the functionality requested by the client; *Parameters* is a set of attributes required in the discovery process; *Return_Values* is set of values returned to the client.

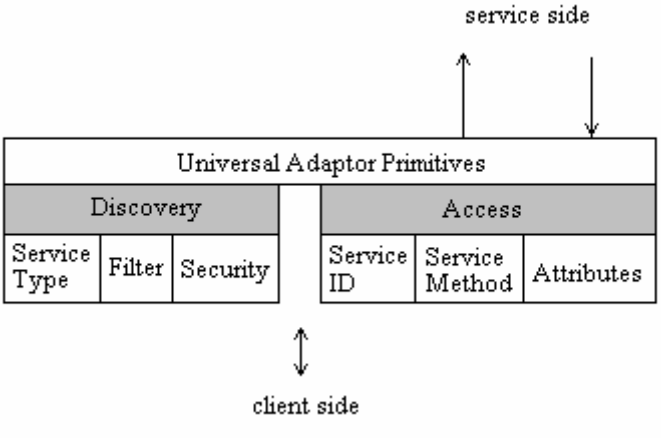


Fig. 2. Universal Adaptor Primitives

The discovery process starts when the client expresses his interest in a particular service. Next, the discovery system searches for the service and returns result to the client. The common feature of all the existing service discovery mechanisms is that they support the process of looking for service and enable the access to the service. Therefore, we propose two universal primitives: *Discovery* and *Access*. Specific UA Primitives with parameters are shown in Figure 2.

The Discovery primitive is of the following format:

[RV_ST, RV_F, RV_S] Discovery (Service Type, Filter, Security).

A client requests the specific service by providing its type in the *ServiceType* parameter. The attributes of the service are specified by the client in the *Filter* parameter. If the authentication of the client for using the service is required, the client provides the required information in the *Security* parameter. The Discovery primitive provides three return values. *RV_ST* is the return value containing list of the discovered services of the requested type. Each of the discovered services has a unique ID assigned by the UA. *RV_F* provides a list of attributes for each of the discovered services. For example, for a printing service, an attribute can be specified to indicate whether the printer supports colour printing or only black&white. *RV_S* is a return value indicating the authentication status.

After the successful discovery of the requested service, the client selects one of the returned services and access the service by using the Access primitive, which is of the following format:

[RV_status] Access (ServiceID, Service Method, Attributes.)

The client provides the parameter *ServiceID*, the ID of the selected service. The *ServiceMethod* parameter specifies the specific method provided by the service. The *Attributes* parameter is a set of attributes specific to the particular service. *RV_status* is the return value indicating the status of accessing the service, e.g., success or fail.

5 Universal Adaptor Mapping (UAM)

In this section, we describe UAM, which performs the mapping from UAP to the SDP specific primitives used in the target environment. As we mentioned before, there is a number of currently existing service discover mechanisms, including protocols and interoperability systems. Moreover, new mechanisms are under development. Therefore, the many-to-many approach providing a mapping between the primitives of each pair of these mechanisms would result in a heavy system with difficulties in adapting to the future service discovery mechanisms. Our approach is a one-to-many approach, providing the tailored mapping between the UAP to each native protocol used in the target environment.

For the structure of components of the UAM, please refer to Figure 3. More specifically, the functions of UAM components are described in Table 1.

Recall that to discover or access a service, the client uses the UA primitives described in previous section. *Translator* is a component responsible of mapping the UA primitives to the primitives of the native protocol used in the Environment. The translation process takes place each time when a client sends a query.

Table 1. Function of UAM components

Component	Function
ESD Description	Rules specifying the characteristics of the service discovery protocol used in the particular environment, in the form of the primitives extracted from the native protocol
Translator	Module that translates UA primitives into native primitives, and outputs an algorithm for service discovery using the native primitives
ESDA	Agent that uses the algorithm generated by Translator to perform service discovery in the target environment. Depending on the native service discovery protocol, ESDA can register for service advertisements or performs active service discovery

Translator performs mapping based on the *ESD Description* (Environment Service Discovery Description), which is set of rules specifying the characteristics of the service discovery protocol used in the particular environment. Because different environments adopted diverse models at the low level, the rules used by *ESD Description* are tailored to each environment and are described in the form of the primitives extracted from the native service discovery protocol. *Translator* outputs an algorithm for the service discovery using the native primitives.

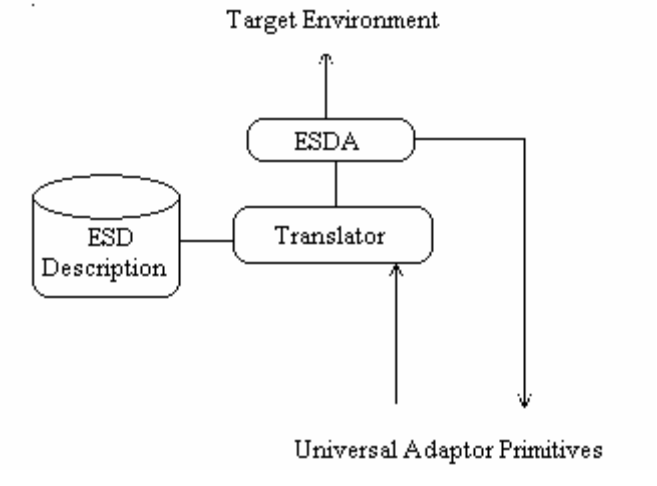


Fig. 3. Universal Adaptor Mapping

ESDA (Environment Service Discovery Agent) is representing the client in the process of discovery in the target environment. It uses the algorithm generated by *Translator*. From the server's point of view, *ESDA* is the same as any other client in the environment using the native protocol for service discovery. All the communications take place between the service discovery protocol and *ESDA*. Another function of *ESDA* is to provide the client with the return values from environment, e.g. information about the attributes.

6 Prototype Implementation and Experience

We have developed a prototype of our proposed UA approach. The system has two service discovery environments. One uses Jini and the other uses SLP as the native service discovery protocols. For Jini, we use Jini2_1 [20] which is an implementation of the Jini technology from Sun Microsystems. For SLP, we use a pure Java implementation of SLP – jSLP [21].

For each environment, a tailored UA is implemented. The implementation of UA for each environment consists of two parts – client side UA and service (environment) side UA, both are written in Java. Client side UA is the same for both Jini specific implementation and SLP specific implementation. On the service side, the Java UA listens on a port for the incoming requests. It parses the request, and uses reflection mechanism to invoke the corresponding service in the local environment, then passes the result back to the client.

As an example, we implemented a weather information service using Jini and SLP separately in two environments. We measured the performance of our UA system, in terms of the overhead in time comparing with the native service discovery protocols. Experiments are conducted with a desktop computer with CPU Pentium D 2.8GHz, 1G Memory and a laptop computer with CPU Pentium Mobile 1.6GHz, 512M Memory connecting to the LAN with 100Mbps Ethernet and 11Mbps 802.11b protocol respectively.

We first measured the time for service discovery in the two environments using native service discovery clients (both for Jini and SLP). Next, we compared that with the discovery time using JiniUA and SLPUA respectively. The results of the experiments are presented in Table 2.

In the Jini environment, the average service discovery time is 638 milliseconds, and in the case of using UA, the average time is 743 milliseconds. In the SLP environment,

Table 2. Discovery overhead of UA

	Jini Environment	**SLP Environment**
Service discovery time without UA [ms]	638	152
Service discovery time with UA [ms]	743	197
Overhead [%]	16	30

the average service discovery time is 152 milliseconds and the average time in the case with UA support is 197 milliseconds. The overhead that UA imposes on service discovery time is mainly introduced by the socket setup and request parse.

We also measured the time for service access in the Jini and SLP environments and compared them with the access time using UA. Results are presented in Table 3.

Table 3. Access overhead of UA

	Jini Environment	**SLP Environment**
Service access time without UA [ms]	832	223
Service access time with UA [ms]	926	268
Overhead [%]	11	20

In the Jini environment, the average service access time is 832 milliseconds, and in UA client, the average time is 926 milliseconds. In the SLP environment, the average service invoking time is 223 milliseconds, and the average time in the case with UA is 268 milliseconds. Similar to the discovery time experiments, the overhead that UA imposes service access time is related to the socket setup and request parse.

In summary, the experiment results show that the discovery and access time overhead imposed by UA is very small.

7 Conclusion

This paper presents Universal Adaptor, a novel approach towards supporting multi-protocol service discovery in pervasive computing. UA consists of Universal Adaptor Primitives and Universal Adaptor Mapping. It provides mapping from UAP to protocol specific primitives. The client makes use of UAP to discover and access services. UAM performs mapping from UAP to service specific SDP primitives. From the point of view of the service side, Universal Adaptor is a tailored component that uses native SDP and performs service discovery on behalf of the client.

Our contribution to the interoperability of service discovery is providing solution that in a lightweight manner bridges not only all existing but future service discovery systems, including standard ones, as well as service discovery mechanisms that support multiple protocols within the domain. Our implementation has shown that Universal Adaptor is a simple and flexible solution. It is easy in sense of writing the code, implementing in diverse infrastructures and using by client.

We will conduct more experiments to investigate the performance of the UA approach. In our future work, there are many issues to investigate. First, we plan to use mobile agent to find services in environments on behalf of users. Second, so far, we have assumed that the client will use UA primitives for service discovery and access, and not considered how to support clients using existing protocols. We will develop the mechanisms to provide transparent UA multi-protocol service invocation to these

clients. Finally, we will investigate the possibility to let the client take the Universal Adaptor and dynamically install it to the target environment.

Acknowledgements

This work is supported by Hong Kong University Research Grant Council under the CERG grant PolyU 5183/04E

References

1. Goland, Y., Cai, T., Leach, P., Gu, Y., Albright, S.: Simple Service Discovery Protocol 1.0. IETF, Internet-Draft Version 03 (October 1999)
2. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service Location Protocol, Version 2. IETF, RFC 2608 (June 1999)
3. Jini Technology Core Platform Specification Version1.2 (December 2001)
4. Salutation Architecture 2.1, Salutation Consortium, Specification (1999)
5. Specification of the Bluetooth System, Bluetooth Forum, Specification (February 2001)
6. Bettstetter, C., Renner, C.: A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In: Proceedings of the EUNICE Open European Summer School, Twente, Netherlands (September 2000)
7. Marin-Perianu, R., Hartel, P., Scholten, H.: A Classification of Service Discovery Protocols. Technical Report TR-CTIT-05-25, Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625 (June 2005)
8. Helal, S.: Standards for Service Discovery and Delivery. IEEE Pervasive Computing 1(3), 95–100 (2002)
9. Zhu, F., Mutka, M.W., Ni, L.M.: Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing 4(4), 81–90 (2005)
10. Richard, G.G.: Service Advertisement and Discovery: Enabling Universal Device Cooperation. IEEE Internet Computing 4(5) (2000)
11. Koponen, T., Virtanen, T.: A Service Discovery: A Service Broker Approach. In: Proceedings of the 37th Annual Hawaii international Conference on System Sciences (Hicss 2004) - Track 9, January 05-08, 2004, vol. 9, IEEE Computer Society, Washington, DC (2004)
12. Dobrev, P., Famolari, D., Kurzke, C., Miller, B.A.: Device and service discovery in home networks with OSGi. IEEE Communications Magazine 40(8), 86–92 (2002)
13. Grace, P., Blair, G.S., Samuel, S.: A reflective framework for discovery and interaction in heterogeneous mobile environments. SIGMOBILE Mob. Comput. Commun. Rev. 9(1) (2005)
14. Bromberg, Y.D., Issarny, V.: INDISS: Interoperable Discovery System for Networked Services. In: Alonso, G. (ed.) Middleware 2005. LNCS, vol. 3790, Springer, Heidelberg (2005)
15. Raverdy, P.G., Issarny, V., Chibout, R., de La Chapelle, A.: A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments. In: Proceedings of MOBIQUITOUS – The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services, San Jose, CA, USA (July 2006)
16. Kang, S., Ryu, S., Kim, N., Lee, Y., Lee, D., Moon, K.: An Architecture for Interoperability of Service Discovery Protocols Using Dynamic Service Proxies. In: Kim, C. (ed.) ICOIN 2005. LNCS, vol. 3391, pp. 786–795. Springer, Heidelberg (2005)

17. Flores-Cortés, C.A., Blair, G.S., Grace, P.: A multi-protocol framework for ad-hoc service discovery. In: Proceedings of the 4th international Workshop on Middleware For Pervasive and Ad-Hoc Computing (MPAC 2006), Melbourne, Australia, November 27 - December 01, 2006, vol. 182, ACM Press, New York (2006)
18. Yu, M., Taleb-Bendiab, A., Reilly, D., Omar, W.: Multi-Standard Service Interoperation Protocol Through Polyarchical Middleware. In: Processing of 4th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet2003), Liverpool, U.K. (June 2003)
19. Kang, S.H., Ryu, S., Kim, N., Lee, Y., Lee, D., Moon, K.D.: An Architecture for Interoperability of Service Discovery Protocols Using Dynamic Service Proxies. In: Kim, C. (ed.) ICOIN 2005. LNCS, vol. 3391, pp. 786–795. Springer, Heidelberg (2005)
20. http://java.sun.com/developer/products/jini/installation_jini1_2_1.html
21. http://jslp.sourceforge.net/

U-Interactive: A Middleware for Ubiquitous Fashionable Computer to Interact with the Ubiquitous Environment by Gestures

Gyudong Shim, SangKwon Moon, Yong Song, Jaesub Kim, and Kyu Ho Park

Computer Engineering Research Laboratory,
Department of Electrical Engineering and Computer Science,
Korea Advanced Institute of Science and Technology
{gdshim,skmoon,ysong,jskim}@core.kaist.ac.kr, kpark@ee.kaist.ac.kr

Abstract. In this paper we present a system, called U-interactive, that provides spontaneous interactions between human and surrounding objects in heterogenous ubiquitous computing environments. Our U-interactive system introduces a virtual map, which contains interactive objects around a user in each ubiquitous environment. In the virtual map, each interactive object is tagged with geographic information and attributes to interact with. Each user can create interactive objects in the virtual map corresponding to physical objects. Also the scope of the map is automatically adjusted according to user's location (inside building or outdoors) by location services. U-interactive system runs on both mobile devices and infrastructures. U-interactive system provides interoperability with communication methods, such as UbiSpace, UPnP, and Web services. We developed our U-interactive system upon a prototype of ubiquitous environment. U-interactive system contains interactive kiosk, printer, sensor networks, and users.

Keywords: Middleware, Ubiquitous Fashionable Computer, HCI, *iThrow*, Spontaneous Interaction, Service discovery, File Sharing.

1 Introduction

Many researches have been come out to realize ubiquitous computing environments in the real world. The common philosophy of the researches is to make a convenient life with surrounding computers. People in the ubiquitous environment access any information by multiple interfaces and displays. It is important to provide easy and comfortable human interfaces in the ubiquitous environments.

As to realize ubiquitous environments, our research project teams have developed testbed on KAIST campus called U-TOPIA and a wearable computers named Ubiquitous Fashionable Computer(UFC)[1]. In order to realize U-TOPIA, we are elaborating the campus-wide infrastructures in KAIST campus. U-TOPIA has been equipped with indoor and outdoor testbed for location services which consist of ZigBee and UWB[11] sensor networks. In addition,

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 694–705, 2007.

Wireless Mesh networks in the campus extends WLAN coverage to the outdoor areas [8].

UFC has been developed as a wearable gadget and cloth as shown in Fig.1 for convenience and portability. In order to interact with the environments user-friendly, UFC has a gesture recognition device called *iThrow* [2]. *iThrow* is a ring typed intuitive interface device. *iThrow* has a 3D accelerometer and a 3D magneto-resistive sensor; thus it can recognize specific commands and two dimensional pointing direction of finger from user gestures.

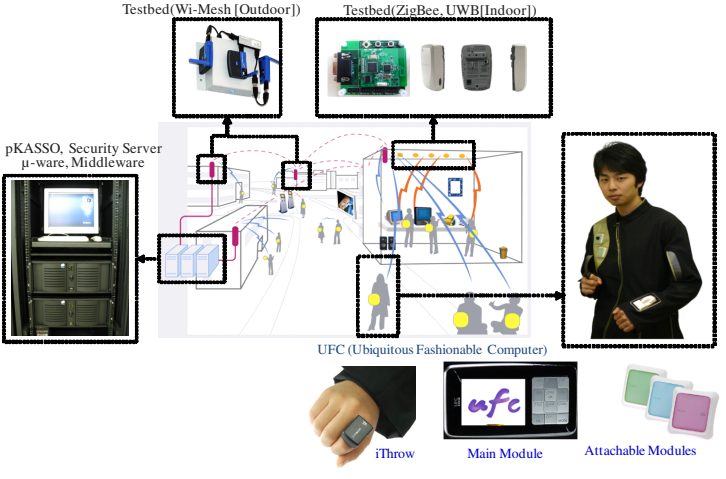


Fig. 1. U-TOPIA architecture and UFC components

A UFC user can select a target inside the testbed by finger pointing as shown in Fig.2-(a). Then the selected target is represented in the display of UFC. After the user selects a target in the U-TOPIA, the user can throw data such as a image file to the target as shown in Fig. 2-(c). Through intuitive gestures, users can interact with objects such as throwing a file, receiving a file, controlling devices, and querying some information of the target. For example, when a user throws a file to a printer, the printer will print the file.

We designed an interactive system, called U-interactive, that provides spontaneous interaction between UFC users and U-TOPIA. U-interactive system provides followings.

- location detection from multiple sources
- target selection from the virtual map
- interaction method and command matching upon an interactive object
- communication between UFC and interactive objects

U-interactive system introduces virtual maps and interactive objects. A *virtual map* contains interactive objects around a user in a specific ubiquitous environment. The scope of the virtual map is automatically adjusted upon user's

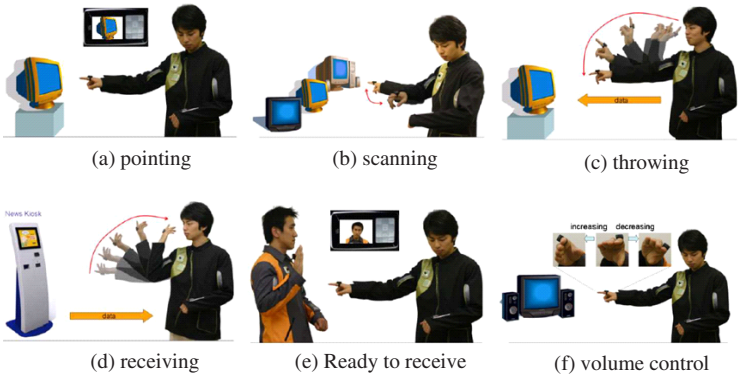
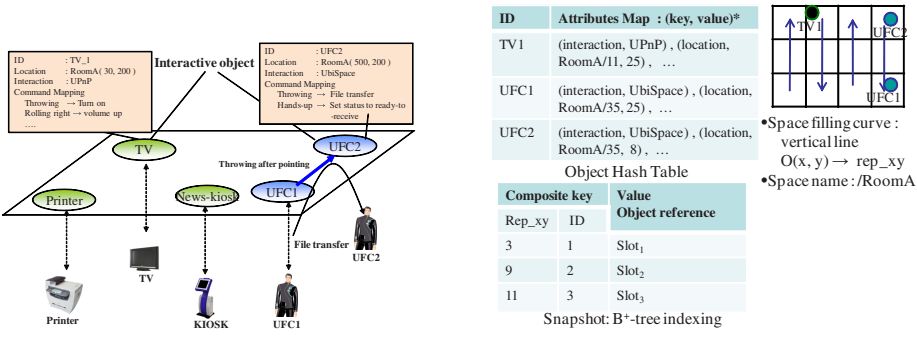


Fig. 2. Gesture operation sets of *iThrow*



(a) concept (b) data structures

Fig. 3. Virtual map and interactive objects

location(inside building or outdoors). In the virtual map, each interactive object is tagged with geographic information and attributes to interact with. An interactive object is an abstraction unit of physical object on the virtual map. Interactive object contains location, and service attributes to interact with. Fig.3 shows the virtual map and interactive objects. In Fig.3, the UFC1 has information about UFC2's location, interaction method, and the command configuration that assigns throwing motion as transferring a file.

To support such interactions at indoor areas and outdoor areas, we designed and implemented U-interactive system. U-interactive system is a customized and evolved incarnation of μ-ware for U-TOPIA where thousands of clients and hundreds of infrastructures coexist. U-interactive system aims spontaneous interaction via HCI, especially *iThrow*. U-interactive includes coordination of location service, automatic virtual map resolution, and interaction through multiple interfaces and commands.

The remainder of this paper is organized as follows: Section 2 presents related works. Section 3 describes U-TOPIA and U-interactive system architecture. Section 4 specifies interactive object management method, and Section 5 describes interaction methods such as UbiSpace and UPnP. Section 6 shows implementation status. Finally we conclude in Section 7.

2 Related Works

Gesture control has been studied by two categories, camera-based and movement sensor-based[5]. Juha at el. presented accelerometer-based gesture recognition method. They developed the gesture control system for design environment especially for TV, VCR, and lighting. Our *iThrow* device is similar to it in movement sensor-based, but our gesture sets are designed for adaptability and mobility of users. So UFC users can probe and choose any items in a given environment.

The concept of virtual map is analogous to the augmented reality. But we focus on the interactions and efficient target selection mechanism in a specific scope. In Virtual Information Towers(VIT)[15], mobile users can interact with visible items on the advertising columns. VIT provides a metaphor to access information which is assigned to physical location. VIT does not provide the discovery of VIT-Directory which has a list of VITs. So VIT system can not provide seamless service and the user should manually set the environment's server. In addition, the user interacts with VIT by web-browser interface of the wearable device.

Tatsuo at el. developed *personal home server* to support spontaneous interaction in home computing environments [16]. The mobile device works as a personal home server which contains personal information and preferences to the services. They utilize inter-operable protocol SOAP for interaction methods. But U-interactive exploits UbiSpace for easy file transfer between devices and services. They aims personalized and secure interaction by RFID tags which store encryption keys. They designed and implemented the system for home networks rather than outdoor or large scale networks. Since the services are discovered by SSDP of UPnP, The system is not scalable for large number of services.

Personal Server[14] is a mobile device that contains personal data which can be accessed through surrounding displays and keyboards input systems such as public kiosk or PC monitor and keyboards. Personal Server contains an embedded web server in the mobile device. The discovery of the host is performed by Bluetooth and UPnP. Actual interactions between the Personal Server and the host are performed by HTTP and SOAP[13] protocol. However Personal Server provides only limited interfaces from the infrastructures.

3 U-Interactive System Architecture

U-interactive is a middleware component for ubiquitous interactive services which can be controlled with a gesture device- *iThrow*. The device of ubiquitous

services consist of various devices such as kiosks and printers. The software on the devices are built on our middleware framework. The software architecture of our middleware is shown in Fig.4. Most of the interactive services are implemented as service bundles over Open Service Gateway initiative(OSGi) Platform[10]. In order to provide location based services, we proposed a ubiquitous context aware middleware framework for campus-wide infrastructures and UFC in μ-ware[3].

μ-ware is a middleware framework that adapts the runtime environment upon the location of the client. μ-ware provides JAVA based extensible and configurable runtime environment to the applications. μ-ware is composed of following components.

- KAIST Ubiquitous Service Platform(KUSP) : OSGi based runtime environments. This framework manages the life time of service bundles.
- USD Protocol: light-weight service discovery protocol. It can discover services across the network by multicast.
- UbiSpace: Coordination component like tuple space. It handles publish/ subscribe operation among UFCs.
- Instant Service Loader: it downloads a service bundle from the servers, after that it installs and executes the bundle.

By these components μ-ware not only gives an efficient computing environment to upper applications on UFC but helps the applications to exploit various ubiquitous service modules. [3]. In the previous work, the main target environments of μ-ware were indoor smart spaces. To be a foundational middleware framework for U-TOPIA, μ-ware has to be enhanced and customized in pursuit of UFC's energy efficiency and interactive functionality even for massive clients in outdoor areas. In this chapter we describe U-TOPIA environments, internal architecture of U-interactive, and coordination of location service.

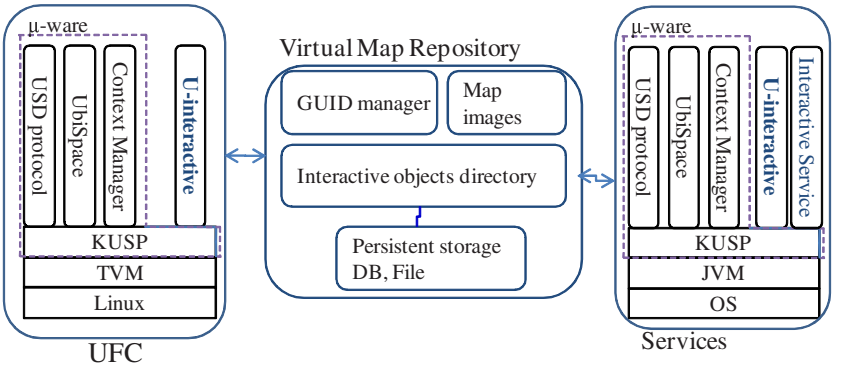


Fig. 4. U-interactive system architecture

3.1 U-Interactive Infrastructures on the U-TOPIA

U-TOPIA equips infra structures such as location service, service discovery, and service provision servers. The servers are constructed with multi-level tree hierarchy for campus wide ubiquitous services. The entire space is divided by sections of the institute, building, floor of building, and room or hall. In order to provide information repository and support collaboration storage, we suggest virtual spaces which are constructed in a hierarchical tree. Each representative space holds the map and the service list of the region. Physical objects can be assigned by attributes of services. And they are referenced by a GUID (Global unique identification). For the management reasons, the GUID manager exists in the root of hierarchy tree. Objects directory contains a table of GUID, name, type, service reference, and information to represent image files, video files, and HTML files. The temporarily available services and location of users are separated from static information in the object directory. File repository provides a storage for the system and makes it possible to share files between UFCs.

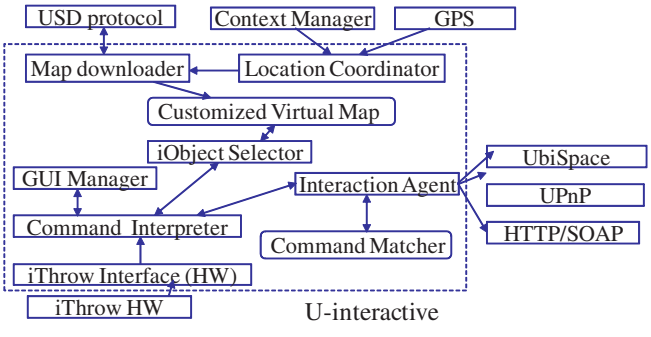


Fig. 5. U-interactive internal architecture: U-interactive cooperates *iThrow*, GPS and infrastructure services - UbiSpace, Context Manager, an USD protocol

3.2 Internal Architecture of U-Interactive

U-interactive internal architecture is shown in Fig.5. U-interactive utilizes foundational services such as Context Manager, UbiSpace, USD protocol of μ-ware. Main roles of U-interactive are virtual map management and interaction between UFC and interactive objects. The virtual map is automatically downloaded from the map server by location coordinator when the logical address of UFC is changed. Command Interpreter takes the command and pointing directions from *iThrow* hardware. Command Interpreter forwards the command to *iObject* Selector or Interaction Agent by the type of *ithrow* command. Interaction Agent takes the interaction by the interactive object type. Interaction Agent decides a proper interface to the interactive object from command matcher which has *ithrow* command to iObject service mapping.

3.3 Coordination of Location Service

Location is described by longitude, latitude, altitude, and logical address. Logical address consists of organization, building, floor, room number in hierarchical order, for example 'KAIST/E3-2/3F/Room3217', 'KAIST/Section1/'. UFC users can resolve their locations by multiple sources such as GPS, ZigBee sensor network, and UWB sensor networks depending on the place. When UFC users are located in an indoor area such as a room or a floor, their locations are resolved from UWB or ZigBee sensor networks. But if they are located in outdoor areas such as campus road or square, their locations are detected basically by GPS on UFC. Because GPS has inaccuracy in densely surrounded buildings, ZigBee beacons of the testbed is used as a hint for more accurate location. The beacon of Zigbee sensor network holds the longitude, latitude, and logical place description. The beacon signal is proximity guarantee since the receiver is within a transmission range from the landmark. If there are more than two beacons from different positions, Received Signal Strength Indication(RSSI) is used on the triangulation. U-interactive coordinates the multiple location sources adaptively based on predefined source priority(UWB>ZigBee>GPS). When GPS is the only available location source, it is required reverse geocoding to translate GPS location to the logical address.

4 Management of Interactive Object

U-interactive assigns physical objects to service and information to construct smart environments. For instance, electrical devices can contains control service as UPnP service and bus stop can have bus time schedule information. Complex building objects can construct logical service hierarchy. As the preceding step to interactions with smart object in the ubiquitous environments, U-interactive assigns GUID(Global Unique Identification) for each objects. Object are grouped by their information and service source type. Geographical information can be handled by organizations' administrator and useful information can be added by users later. *GUID Manager* in our system handles issuing of identifications. GUID has prefix of geographic information and sequential serial number for services. GUID could be URI or bar code or RFID depends on the object. Each UFC has unique PANDA ID(16 bytes) for recognition and authentication.

4.1 Interactive Object Registration

We designed virtual map data structures as shown in Fig.3-(b). For fast spatial queries the repository maintains a current snapshot of the interactive objects. The location of interactive object is converted one dimension representative value by vertical line space filling curve. The snapshot is built by B^+-index whose key is assigned by the converted value and object id. Detail interactive object information is maintained in an attribute map hash table. When a new object is registered, a new element is inserted in the object hash table and the location is inserted the snapshot.

Our services on the KUSP platform can register themselves in the current level of their location repository. Since we expect most of services are ported in OSGi compatible bundle, they can register and deregister by bundle start/stop phase. As a representative location of the single KUSP platform, *Context Manager* has default location of the platform. Default location is assigned if the service doesn't specify the location. Context manager takes charge of the service registration of other service bundles. Holding only information objects such as building information are managed by a central administrative way(such as institute administrative building) or fully decentralized way(each home appliances by user administration). Geographic objects such as building, statue and pond are stored in the Map server conjunction with Geographic Information System.

4.2 Interactive Object Discovery

When UFC detects logical address change, the virtual map repository is resolved by logical address key lookup in USD protocol[4]. After the infrastructures are spread out over the same network and the number of users' discovery requests increases, the scalability of entire system become worse because of indirect multicast search algorithm of UPnP[12]. Therefore we add caching mechanism of service discovery results based on the repository of UFC and infrastructures. The global infrastructure topology of an institute is maintained by a hierarchical structure. The virtual map repository stores topology of the infrastructures in a global view, the discovery of repository from UFC can be resolved readily by a close repository. When UFC users move around the logical place, they can adapt their location and intention by instant downloading of information of the space such as addresses of map servers, available file server.

4.3 Target Selection in a Virtual Map

Basically U-interactive sets the space boundary by logical address scope to UFC. UFC can see the target in the given boundary space with objects with priorities. The public and hot services are ranked high priority by usage count and temporal and moving objects are ranked low priority by reduction of movement. This priority helps user to select clustered targets in narrow directions. UFC user can select a target object in user friendly manner by the scanning gesture of *iThrow*. Yoo et al, proposed target selection algorithm which is ray based minimum angle difference and adaptive angle placement of targets [2]. We adopt ray based minimum difference angle target selection of [2].

When a user in the smart room alone, most of object are stationary and he keep stationary for a second or more than an hour. Then the objects position and status are static information. So as to reduce repeated location and object query to the space repository, UFC selects the target object in own repository on the stationary state. To simplify target selection operation, the space objects are modeled as two dimensional points. the origin is the location of UFC and each

object calculated in polar coordination system. If the UFC location is (x_1, y_1) and the target is (x_2, y_2) in cartesian, the polar coordination is given by

$$(r, \theta) = (\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1})). \quad (1)$$

As long as the UFC is stationary with a given boundary D_{update}, the calculated polar coordination positions are reused to reduce complex calculation of *sqrt* and tan^{-1} operation in the UFC. If a new target scanning is performed at the same position, target object is selected by lookup of given angle difference of pointing direction in the list of objects sorted by angle and distance.

5 Interaction Methods

U-interactive interacts with interactive objects by multiple interface methods to various services. The interaction methods are dependant on the interactive object interfaces. UPnP devices are interacted by UPnP protocol and Ubiquitous interactive services are communicated by UbiSpace. For the web services, U-interactive handles commands by SOAP.

5.1 UbiSpace

UbiSpace handles specific commands and file sharing of interactions. In order to reduce the burden of application development in the data sharing, we choose UbiSpace as a collaborative data sharing service for file and java objects. UbiSpace provides publish/subscribe event handling system. Furthermore it has repositories of file sharing in the infrastructures. Each object contains a key to identify and it is indexed by B^+-tree for fast look-up. Each client can subscribe data for specified the key. UbiSpace derives tuple spaces such as TSpaces[7] and JavaSpaces[9] in central temporal and time decoupling aspects, but it simplifies object matching by name key, and supports file publish/subscription operation for file sharing between clients. The only basic operations are as shown table 1 for limitation of the space.

To support basic simple file transfer, we integrated same API while UbiSpace inspects the object whether it is a file. The source file is uploaded the file repository of server and after that subscribers download the file later by *subscribe* or *read* operation. Because of large file transfer overhead, UbiSpace

Table 1. UbiSpace basic APIs

long insert(String key, java.io.Serializable obj)
Object take (String key)
Object read (String key)
ITuple delete (long itemID)
long publish(java.lang.String key, java.io.Serializable obj)
long subscribe(java.lang.String key, EventHandler callback)
void unsubscribe (long seqNumber)

exploits network file system to clients. When a UFC joins the UbiSpace server, it automatically mounts the file system on a given private directory. Large file operations can be exploited by native distributed file system operations. Most services are accessible anyone in the place but some administrative information and restrictive services should be protected from unauthorized access. By using publish/subscribe data access control of UbiSpace, it can release the burden of access control list management on the applications.

6 Implementation and Experiment

The testbed and terminal team of our project implemented the UFC show room. We implemented the indoor U-interactive prototype on the showroom. There are UWB and ZigBee sensor networks for location service. The location is accurate on limited region up to 15*cm* resolution. The outdoor virtual spaces are under development but they would be installed in months.

As U-services examples, U-Desktop and U-Print service in the room are implemented by OSGi bundles. The U-Desktop service displays multimedia file on public kiosk by subscription of UFC users in the space. The publish and subscription of the images and movie files are performed by UbiSpace *publish*, *subscribe* operations both of U-service and UFC. U-print service provides UFC with customized content generation by UFC's throwing image files.

The U-Kiosk and U-Print services are implemented as service bundles of OSGi. To exploit Windows native services and applications, the service bundles connect dynamic loadable libraries by JNI interface to display and print a file by *Execute* function of Windows MFC. As an example codes, the U-Print service can be easily and concisely implemented as follows.

```
int printSubscriptionID = us.subscribe(this.FILE_KEY + serviceID,
new EventHandler() {
    public void todo(String key, Serializable obj) {
        if (obj instanceof File) {
            File file = (File)obj;
            // JNI interface for Windows Print service
            userviceAdaptor.print_process(file);
        }
    }} );
}
```

The gesture operation sets are shown in Fig.2. File sharing operation (c),(d) in Fig.2 is performed by UbiSpace publish/subscribe API also. The rest operations - pointing, scanning, volume control are performed by publish/subscribe of *IThrowCommand* object which is published by the UFC.

The performance of UbiSpace is experimented by latency comparison with T-Space. The latency between the server and the client(PC) is examined when there are 8000 tuples in the server. The tuple object size is 1KBytes. Fig.6 shows that our UbiSpace outperforms T-Space for the best case 111% in the

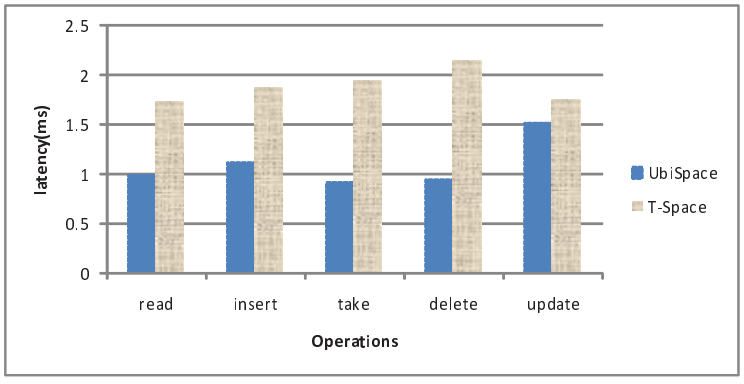


Fig. 6. UbiSpace latency comparison with T-Space

take operation. The reason of fast response comes from efficient tuple indexing by B^+-tree and setting no TCP delay on the socket. The tuples are indexed by a composite key which consists of the tuple name, the tuple field size, and fields.

7 Conclusions

U-interactive system make it possible for UFC users to interact with interactive objects in ubiquitous environments by intuitive gestures. We propose new concepts, *virtual map* and *interactive objects*. The virtual map scope is automatically resolved by location of a user to be convenient. U-interactive system provides interoperability by choosing a interaction method adaptively, such as UPnP, SOAP, and UbiSpace. U-interactive provides learnable and customizable mechanism to adapt various interactive object types. The standard of command sets is required to be general and learnable interfaces for *iThrow*. It is crucial for the system to tag the location on any physical objects. We are plan to integrate multiple HCIs in U-interactive such as speech recognition and small keypads.

References

1. Lee, J., Lim, S.-H., Yoo, J.-W., Park, K.-W., Choi, H.-J., Park, K.H.: A Ubiquitous Fashionable Computer with an i-Throw Device on a Location-based Service Environment. In: PCAC 2007 (2007)
2. Yoo, J.W., Jeong, Y.W., Song, Y., Lee, J.P., Lim, S.H., Park, K.W., Park, K.H.: iThrow: A New gesture-based wearable input device with target selection algorithm. In: ICMLC 2007 (to be appeared, 2007)
3. Song, Y., Moon, S.K., Shim, G.D., Park, D.Y.: A Middleware Framework for Wearable Computer and Ubiquitous Computing Environment. In: PercomW 2007, pp. 455–460 (2007)
4. Moon, S., Kim, J., Park, D.: USD Protocol: Ubiquitous Service Discovery Protocol on Infrastructure-based architecture for Ubiquitous Fashionable Computer. In: MUE 2007, pp. 779–784 (2007)

5. Kela, J., Korpipaa, P., Mantyjarvi, J., Kallio, S., Savino, G., Jozzo, L., Marca, D.: Accelerometer-based gesture control for a design environment. Personal Ubiquitous Comput. 10(5), 285–299 (2006)
6. Park, K.-W., Lim, S.S., Seok, H.C., Park, K.H.: Ultra-Low-Power Security Card, PANDA for PKI-based Authentication and Ubiquitous Services. In: Proceedings of Confer-ence on Next Generation Computing, pp. 367–373 (November 2006)
7. Lehman, T.J., Cozzi, A., Xiong, Y., Gottschalk, J., Vasudevan, V., Landis, S., Davis, P., Khavar, B., Bowman, P.: Hitting the distributed computing sweet spot with TSpaces. Computing Networks, 457–472 (2001)
8. KAIST UFC Project, http://core.kaist.ac.kr/UFC
9. Freeman, E., Arnold, K., Hupfer, S.: JavaSpaces Principles, Patterns, and Practice (1999)
10. OSGi alliance, http://www.osgi.org
11. UbiSense, http://www.ubisense.net
12. UPnP Forum, http://www.upnp.org
13. SOAP:Simple Object Access Protocol, http://www.w3c.org/2002/ws
14. Want, R., Pering, T., Danneels, G., Kumar 1, M., Sundar, M., Light, J.: Changing the Way We Think about Ubiquitous Computing. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, Springer, Heidelberg (2002)
15. Leonhardi, A., Kubach, U., Rothermel, K., Fritz, A.: Virtual Information Towers - A Metaphor for Intuitive, Location-Aware Information Access in a Mobile Environment. In: ISWC 1999 (1999)
16. Nakajima, T., Satoh: A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. Personal Ubiquitous Comput. 10(6), 379–391 (2006)

Towards Context-Awareness in Ubiquitous Computing

Edwin J.Y. Wei and Alvin T.S. Chan

Department of Computing, The Hong Kong Polytechnic University
{csjwei,cstschan}@comp.polyu.edu.hk

Abstract. Future ubiquitous computing has accelerated the need of context-awareness that leverages information about surrounding situation so as to adapt applications. There is considerable interest in context-awareness, and many prototypes have been proposed, which have demonstrated the potential of context-aware applications. That notwithstanding, these kinds of systems are known to be difficult to design, develop and maintain. This paper considers these difficulties as it discusses the core issues of context-aware computing, including definition of context, techniques of acquiring, modeling and adapting to contextual information. It intends to provide the community with a comprehensive and detailed view of current state of the art.

Keywords: Context-Awareness, Ubiquitous Computing, Context Definition, Context Acquisition, Context Modeling, Context-aware Adaptation.

1 Introduction

The need for context-awareness, which leverages information about surrounding situation so as to adapt applications, has been accelerated by the vision of ubiquitous computing. Computing devices in ubiquitous computing environment now exhibit a high degree of mobility and their computational systems must adapt to heterogeneous and dynamic surrounding environment where they are within. At the same time, more and more everyday devices, such as digital cameras and watches, are now equipped with computing capabilities, so that applications in ubiquitous computing environment need to take into account the attributes of different devices, which otherwise will result in unsatisfactory user experience.

Due to the interest of context-awareness, many research works have been proposed in this area [1] [2] [3] [4] [5]. These prototypes have demonstrated the potential of context-aware applications, but have also shown that designing, developing and maintaining these kinds of systems are still extremely difficult, to say the least. Lots of technical challenges remain to be addressed before even simple context-aware systems can be widely and realistically developed. In particular, every context-aware application needs to consider four basic issues: what is context, how to acquire them, how to represent them, and how to adapt to them. This paper discusses these core issues of context-aware computing, and

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 706–717, 2007.

intends to provide the community with a comprehensive and detailed view of current state of the art.

The remainder of this paper is organized as follows: In section 2, we review definitions of context and present our own definition from an application point of view. Section 3 focuses on currently available context acquisition techniques for context-aware applications. Section 4 discusses several basic technical aspects of context models including data structure, integrity and manipulation. In section 5, we detail various design concerns of context-aware adaptation. Section 6 concludes and summarizes related challenges in context-aware research.

2 Definitions of Context

In order to effectively utilize contexts, we should first understand what contexts are. Researchers in the context-aware computing community have invariably offered their own definitions of context based on their research background. Schilit and Heimer [6] first introduced context-aware computing in 1994 and set three parameters for contexts: the software's location of use, the collection of nearby people and objects, and changes to those objects over time. For a long time, contexts are defined by enumerating examples or choosing synonyms [7] [8] [9] [4]. In 2000, Dey and Abowd [10] proposed a more generic definition:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

More recently, inspired by social sciences, some researchers argue that contexts are socially and psychologically constructed outcomes of human activities rather than stable, and objective sets of features that externally characterize activities [11] [12]. Oulasvirta *et al.* [13] summarize these definitions of context, and divide them into two camps which are realism and constructivism. Realism posits contexts as existing ontologically, and that they can be correctly recognized and adapted to if properly instrumented and programmed. Whereas, constructivism recognizes that contexts are human creations, social and psychological, and that people should be provided resources to create and maintain these contexts.

No matter realism or constructivism, in their answers to "what is context", user interaction with applications is overly emphasized. Actually, many non-interactive programs can also make use of surrounding situation. We argue that considering the issue of context definition from an application point of view will be a more reasonable way to help developers to construct context-aware applications. With this in mind, we present here our own definition of context:

Context is application specific, and context of an application is any external information which can be utilized to adapt the data, behavior or structure of this application.

Our definition of context indicates three important features of context. First of all, context is application specific. Contexts of one specific application may make

no sense to others. Secondly, context is external to applications. Only information outside one application can be regarded as the context of that application. Finally, context can be used to adapt not only behavior of applications, but also their data, or even structures.

3 Acquiring Contexts

In practice, the information from three categories of context may be used to adapt applications, which are physical contexts, computing contexts and user contexts. Physical contexts are physical circumstances of an application like noise level and temperature. Computing contexts refer to an application's own execution conditions such as host computing resources, available peripheral devices, network capacity and connectivity, and so on. Finally, user contexts are anything regarding users such as their location, presence, identities, abilities, activities, etc. The vision of future ubiquitous computing paradigm requires these contexts to be captured without user interaction. In this section, we review various techniques that have been used to implicitly acquire these contextual information.

3.1 Acquiring Physical Contexts

Physical contexts are primarily obtained through specially designed physical sensors, which convert physical properties or phenomena, such as light and noise, into a corresponding measurable data. For example, the TEA project [4] uses photodiode, accelerometers, and other sensors to measure various contextual information such as light level, tilt and vibration, proximity of humans or other heat-generating objects, and so on. Further sensor progress in the development and manufacture of sensors will allow more and greater varieties of physical contextual information to be captured and used. Table 1 provides a summary of some common types of sensors for context-aware applications.

3.2 Acquiring Computing Contexts

Computing contexts are normally collected by software routines. Most mainstream operating systems provide primitives for developers to get related runtime information about the device hardware, and reduce the work involved in development. For example, in most Unix/Linux operating systems, one can employ commands like iostat, vmstat, and netstat to report statistics for I/O devices, virtual memory, and network condition. Symbian OS also provides a comprehensive software development toolkit for developers to monitor CPU, memory and storage usage, etc. In addition to using OS primitives, it is also possible to leverage user-level modules to sense computing contexts. For instance, applications can obtain device management information of most network devices like routers and firewalls by sending SNMP requests, and applications base on [14] or [15] can be notified of change of network bandwidth via upcalls.

Table 1. Common Types of Sensors

Type of Sensor	Context Sensed	Sensor Examples
Temperature Sensor	Temperature	Thermometer, Thermocouple, Thermistor
Heat Sensor	Heat	Bolometer, Calorimeter
Magnetism Sensor	Orientation	Magnetic Compass, Fluxgate, Compass
Pressure Sensor	Altitude, Atmospheric Pressure, Speed	Altimeter, Barometer
Gas/Liquid Flow Sensor	Velocity of the Wind, Rate of Fluid Flow	Anemometer, Mass Flow Sensor
Mechanical Sensor	Acceleration, Position, Angle, Deformation	Acceleration Sensor, Position Sensor, Selsyn
Chemical Sensor	Proportion of Gas	Carbon Monoxide Detector, Ion-Selective Electrode
Light Sensor	Light	Phototubes, Photodiode
Sound Sensor	Audio	Microphone, Hydrophone, Seismometer
Motion Sensor	Speed, Acceleration	Radar Gun, Speedometer, Odometer
Orientation Sensor	Orientation	Gyroscope, Artificial Horizon

3.3 Acquiring User Contexts

It is difficult to directly sense user context through dedicated hardware sensors or software routines. Rather, they have to be derived from original measurements by software programs where original measurements are processed collectively. In the followings, we discuss in some detail on the techniques used to acquire user contexts especially location information.

Location. The most commonly acquired attribute of user context is location. In this section, we describe four most frequently used techniques for the acquisition of location information, which are trigonometry, signature matching, cellular proximity and computer vision.

Trigonometry, including trilateration and triangulation, leverages the geometry of triangles to determine the positions of objects. Trilateration utilizes the measured distance between the subject and three or more reference points, as well as the known locations of these reference points, to compute the subject's location. Differently, triangulation uses angle measurements and at least one known distance to complete the computation. In order to measure the required distances and angles for trigonometry, time of arrival/time difference of arrival (TOA/TDOA) [16] [17], received signal strength (RSS) [18] and angle of arrival (AOA) [19] of various communicational signals are most frequently used. Trigonometric approaches are fine-grained localization techniques, frequently used outdoors. However, due to rough wall surfaces and obstacles

between emitters and receivers, communication signal propagation in indoor environment suffers from multipath, non-line-of-sight (NLOS), and local shadowing, which result in unreliable measurements of location metrics such as TOA, TDOA and AOA. Therefore trigonometric approaches fail to provide adequate location accuracy indoors.

The positioning process based on signature matching consists of two phases: an off-line phase of collecting data, and a real-time phase of inferring the users' location [20]. In the off-line phase, necessary information of the entire zone of interest is collected to produce signatures, and the latter are then stored as a function of user's location. In the real-time phase, incoming data is analyzed and compiled into a unique signature which is compared with the recorded signatures to identify the closest record and then the location is inferred. In order to produce a unique signature for each location, several types of communicational signal information such as received signal strength (RSS), signal noise ratio (SNR), angular power profile (APP) and power delay profile (PDP) can be utilized. Another interesting information frequently used to construct distinct signatures is the ground reaction force (GRF) gathered by pressure sensors. GRF refers to the reaction force supplied by the ground in response to the weight and inertia of a body exerted on the ground. For example, the Smart Floor [21] sets load cells under floor tiles to gather GRF profile, and choose ten profile features, including the mean value, standard deviation, length of the profile and so on, to use as signatures for each GRF profile. Signature matching approaches can effectively counteract the problems of signal propagation in indoor environment. The major drawback of these approaches is that developers have to collect a great quantity of data to generate the signature database. Furthermore, changes to the environment may require reconstruction of the predefined dataset or retrieval of an entirely new dataset. Consequently, it is not suited for ad hoc deployment scenarios [22].

Cellular proximity location sensing techniques determine the location of a subject when it is near a known access point. In a cellular network, each fixed access point, with known location, owns its sensing cell. Whenever the subject enters the cell, it is sensed by the access point, and its location is therefore pinned down to the resolution of a cell. Cellular proximity approaches can be used both indoors and outdoors, and also requires no collection of off-line data. However, they are coarse-grained localization techniques. Since the subject's location information is sensed by judging whether the subject is in the range of an identified area, the sensing accuracy is determined by the radius of the identified area. Moreover, they also incur significant installation and maintenance costs. Using cellular proximity techniques, the cellular network must provide thorough coverage through adequate placement and density of access points.

Location information can be also derived from analysis of data from visual images. Visual processing techniques like depth and color segmentation, blink detection, and color histogram matching can be used to analyze visual streams from cameras, and recognize one or several objects, together with their 2D positions in the image or 3D positions in the scene. By combining these positions

with knowledge of camera's relative location, fields of view, and heuristics on the movements of objects, the final location of objects can be computed. For instance, Microsoft's Easyliving [23] uses two color stereo cameras each connected to a PC to track multiple people in a living room. Vision based location sensing techniques are the most flexible approaches. They can be used either indoors or outdoors, and do not require any sort of devices to be worn by users. However, as scene complexity increases and more occlusive motions occur, more works have to be done to maintain analysis accuracy [24].

Other User Contexts. Other user contexts can also be acquired via a number of novel ways. MIT's Office Assistant [25] uses pressure sensors to detect visitors. Schmidt *et al.* [26] make use of load sensing technique to explore more pervasive augmentation of surfaces in everyday environment including floors, tables and other high interactive spaces, and from these load-sensitive surfaces extract three context primitives: weight, position and type of interaction. Moore *et al.* [27] measure image-, object-, and action-based information from videos to recognize human activities such as reading, coffee break and washing dishes, and objects like winding road and parking car.

4 Modeling Context

Context modeling is concerned with representing, structuring and organizing contextual data and relationships between them, in order to facilitate the storage and operations of them. A well-designed context model needs to consider three basic technical aspects: data structure, integrity and manipulation.

4.1 Data Structure

The underlying data structure used to exchange context information inside and between applications is the first basic issue for context models. An appropriate data structure for contextual information will facilitate not only the representation of contextual data, but also their storage, validation, modification, retrieval, and even reasoning. Currently, tuples, objects and markup schemes are three most popular data structures used to represent contextual information.

The simplest structure to represent contextual information is key-value pair (2-tuple). Every pair describes one aspect of the surrounding situation of an application. A set of these pairs describes the whole environment where applications are actively deployed. Early works in context-awareness, [28] for instance, frequently used tuples as the underlying context structures. Tuples are easy to implement and manage. Most programming languages provide direct support to construct tuples. For example, as a fundamental data type, Lisp provides list, which is a finite ordered sequence of elements. Eiffel also has a built-in type of tuple. However, tuples lack structure and formality. As a result, they can not effectively express sophisticated contextual information and relationships.

Contexts can also be modeled as a set of related objects. Contextual information is embedded as the states of these objects, accessed and modified through

accessor and mutator methods. Object-oriented context models' encapsulation and reusability cover parts of the problems arising from the dynamics of contexts [29]. The major drawback of object-oriented models is that context objects are programming language dependent, which will affect their portability among different applications and platforms. Although there do exist some techniques advocating language- and platform- independent implementation, in this area much work remains to be done.

Various markup languages can be also used to model context data, which use a hierarchical data structure consisting of markup tags with attributes and content. Among them, XML schema languages, like DTD, XML Schema and RELAX NG, are the simplest ones. We can also model contexts using other more expressive and formal data representation markup languages. Resource Description Framework (RDF) is such an alternative. Using RDF, contexts are modeled as a set of statements about resources and can be exchanged between applications without loss of meaning. RDF also provides a vocabulary description language, RDF Schema (RDF-S), to help modeling not only structures of related resources, but also relationships between them. The capability of modeling context relationships enables context-aware applications to reason with contextual information. Another powerful data modeling markup language is the Web Ontology Language (OWL), which provides more vocabularies for expressing meaning and semantics than XML, RDF, and RDF-S. Using OWL, contextual information is modeled as a set of ontologies for specific application domains. Markup scheme context models can be used to represent complex contexts and relationships like object-oriented models. Moreover, they enable a high degree of context sharing. Contextual information can be share among different applications across different platforms. However, markup based syntax is redundant or large compared to binary representations of similar data.

4.2 Integrity

A well-designed context model needs to support the validation of structure and data integrity of contextual information. In an extra dynamic environment, contextual information may be partially lost and become incomplete, so they must be validated before used. Integrity validation can be performed at two levels: structure and data. The purpose of structure validation is to check whether the information acquired is complete in structure. For example, whether they have a predefined ending or the necessary components exist. Data validation is a more meticulous examination and attends to the data type, range, and even semantics of the information to be validated. What kinds of integrity validation a context model can support, to some extent, depends on the data structure it employs. When using tuples as the underlying data structure, limited structure integrity validation can be applied due to their lack of structure and formality; With the help of compilers and programming interfaces, object-oriented models can be validated in terms of context structure and data integrity; For markup scheme context models, there exist many tools and specifications available for validating structure and data integrity.

4.3 Manipulation

A comprehensive context model also needs to define operators which can be applied to the data structure. Except for the basic C.R.U.D. operations, another especially important operator for contextual data is context reasoning. Effective context reasoning can introduce more new contextual information derived from other types of contexts to improve user experience. For example, given contextual information like Jack's location is in the presentation room, his posture is sitting and the projector is on, the application may infer that Jack is now involved in a presentation, and automatically turn Jack's cell phone to vibration mode. Furthermore, context reasoning helps to resolve context inconsistency and conflict, and provide more exact context query results. Similar to integrity validation, context manipulation also depends on the underlying data structures of context models. For example, tuple-based contexts are easy to modify, however tuples support only simple context queries and reasoning, and are not easy to use with other efficient context retrieval and reasoning algorithms. By contrary, there are many techniques available for querying and reasoning with markup context information, but markup schemes suffer from its operational difficulties.

5 Adapting to Contexts

After acquiring raw contextual data, representing fine-grained contexts, context-aware applications have to struggle with the issue of adaptation. In particular, developers need to consider three adaptation-related questions: what to adapt, how to adapt, and when to adapt.

5.1 What to Adapt

There are three kinds of adaptations which applications may use in order to adapt to contexts: data adaptation, behavioral adaptation and structural adaptation. By data adaptation we mean that applications may change their operating data in some ways such as changing the quality of data to be accessed, transforming data form to a more suitable one, or accessing a different set of data. For example, in Odysssey, the server can select the most appropriate data at runtime from several pre-generated versions with different fidelity level according to the available resources or even energy [30]. By behavioral adaptation we refer to the fact that applications may behave differently in different contexts. For instance, a context-aware video player may pause when the audience leave and resume when they return. By structural adaptation we mean that applications may modify their internal structures or processing sequences to counteract contexts yet retain the same functions. For example, to adapt to poor computing resource, a compiler may unload the code optimization module.

5.2 How to Adapt

Two general approaches have been used to realize context-aware adaptation: transformational adaptation and compositional adaptation [31]. In transformational

adaptation, applications directly modify related specifications and/or implementations to suit changing contexts. Compositional adaptation, in contrast, does not directly modify. Rather, it responds to contexts through adding, removing, replacing, or even changing the interconnections of application algorithmic or structural parts. For example, to adapt web contents to resource-constrained devices, applications may directly transcode the original data by transformational adaptation, or replace the original data with a new version by compositional adaptation.

To use transformational adaptation, some crucial variables to describe context-aware aspects of applications are usually defined and tuned to adapt to contextual information. Although transformational adaptation is easier to implement than compositional adaptation, it suffers from two major drawbacks: first, it is hard to realize structural adaptation, which usually requires adding or removing structural parts of applications. Moreover, unimplemented versions of data and behavior cannot be introduced into applications during runtime. In transformational adaptation, everything should be designed and implemented in advance.

Compositional adaptation is more flexible. It is applicable for all types of adaptation, including data adaptation, behavioral adaptation and structural adaptation, and it allows new data, behavioral or structural parts to be adopted to address unforeseen concerns after the original construction of applications. On the downside, compositional adaptation is more complex than transformational adaptation. Developers need to pay attention to synchronization and state migration between different components [32]. For example, in the case of a streaming player which can replace various encoders and decoders using various algorithms, if the encoder at the sender side is replaced before the decoder is replaced at the receiver side, the video content may be decoded incorrectly. Thus, synchronization among components is required while adding, removing or replacing components. At the same time, in order to keep state consistence, when replacing components, state migration is necessary to correctly initialize the new version of a component.

5.3 When to Adapt

Context adaptation may occur throughout almost the entire lifetime of an application, from compile time, to load time, to run time. Generally speaking, later adaptation time supports more powerful adaptation methods, but also complicates the problem of ensuring consistency in adapted programs [33]. In compile time, context-aware applications can be adapted to contexts, especially computing contexts, with the help of compilers. For example, programs are expected to use various compiler flags for various target machine models when using GNU Compiler Collection (GCC). Another example is aspect-oriented programming (AOP). In AOP, during compilation, an aspect weaver can be used to weave different crosscutting concerns of the program together to form a program with new behavior. Context-aware applications can also defer the adaptation decision until the application load time, enabling developers to configure applications in respond to current environment after compile time. The simplest load time

adaptation implementation is to use command-line parameters. Another more flexible way is to use external configuration files. Apart from these two general methods, there are particular languages that provide primitives to support load time adaptation. For example, using Java, developers can design their own class loaders, and decide which classes are to be loaded according to current contexts, hence adapting applications. The most powerful adaptation takes place in application run time so that context-aware applications can be dynamically adapted while it is being used. Examples of techniques to implement runtime adaptation include computational reflection and dynamic weaving of aspects.

6 Challenges for Context-Awareness in Ubiquitous Computing

We have discussed the core issues of context-awareness in ubiquitous computing. During this process, we can observe that research into context-aware computing is still at its early stage and there exist several central research challenges in context-aware computing.

First of all, there is still a fundamental lack of understanding of context. Many definitions are available but none is satisfactory to most people. As a result, almost all current context-aware applications are built on their own understandings to contexts. This case makes it impossible to exchange context information between different applications. Secondly, more contexts wait to be sensed and used. Currently, most of the focus in context-aware research is on location and lots of location-sensing approaches have been explored but there have been few attempts to enhance applications by applying other contextual information, such as human emotions and activities. Finally, more comprehensive context-aware middlewares need to be provided. Early context-aware research efforts directly interact with the underlying network and operating systems to extract contextual information, process it, and adapt to it entirely at the application layer. These approaches were hard to be reused, and application developers have to "re-invent the wheel" each time a new context-aware application being developed. Moreover, the complexity of communicating with various sensors, modeling and reasoning raw contextual data, and adapting application behavior tends to distract developers from implementing real application logic, and this will slow down productivity and compromise product quality. Mechanism and primitives must be provided to conceal the complexity of these issues from context-aware application developers, and to facilitate the development process. However, although most of the existing middlewares provide software abstraction and management facilities for sensor devices, they do not provide off-the-shelf context acquisition components or underlying supports for communication with sensors. Additionally, the task of context-aware adaptation is seldom addressed at the middleware layer and just left as an application concern, and few of the works in context-aware middlewares support all needs of context acquisition, modeling, and adaptation.

References

1. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System. ACM Transactions on Information Systems 10(1), 91–102 (1992)
2. Abowd, G.D., Atkerson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A Mobile Context-Aware Tour Guide. Wireless Networks 3(5), 421–433 (1997)
3. Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: Developing A Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 17–24 (April 2000)
4. Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Laerhoven, K.V., de Velde, W.V.: Advanced Interaction in Context. In: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, pp. 89–101 (1999)
5. Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., Wong, F.L.: SenSay: A Context-Aware Mobile Phone. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
6. Schilit, B.N., Heimer, M.M.: Disseminating Active Map Information to Mobile Hosts. IEEE Network 8(5), 22–32 (1994)
7. Schilit, B.N., Adams, N., Want, R.: Context-Aware Computing Applications. Mobile Computing Systems and Applications, 85–90 (1994)
8. Long, S., Kooper, R., Abowd, G.D., Atkeson, C.G.: Rapid Prototyping of Mobile Context-Aware Applications: the Cyberguide Case Study. In: International Conference on Mobile Computing and Networking, pp. 97–107 (1996)
9. Brown, P.J., Bovey, J.D., Chen, X.: Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications 4(5), 58–64 (1997)
10. Dey, A.K., Abowd, G.D.: Towards a Better Understanding of Context and Context-Awareness. In: CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness (2000)
11. Dourish, P.: What We Talk About When We Talk About Context. Personal and Ubiquitous Computing 8(1), 19–30 (2004)
12. Tamminen, S., Oulasvirta, A., Toiskallio, K., Kankainen, A.: Understanding Mobile Contexts. Personal and Ubiquitous Computing 8(3), 135–143 (2004)
13. Oulasvirta, A., Tamminen, S., Höök Comparing, K.: Two Approaches to Context: Realism and Constructivism. In: Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense And Sensibility, pp. 195–198 (2005)
14. Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.R.: Agile Application-Aware Adaptation for Mobility. In: Proceedings of the 6th ACM Symposium on Operating Systems Principles, pp. 276–287 (1997)
15. Andersen, D., Bansal, D., Curtis, D., Seshan, S., Balakrishnan, H.: System support for bandwidth management and content adaptation in Internet applications. In: Proceedings of 4th Symposium on Operating Systems Design and Implementation, pp. 213–226 (October 2000)
16. Ward, A., Jones, A., Hopper, A.: A New Location Technique for the Active Office. IEEE Personal Communication 4(5), 42–47 (1997)
17. Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.: The Anatomy of a Context-Aware Application. Wireless Networks 8(2-3), 187–197 (2002)

18. Small, J., Smailagic, A., Siewiorek, D.P.: Determining User Location for Context Aware Computing Through the Use of a Wireless LAN Infrastructure, Project Aura Report, Carnegie Mellon University (2000), http://www.cs.cmu.edu/~aura/publications.html
19. Aitenbichler, E., Muhlhauser, M.: An IR Local Positioning System for Smart Items and Devices. In: Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, pp. 334–339 (May 2003)
20. Nerguizian, C., Despins, C., Affes, S.: Geolocation in Mines With an Impulse Response Fingerprinting Technique and Neural Networks. IEEE Transactions on Wireless Communications 5(3), 603–611 (2006)
21. Orr, R.J., Abowd, G.D.: The Smart Floor: A Mechanism for Natural User Identification and Tracking. In: Proceedings of International Conference on Human Factors in Computing Systems, pp. 275–276 (2000)
22. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less Low-Cost Outdoor Localization for Very Small Devices. IEEE Personal Communication 7(5), 28–34 (2000)
23. Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., Shafer, S.: Multi-Camera Multi-Person Tracking for EasyLiving. In: Proceedings of the 3rd IEEE International Workshop on Visual Surveillance, pp. 3–10 (July 2000)
24. Hightower, J., Borriello, G.: Location Systems for Ubiquitous Computing. IEEE Computer 34(8), 57–66 (2001)
25. Yan, H., Selker, T.: Context-Aware Office Assistant. In: Proceedings of the 5th International Conference on Intelligent user Interfaces, pp. 276–279 (2000)
26. Schmidt, A., Strohbach, M., van Laerhoven, K., Friday, A., Gellersen, H.W.: Context Acquisition Based on Load Sensing. In: Proceedings of the 4th International Conference on Ubiquitous Computing, pp. 333–350 (2002)
27. Moore, D.J., Essa, I.A., Hayes, M.H.: Exploiting Human Actions and Object Context for Recognition Tasks. In: Proceedings of IEEE International Conference on Computer Vision 1999 (ICCV 1999) (March 1999)
28. Schilit, B.N., Theimer, M.M., Welch, B.B.: Customizing Mobile Applications. In: Proceedings of USENIX Symposium on Mobile and Location-Independent Computing (USENIX Association), pp. 129–138 (August 1993)
29. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. In: 1st International Workshop on Advanced Context Modeling, Reasoning and Management, pp. 34–41 (2004)
30. Flinn, J., Satyanarayanan, M.: Energy-aware Adaptation for Mobile Applications. In: Proceedings of the 17th ACM Symposium on Operating System Principles, pp. 48–63 (December 1999)
31. Tekinerdogan, B., Aksit, M.: Adaptability in object-oriented software development workshop report. In: Cointe, P. (ed.) ECOOP 1996. LNCS, vol. 1098, Springer, Heidelberg (1996)
32. Biyani, K.N., Kulkarni, S.S.: Building Component Families to Support Adaptation. In: Proceedings of the 2005 workshop on Design and evolution of autonomic application software (DEAS 2005), St. Louis, Missouri, USA (May 2005)
33. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. IEEE Computer 37(7), 56–64 (2004)

Real-Time Embedded Software Design for Mobile and Ubiquitous Systems

Pao-Ann Hsiung*, Shang-Wei Lin, Chin-Chieh Hung, Jih-Ming Fu, Chao-Sheng Lin, Cheng-Chi Chiang, Kuo-Cheng Chiang, Chun-Hsien Lu, and Pin-Hsien Lu

Department of Computer Science and Information Engineering
National Chung-Cheng University, Chiayi, Taiwan–621, ROC
hpa@computer.org
http://www.cs.ccu.edu.tw/~pahsiung/

Abstract. Currently available application frameworks that target at the automatic design of real-time embedded software are poor in integrating functional and non-functional requirements for mobile and ubiquitous systems. In this work, we present the internal architecture and design flow of a newly proposed framework called *Verifiable Embedded Real-Time Application Framework* (VERTAF), which integrates three techniques namely software component-based reuse, formal synthesis, and formal verification. The proposed architecture for VERTAF is component-based which allows plug-and-play for the scheduler and the verifier. The architecture is also easily extensible because reusable hardware and software design components can be added. Application examples developed using VERTAF demonstrate significantly reduced relative design effort, which shows how high-level reuse of software components combined with automatic synthesis and verification increases design productivity.

Keywords: application framework, code generation, real-time embedded software, formal synthesis, formal verification, scheduling, software components, UML modeling.

1 Introduction

With the proliferation of embedded mobile and ubiquitous systems in all aspects of human life, we are making greater demands on these systems, including more complex functionalities such as pervasive computing, mobile computing, and real-time embedded computing. Currently, the design of real-time embedded software is supported partially by modelers, code generators, analyzers, schedulers, and frameworks [1, 2, 3, 4, 5, 6, 7, 8]. Nevertheless, the technology for a completely integrated design and verification environment is still immature. Furthermore, the methodologies for design and for verification are also poorly integrated relying mainly on the experiences of embedded software engineers. Motivated by the status-quo, this work demonstrates how the integration of software engineering techniques such as software component reuse, formal software synthesis techniques such as scheduling and code generation, and formal

* Corresponding author.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 718–729, 2007.

verification technique such as model checking can be realized in the form of an integrated design environment targeted at the acceleration of real-time embedded software construction.

Mobile and ubiquitous systems involve the dynamic reconfiguration of applications in response to changes in their environments. Middlewares such as network layer mobility support, transport layer mobility support, traditional distributed systems applied to mobility, middleware for wireless sensor networks, context awareness based middleware, and publish-subscribe middleware are required for efficient development of mobile and ubiquitous applications. A user can develop an application using such middlewares, however it can sometimes be too tedious and complex to consider all the different possible environments and application features. Examples of environments include office and domestic spaces, educational and healthcare institutions and in general urban and rural environments. Examples of applications include domestic and industrial security applications, education and learning applications, healthcare applications, traffic management, commercial advertising, games and arts, rescue operations, and military. Given such complex combinations of environments and applications, one would desire a higher level of reuse than that allowed by object-oriented design and middlewares. We are thus proposing an integrated design framework that allows such higher level of reuse.

As described below, several issues are encountered in the development of an integrated design framework.

1. To allow software component reuse, how do we define the syntax and semantics of a reusable component?
2. What is the control-data flow of the automatic design and verification process?
3. What kinds of model can be used for scheduling and verification?
4. What methods are to be used for scheduling and for verification?
5. How do we generate portable code that not only crosses real-time operating systems (RTOS) but also hardware platforms. What is the structure of the generated code?

Briefly, our solutions to the above issues can be summarized as follows.

1. Software Component Reuse and Integration: A subset of the Unified Modeling Language (UML) [9] is used with restrictions for automatic design and analysis.
2. Control Flow: A specific control flow is embedded within the framework, where scheduling is first performed and then verification because the complexity of verification can be greatly reduced after scheduling [3].
3. System Models: For scheduling, we use variants of Petri Nets (PN) [5] and for verification, we use Extended Timed Automata (ETA) [10], both of which are automatically generated from UML models that follow restrictions and guidelines.
4. Design Automation: For synthesis, we employ quasi-static and quasi-dynamic scheduling methods [5] that generate program schedules for a single processor. For verification, we employ symbolic model checking [11] that generates a counterexample in the original user-specified UML models whenever verification fails for a system under design. For handling complexity, we applied model-based, architecture-based, and function-based abstractions during verification.

5. Portable Efficient Multi-Layered Code: For portability, a multi-layered approach is adopted in code generation. To account for performance degradation due to multiple layers, system-specific optimization and flattening are then applied to the portable code. System dependent and independent parts of the code are distinctly segregated for this purpose.

In summary, this work illustrates how an application framework may integrate all the above proposed design and verification solutions. Our implementation has resulted in a Verifiable Embedded Real-Time Application Framework (VERTAF) whose features include formal modeling of real-time embedded systems through well-defined UML semantics, formal synthesis that guarantees satisfaction of temporal and spatial constraints, formal verification that checks if a system satisfies all properties, and code generation that produces efficient portable code.

The article is organized as follows. Section 2 described previous related work. Section 3 describes the design and verification flow in VERTAF along with an illustration example. Section 4 presents the experimental results of an application example. Section 5 gives the conclusions with some future work.

2 Previous Work

The software in mobile and ubiquitous systems has both traditional features of real-time embedded systems and also contemporary features such as adaptive resource management, proactive service discovery, context-aware coordination, multi-agents, and models for heterogeneous platforms [12]. This is mainly due to the unique requirements of such systems including interoperability, heterogeneity, mobility, survivability, security, adaptability, ability of self-organization, augmented reality, and scalable content. Though there are numerous work on the software in mobile and ubiquitous systems, besides VERTAF, there is practically no design environment that can encompass the whole design and verification flow of such systems. In the following, we briefly survey two main areas of research in this domain, including middleware and frameworks.

Middleware design is important for ubiquitous systems because it is through this software that an application connects to the network and exchanges data with other applications. Typical examples include the OSA+ middleware architecture [13], the Reconfigurable Context-Sensitive Middleware (RCSM) [14], and the T-Engine architecture [15]. The OSA+ middleware facilitates the development of distributed real-time applications in a heterogeneous environment. Some essential features of OSA+ include quality of service information requirement for each service, explicit support for asynchronous communication, real-time memory services, and small memory footprint. OSA+ has been applied to e-health management services including patient identification, location monitoring, remote checking, and continuous accurate monitoring of patient's vital signs. The RCSM architecture facilitates the development of real-time context-aware software in ubiquitous computing environments. This architecture mainly combines CORBA and FPGA such that CORBA allows mobility and FPGA allows dynamic reconfiguration (ubiquity). RCSM has been applied to sensor networks such that object interactions are context-triggered. The T-Engine architecture is an open, real-time embedded systems platform aimed at improving software productivity. The

T-Engine consortium includes computer hardware and software vendors, telecommunication carriers, and computer-using companies. T-Engine adopts a layered architecture including application, middleware, kernel, monitor, and hardware layers.

There are several either fixed architectures or variable frameworks that have been proposed in the literature for mobile and ubiquitous systems. Typical examples include the Connected Multimedia Services (CMS) framework [16], the Earl Gray JVM-based Component (EGC) framework [17], and the Static Composition Framework (SCF) of service-based real-time applications [18]. The CMS framework is based on SIP and X.10 protocols and allows multimedia sessions to be preserved when a user moves from one computing environment to another. The EGC framework analyzes component dependencies using a component-based JVM called Earl Gray. The SCF framework allows to announce services, to discover services, and to select services for an application.

3 Design and Verification Flow in VERTAF

As shown in Figure 1, VERTAF provides solutions to the various issues introduced in Section 1. The control and data flows of VERTAF are represented by solid and dotted arrows, respectively. Software synthesis is defined as a two-phase process: a machine-independent software construction phase and a machine-dependent software implementation phase. This separation helps us to plug-in different target languages, middleware, real-time operating systems, and hardware device configurations. We call the two phases as front-end and back-end phases. The front-end phase is further divided into three sub-phases, namely UML modeling phase, real-time embedded software scheduling phase, and formal verification phase. There are two sub-phases in the back-end phase, namely component mapping phase and code generation phase. We will now present the details of each phase illustrated by a running example called Entrance Guard System with Mobile and Ubiquitous Control (EGSMUC). EGSMUC is a real-time embedded system that controls any entrance with a programmable electronic lock installed. Two ways of control accesses are allowed: (a) registered users can be authenticated locally at the entrance itself, and (b) guest users may obtain a remote authentication through master acknowledgment. Here, a master could be the owner of the building to which the entrance system is protecting and he or she can have mobile and ubiquitous control access to EGSMUC. The master can grant entry access to the guest user irrespective of how he or she is connected to EGSMUC (mobile access) and also irrespective of where he or she is located (ubiquitous access). We will model EGSMUC and VERTAF will automatically synthesize and verify the code for the system.

3.1 UML Modeling

Three UML [9] diagrams are extended for real-time embedded software specification as follows.

- *Class Diagrams with Deployment*: A deployment relation is used for specifying a hardware object on which a software object is deployed. Two types of methods: event-triggered and time-triggered are used for modeling real-time behavior.

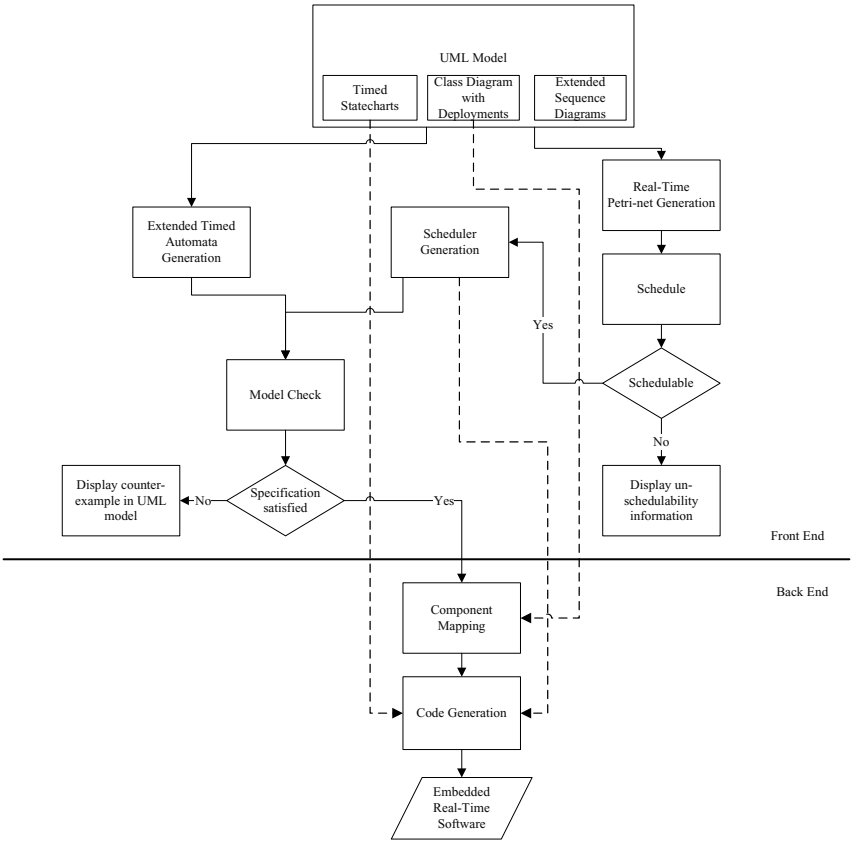


Fig. 1. Design and Verification Flow of VERTAF

- *Timed Statecharts*: UML statecharts are extended with real-time clocks that can be reset and values checked as state transition triggers.
- *Extended Sequence Diagrams*: UML sequence diagrams are extended with control structures such as concurrency, conflict, and composition, which aid in formalizing their semantics and in mapping them to Petri net models for scheduling.

For our running EGSMUC example, the system class diagram with deployment is shown in Figure 2. Other diagrams are omitted due to page limit.

3.2 Real-Time Embedded Software Scheduling

There are two issues in real-time embedded software scheduling, namely how are memory constraints satisfied and how are temporal specifications such as deadlines satisfied. Based on whether the system under design has an RTOS specified or not, two different scheduling algorithms are applied to solve the above two issues.

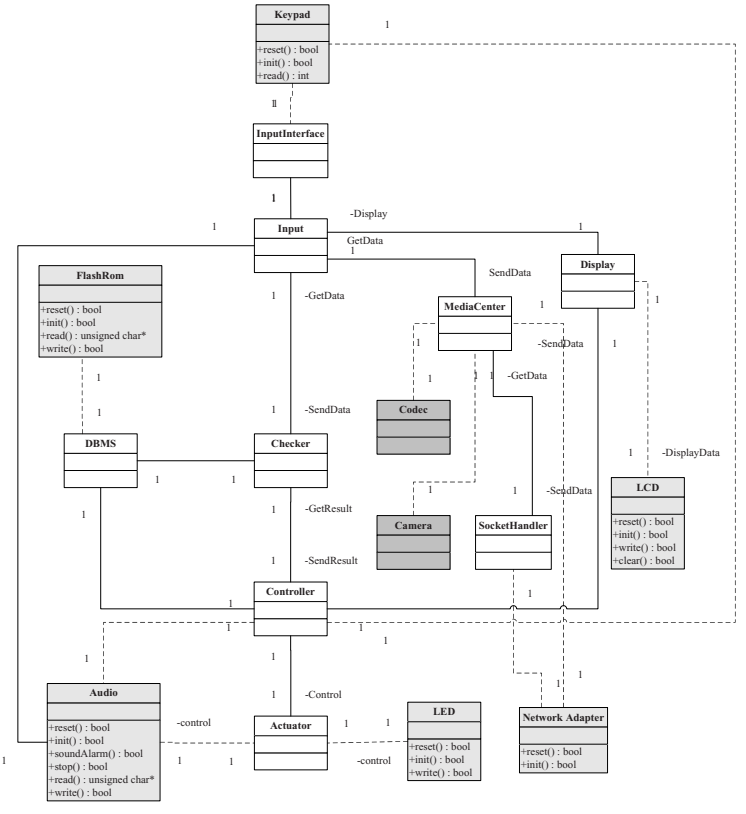


Fig. 2. Class Diagram with Deployment for Entrance Guard System with Mobile and Ubiquitous Control

- *Without RTOS*: *Quasi-dynamic scheduling* (QDS) [5] is applied, which requires *Real-Time Petri Nets* (RTPN) as system specification models. QDS prepares the system to be generated as a single real-time executive kernel with a scheduler.
- *With RTOS*: *Extended quasi-static scheduling* (EQSS) [19] with real-time scheduling [20] is applied, which requires *Complex Choice Petri Nets* (CCPN) and set of independent real-time tasks as system specification models, respectively. EQSS prepares the system to be generated as a set of multiple threads that can be scheduled and dispatched by a supported RTOS such as MicroC/OS II or ARM Linux.

To apply the above scheduling algorithms, we need to map the user-specified UML models into Petri nets, RTPN or CCPN, which are generated automatically from user-specified UML sequence diagrams, through a case-by-case construction. It is out-of-scope here. The set of RTPN or CCPN is then input to QDS or EQSS, respectively, for scheduling. Details on the scheduling procedures can be found in [5], and [19].

For systems without RTOS, we need to automatically generate a scheduler that controls the system according to the set of transition sequences generated by QDS. In VERTAF, a scheduler is constructed as a separate class that observes and controls the

status of each object in the system. Temporal constraints are monitored by the scheduler class using a global clock.

For our running EGSMUC example, a single Petri net is generated from the user-specified set of statecharts, which is then scheduled using QDS. In this example, scheduling is required only for the timers associated with the actuator, the controller, and the input object. After QDS, we found that EGSMUC is schedulable.

3.3 Formal Verification

VERTAF employs the popular model checking paradigm for formal verification of real-time embedded software. In VERTAF, formal ETA models are generated automatically from user-specified UML models by a flattening scheme that transforms each statechart into a set of one or more ETA, which are merged, along with the scheduler ETA generated in the scheduling phase, into a state-graph. The verification kernel used in VERTAF is adapted from *State Graph Manipulators* (SGM) [8], which is a high-level model checker for real-time systems that operate on state-graph representations of system behavior through manipulators, including a state-graph merger, several state-space reduction techniques, a dead state checker, and a TCTL model checker. There are two classes of system properties that can be verified in VERTAF: (1) system-defined properties including dead states, deadlocks, livelocks, and syntactical errors, and (2) user-defined properties specified in the *Object Constraint Language* (OCL) as defined by OMG in its UML specifications. All of these properties are automatically translated into TCTL specifications for verification by SGM.

For our running EGSMUC example, the ETA for each statechart were generated and then merged with the scheduler ETA. There are seven other ETA in this system example. All ETA were input to SGM and AGR was applied. Reduction techniques were then applied to each state-graph obtained from AGR. OCL constraints were then translated into TCTL and verified by the SGM model checker kernel.

3.4 Component Mapping

This is the first phase in the back-end design of VERTAF and starts to be more hardware dependent. All hardware classes specified in the deployments of the class diagram are those supported by VERTAF and thus belong to some existing class libraries. The component mapping phase then becomes simply the configuration of the hardware system and operating system through the automatic generation of configuration files, make files, header files, and dependency files. The corresponding hardware class API will be linked in during compilation.

An issue in this phase is the possible conflicts among hardware devices specified in a class diagram such as interrupts, memory address ranges, I/O ports, and bus-related characteristics such as device priorities. Users are warned in this case.

3.5 Code Generation

There are basically three issues in this phase including hardware portability, software portability, and temporal correctness. We adopt a multi-tier approach for code generation: an operating system layer, a middleware layer, and an application with scheduler

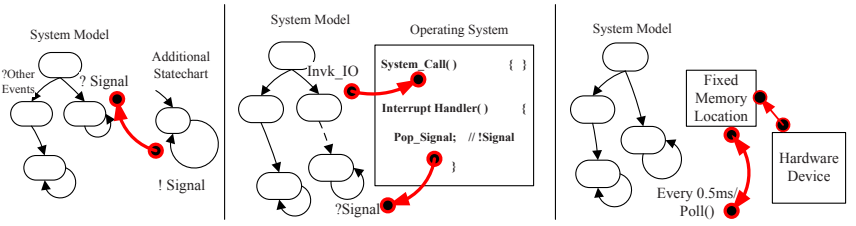


Fig. 3. I/O Delegation, Invocation, and Polling

layer, which solves the above three issues, respectively. Currently supported underlying hardware platforms include dual core ARM-DSP based, single core ARM, StrongARM, or 8051 based, and Lego RCX-based Mindstorm systems. For hardware abstraction, VERTAF supports MicroHAL and the embedded version of POSIX. For operating systems, VERTAF supports MontaVista Linux, MicroC/OS, Embedded Linux, and eCOS. For middleware, VERTAF is currently based on the Quantum Framework [7]. For scheduler, VERTAF creates a custom ActiveObject according to the Quantum API. Included in the scheduler is a temporal monitor that checks if any temporal constraints are violated.

Each ETA that is generated either from UML statecharts or from the scheduled Petri nets (sequence diagrams) is implemented as an ActiveObject in the Quantum Framework. The user-defined classes along with data and methods are incorporated into the corresponding ActiveObject. The final program is a set of concurrent threads, one of which is a scheduler that can control the other objects by sending messages to them after observing their states. For systems without an OS, the scheduler acts as a real-time executive kernel.

During code generation and the validation of automatically generated code, we discovered a peculiar problem as described in the following. UML statecharts have *run-to-completion* (RTC) semantics, that is, all actions within a state will complete execution, even if a new event or signal is received, before transiting to another state. However, in real-time embedded systems, I/O actions are usually infinite loops that poll hardware devices for data. If such I/O related high-latency low-priority events are modeled into a statechart with user-defined low-latency high-priority events, then due to RTC semantics a class object will deadlock during execution if there is no data from an I/O device that is polled. High-priority events cannot be handled. We observed this problem after code was automatically generated for our running EGSMUC example. As solutions, three methods are proposed for synthesizing the I/O interface between a user class and an I/O device. The methods are illustrated in Figure 3 and described as follows.

1. **I/O Delegation:** The deadlock can be removed from a user-defined statechart by introducing an additional statechart, as shown in the leftmost part of Figure 3, which has only one state and one self-loop transition. I/O devices are polled in that state and whenever data is available, an event or signal is broadcast. This statechart never receives any events or signals from other statecharts. The original statechart only waits for events from this statechart. However, while waiting, it can also handle high-priority low-latency events. Thus, there is no deadlock and the RTC semantics is also not violated.

Table 1. Mapping Devices and I/O Handling Mechanisms

I/O Device			OS support		
Type	**Interrupt**	**Buffer**	**AIO**	**BIO**	**NBIO**
WI	Yes	Yes	D, I	D	D, P
WB	No	Yes	N/A	D	D, P
NB	No	No	N/A	D	D, P

AIO: Asynchronous I/O, BIO: Blocking I/O, NBIO: Non-Blocking I/O,
WI: With Interrupt, WB: With Buffering, NB: No Buffering, D: Delegation, I: Invocation, P: Polling.

2. **I/O Invocation:** If the operating system supports asynchronous I/O operations, the infinite polling loops can be replaced by invoking asynchronous I/O operations through system calls, as shown in the middle part of Figure 3. After invoking an asynchronous I/O operation, the statechart can continue with other operations. When an I/O device has finished an invoked I/O operation, it interrupts the processor. The corresponding interrrupt handler then broadcasts an event, which is received by the original system statechart.
3. **I/O Polling:** This approach assumes that the I/O data will be stored in a fixed memory location such as the buffers in a hardware controller or OS. We can use a timer to poll the I/O device periodically instead of polling it in infinite loops, as shown in the rightmost part of Figure 3. Thus, the statechart will not be blocked in an infinite loop and can handle other events or signals between two timer periods.

One of the above proposed methods can be selected for automatic interface synthesis during code generation by identifying the type of I/O device. In general, I/O devices can be classified into three types: (1) WI: with buffering and interrupt support, (2) WB: with buffering but no interrupt support, and (3) NB: with neither buffering nor interrupt support. Examples include hard disk drives with interrupt support, infra-red remote controller with only buffering and no interrupt, and touch or light sensors with neither buffering nor interrupt support. The I/O delegation, invocation, and polling methods that are applicable for the three types of devices are given in Table 1 under different conditions of OS support. Normally, an OS might support asynchronous I/O (AIO), blocking I/O (BIO), and non-blocking I/O (NBIO). Table 1 can be read as follows. For example, for an I/O device of the WI type, if the OS supports only BIO, then only the I/O delegation method can be used to synthesize the interface between that device and a user-defined statechart.

As observed from Table 1, the I/O delegation method is a universally applicable method, except for cases where no interface is possible such as AIO with WB and with NB devices. For our running EGSMUC example, the interfaces for infra-red remote controller, for the network adaptor, and for the keypad were all synthesized using the I/O delegation method. We also implemented the I/O invocation and polling methods for the network adaptor, which is of the WI type. All three implementations for the network adaptor were functionally equivalent, except for performance differences.

For our running example, the final application code consisted of 9 activeobjects derived from the statecharts and 1 activeobject representing the scheduler. Makefiles were generated for linking in the API of the 8 hardware classes and configuration files were generated for the ARM-DSP dual microprocessor platform called DaVinci from Texas

Instruments with MontaVista Linux as its operating system on the ARM processor and DSP/BIOS real-time kernel as the operating system on the DSP TMS6646DSP processor. There were totally 2,340 lines of C code for the full EGSMUC system, out of which the system designers had to write only around 263 lines of C code, which is only 11.2% of the full system code.

4 Analysis and Evaluation

For the running example EGSMUC, we now analyze why VERTAF is capable of generating a significant part of the system implementation code, thus alleviating the designer from the tedious and error-prone task of manual coding. Due to its application framework architecture, VERTAF supports software components that are commonly found in mobile, ubiquitous, real-time, and embedded application domains. We classify the components supported by VERTAF into the following.

- Storage and I/O Devices: This class includes all the storage and I/O devices that are supported by VERTAF and required for implementing a real-time embedded system. Examples from the EGSMUC system include FlashRom, Keypad, LCD, Audio, LED, and Camera.
- Communication Interfaces: This class includes all the interface components that allow connection with the external world, for example, wired and wireless network connection, Bluetooth, and GSM/GPRS. Network adapter is an example from EGSMUC system.
- Multimedia Processing: This class includes all the components providing API for multimedia encoding and decoding through codecs specific to hardware platforms such as the codecs provided by TI for DaVinci multimedia platform. The DSP class in the EGSMUC system is an example.
- Control and Management Interfaces: This class includes all the components for controlling and managing system components, such as the socket handler in the EGSMUC example.

To implement mobile and ubiquitous control access in a real-time embedded system, a user normally, without VERTAF, would have to install a web server, write multimedia processing code, write network code, and integrate everything together, along with application-specific context awareness or publish-subscribe middlewares. With VERTAF, most of these tedious work are not required as long as the user configures the correct components from the framework for use in his or her application.

For illustration purposes, we show how the Media Center class in the EGSMUC example was implemented using VERTAF. The Media Center class is responsible for getting acknowledgment from a mobile master ubiquitously, which means whenever a guest wants to enter the building that the EGSMUC system is guarding, the Media Center notifies the DSP class to use the Camera to capture an image of the guest and then send the guest image to a master (the owner of the building or house). The master can send an acknowledgment through the web after which the guest can enter the building. A password is setup by a guest so that the guest can enter the building within the span of time set by the master beforehand.

The architecture of the code generated by VERTAF consists of three parts, namely a web server, a QF activeobject, and an image processing interface. The web server allows a master to connect to EGSMUC using a web browser that can run Java applets. The applet opens a socket connection between the media center and the client machine of the master. The image of the guest requesting entrance is captured and processed through the image processing interface. When a master acknowledges, the guest is notified through the input class. The control and data flows of the media center are automatically generated by VERTAF and the user has to merely specify the sequence diagrams and deploy the related classes to hardware or software components in the class diagram as shown in Figure 2. Hence, VERTAF can save a lot of coding and design efforts.

There were totally 18 objects in the final application generated by VERTAF, out of which the user or designer had to only model 7 classes. The remaining 11 classes included components from all the four categories as described at the start of Section 4. Empirical results obtained from comparing two different implementations of the EGSMUC system, one using VERTAF, and one without using VERTAF, showed that not only the user written code reduced to 11.2% and the number of objects reduced to 41%, but the total time required to develop the application also reduced by more than 60%. The average learning time for each designer using VERTAF was approximately 0.1 day. The experimental and empirical results all show that VERTAF is beneficial to designers of real-time embedded software with mobile and ubiquitous control access.

By employing various construction guidelines for design and several reduction techniques for verification as described in Section 3.3, VERTAF is scalable to large and complex applications. Since VERTAF was constructed in a component-oriented way, one can also easily extend its features by plug-and-play of new components. The flow of VERTAF can also be easily modified to incorporate the changes.

5 Conclusions and Future Work

An object-oriented component-based application framework, called VERTAF, was proposed for the development of real-time embedded system applications with mobile and ubiquitous control access. It was a result of the integration of three different technologies: software component reuse, formal synthesis, and formal verification. Starting from user-specified UML models, automation was provided in model transformations, scheduling, verification, and code generation. VERTAF can be easily extended by integrating new specification languages and scheduling algorithms.

Future extensions will include support for share-driven scheduling algorithms. VERTAF will be enhanced by considering more advanced features of real-time applications, such as: network delay, network protocols, and on-line task scheduling. Performance related features such as context switch time and rate, external events handling, I/O timing, mode changes, transient overloading, and setup time will also be incorporated.

References

1. Amnell, T., Fersman, E., Mokrushin, L., Petterson, P., Yi, W.: TIMES: a tool for schedulability analysis and code generation of real-time systems. In: Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems (September 2003)

2. Douglass, B.: Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison Wesley, USA (1999)
3. Hsiung, P.: Embedded software verification in hardware-software codesign. Journal of Systems Architecture - the Euromicro Journal 46(15), 1435–1450 (2000)
4. Hsiung, P., Cheng, S.: Automating formal modular verification of asynchronous real-time embedded systems. In: Proceedings of the 16th International Conference on VLSI Design (VLSI 2003), pp. 249–254. IEEE CS Press, Los Alamitos (2003)
5. Hsiung, P., Lin, C.: Synthesis of real-time embedded software with local and global deadlines. In: Proceedings of the 1st ACM/IEEE/IFIP International Conference on Hardware-Software Codesign and System Synthesis, pp. 114–119. ACM Press, New York (2003)
6. de Niz, D., Rajkumar, R.: Time Weaver: A software-through-models framework for embedded real-time systems. In: Proceedings of the International Workshop on Languages, Compilers, and Tools for Embedded Systems, pp. 133–143 (June 2003)
7. Samek, M.: Practical Statecharts in C/C++ Quantum Programming for Embedded Systems. CMP Books (2002)
8. Wang, F., Hsiung, P.: Efficient and user-friendly verification. IEEE Transactions on Computers 51(1), 61–83 (2002)
9. Rumbaugh, J., Booch, G., Jacobson, I.: The UML Reference Guide. Addison Wesley Longman, Reading (1999)
10. Alur, R., Dill, D.: Automata for modeling real-time systems. Theoretical Computer Science 126(2), 183–236 (1994)
11. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
12. Niemelä, E., Latvakoski, J.: Survey of requirements and solutions for ubiquitous software. In: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, pp. 71–78. ACM Press, New York (2004)
13. Brinkschulte, U., Bechina, A., Keith, B., Picioroaga, F., Schneider, E.: A middleware architecture for ubiquitous computing systems with real-time needs. In: Proceedings of the IAR Workshop, Institute for Automation and Robotic Research, France (November 2002)
14. Yau, S.S., Karim, F.: Context-sensitive middleware for real-time software in ubiquitous computing environments. In: Proceedings of the 4th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp. 163–170. IEEE CS Press, Los Alamitos (2001)
15. Sakamura, K., Koshizuka, N.: T-Engine: the open, real-time embedded-systems platform. IEEE Micro 22(6), 48–57 (2002)
16. Kwak, J.Y., Sul, D.M., Ahn, S.H., Kim, D.H.: An embedded software architecture for connected multimedia services in ubiquitous network environment. In: Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems, pp. 61–64. IEEE CS Press, Los Alamitos (2003)
17. Ishikawa, H., Ogata, Y., Adachi, K., Nakajima, T.: Requirements for a component framework of future ubiquitous computing. In: Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems, pp. 9–12. IEEE CS Press, Los Alamitos (2003)
18. Estevez-Ayres, I., Garcia-Vails, M., Basanta-Val, P.: Static composition of service-based real-time applications. In: Proceedings of the 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 11–15. IEEE CS Press, Los Alamitos (2005)
19. Su, F., Hsiung, P.: Extended quasi-static scheduling for formal synthesis and code generation of embedded software. In: Proceedings of the 10th IEEE/ACM International Symposium on Hardware/Software Codesign (CODES 2002), pp. 211–216. ACM Press, New York (2002)
20. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard-real time environment. Journal of the Association for Computing Machinery 20, 46–61 (1973)

Schedulable Online Testing Framework for Real-Time Embedded Applications in VM

Okehee Goh and Yann-Hang Lee

Computer Science and Engineering Department
Arizona State University, Tempe AZ, USA
{ogoh,yhlee}@asu.edu

Abstract. The paper suggests a VM-based online testing approach in which software testing is piggybacked at runtime on a system that operates to serve actual mission. Online testing in VM is facilitated with a framework that uses persistence service to initialize the testing operation with a consistent system state. The testing operation then runs in an isolated domain which can be scheduled independently of the operating version. Thus, testing operation cannot cause unbounded pause time nor spoil the normal operation. We evaluate the feasibility of schedulable online testing with a prototype developed in MONO CLI (Common Language Infrastructure) and the experiment on the prototype.

Keywords: Online Testing, Virtual Machine, Real-Time Embedded Applications.

1 Introduction

Nowadays, the applications of real-time embedded systems have proliferated from industrial controls to home automation, communication consumer gadgets, medical devices, defense systems and so forth. The apparent trends of the systems include sophisticated features and a short production cycle. The trends have sought solutions more in software rather than in hardware and also have led to the application of virtual software execution environment (VM) as a runtime environment. VM, populated with JVM [10] and CLI[1] [5], features high portability from using intermediate code, high productivity and reusability of object-oriented languages, and a safe runtime environment. The features are beneficial to the development of real-time embedded systems as the production cycle and cost can be reduced.

As software plays an increasingly significant role in embedded systems, the demands of upgrading software are anticipated for bug fixing and for extended functionality. In fact, most embedded systems, which have long lifetimes and require high availability, are generally passive on software upgrade because upgrading software requires to restart the systems, and the newly upgraded software

[1] CLR (Common Language Runtime), which is a CLI's implementation by Microsoft and an integral part of Microsoft .NET framework, is more popularly known than CLI.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 730–741, 2007.

can introduce new types of bugs and faults. Research work on online upgrade [12,4], and reconfigurable systems [14], has been focusing on providing facilities to accomplish the software upgrade at runtime. In the meanwhile, there is no doubt as to the importance of software testing to verify the correctness, the completeness, and security, especially for mission- or safety-critical software in which even minor changes of the software require extensive testing [15].

*Software testing*is generally conducted in off-line environment with test cases generated by predetermined inputs. The weakness of the predetermined inputs, particularly derived from a model-driven formal system specification, is that the testing results are restricted to the correctness and completeness of a given model. Furthermore, an off-line testing environment for embedded software means that the software is tested in a simulation mode and does not participate in an actual mission. However, the testing of embedded software should be able to deal with external interferences and unexpected behavior of the target application environment. All possible inputs may not be known ahead, and the generation of complete test cases for all execution conditions is very problematical.

To overcome the aforementioned limitations, we suggest an online testing environment for software upgrades as a supplementary approach of an off-line testing. The testing of software upgrades is piggybacked at runtime on the systems that operate to serve an actual mission. Hence, the execution of the software dedicated to the actual mission coexists with the execution of the software to be tested. The apparent benefit of online testing is that testing undergoes not in a limited runtime environment but in an actual target environment connected to physical world. That is, the software testing is conducted by using actual inputs. To simplify terminologies to be used hereafter, the software for an actual mission, and the software to be tested are called software under operation (SUO), and software under test (SUT), respectively.

Most embedded applications run periodically with long lifetimes. At each period, they conduct computation by taking external input data, and then the computation results, represented as output data, is used to activate the target hardware. Some embedded applications' computation at each period is based on the state that has been accumulated from computations of preceding periods as well as the newly sampled input data. For online testing of this type of applications–*stateful software*, SUT must be able to start with the accumulated computation state of SUO, and since then, gets applied with the same inputs that SUO receives. Furthermore, if SUO is characterized by time constraints, whose virtue includes timely correctness, the online testing piggybacked has to be nonintrusive: the latency and pause time that SUO encounters due to online testing must be predictable and controllable. Certainly, any faulty behavior caused by SUT must be isolated to prevent SUO operations from any impact.

In this paper, we aim at a framework of schedulable online testing (SOTF) for real-time embedded software in VM. With the advent of an online testing request, the framework provides facilities to enable a testing mode where both SUO and SUT are executed concurrently. On the termination of testing, the system returns back to an operation mode of executing SUO only. It achieves

fault isolation by executing SUT in a separate partition. By checkpointing the accumulated computation state of SUO, SUT begins to execute from a consistent state. In addition, the framework logs the external input data which SUO receives, and reconstructs the logs for the execution of SUT. Finally, SOTF employs a preemptible mechanism for checkpointing and recovery of persisted states and provides the flexibility to resume the testing anytime. Hence, the timely correctness of SUO can be ensured.

In the following section, we give a discussion of related works. The target application model of online testing is introduced in Section 3. Then, the approaches and designs of the proposed schedulable online testing framework on CLI (Common Language Infrastructure) [5]'s open source platform, MONO [17], are presented in Section 4. In Section 5, the experiment on the prototyped SOTF is used to show the space overhead incurred by the testing framework. The overhead and the source of latency of online testing with SOTF are identified. Finally, Section 6 draws a conclusion.

2 Related Works

One of well-known research areas with a key role of checkpointing and/or logging is log-based rollback recovery [6]. Logging-based recovery protocol, especially on .NET framework [2,1], is tuned at component oriented distributed applications The work was motivated with the problems that process-based recovery protocol cannot detect the failure of components, and checkpointing/recovery in a process level is very heavy. The prototype employs .NET's *Object Serialization* and *Context* to support checkpointing/recovery and to enable interceptions of messages (to aid logging) on persistent components, respectively. The rebinding of recovered components is done through .NET *Remoting*'s registry so that other stable components can access the recovered components.

Simplex architecture [13] is to support the evolvability of dependable real-time computing systems. The architecture adopts analytical redundant logic: running a trusted version (a fault-proof component) and an upgrade version (a not-yet-fault-proof component) in parallel as separate software. The architecture has decision logic that monitors the behavior of an upgrade version. If a faulty action is detected from the version, the control of the system is switched to the trusted version. Resource isolation is emphasized to prevent a trusted version from being corrupted due to the faulty behavior of an upgrade version. Lee et al. [9] extended the Simplex architecture for online testing and upgrade of industrial controller in the Linux OS environment by applying the technique of *Process Resurrection* [8].

RAIC (Redundant Arrays of Independent Components) [11] is a technology that uses groups of similar or identical components to provide software dependability and allows component hot-swapping; the architecture allows addition or removal of components at runtime. When a component is swapped, the state transfer from an old component to a replaced component is supported, if the component is stateful. If the components are faulty, as examined using built-in

testing code on the controller, the controller handles the faulty exceptions and recovers the application state so that the fault is not exposed to the applications.

The primary difference of our work from the previous works is that our framework aims a testing facility in VM that allows preemption to reduce blocking delays. Thus, flexible scheduling of testing can be carried out while ensuring the timeliness of regular service.

3 Target Application Model

The target application model we envision for SOTF is a closed-loop control system. The systems basically include sensors, control processes, and actuators, and the control tasks run concurrently and periodically to control a target plant. A simplified view of the system can be described by three application-level objects for the three system components: InputData, ControlProcess, and OutputData.

Figure 1 is a simplified closed-loop control system applying the schedulable online testing. In the figure, *InputData* indicates the data collected from sensors and taken by a control process, and *OutputData* are the computational results generated by the process and passed to actuators. We assume that a control process demands upgrades to meet performance enhancement or new business requirements. The upgrade version's control process has to access the same InputData as the operation version; that is, the upgrade version maintains the same frequency and format for the access to InputData as the operation version does. It is a reasonable assumption because InputData, which is generated by a sensor as a result of monitoring the target plant, does not get changed unless the sensor or the target plant gets replaced or upgraded. The same assumption is applied to OutputData with respect to the actuator.

Consider a stateful control process where the computation result of each periodic operation depends on not only the InputData obtained at the current period but also the accumulated state from the computations of the preceding periods. An upgrade version's control process can start online testing by being initialized with the state of an operation version's control process at the

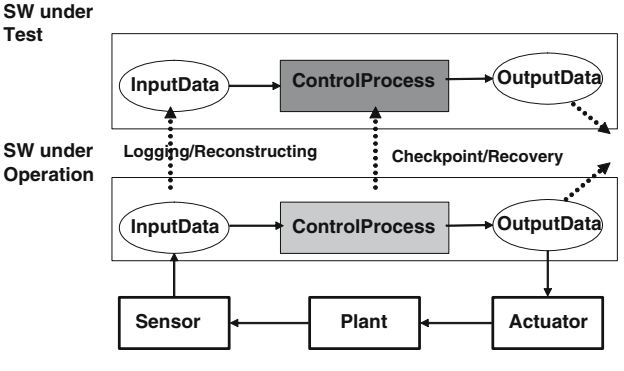


Fig. 1. Target application model of Online Testing

moment that online testing is triggered, and by accessing the same InputData as the operation version.

The concern of online testing with real-time applications is to ensure time constraints of regular service while online testing is under way. An approach to address this issue is to schedule the testing as a background task, which runs when no real-time task is ready. One of the problems that arise under this scheduling approach is the availability of InputData. When an upgrade version is ready to run, InputData that primarily stimulates the operation version's control process, may not be available any more. Also, the initial state must be preserved until the upgrade versions start to execute. The foremost important issue is that the testing operation, the logging of Inputdata, and the preservation of the initial state must be preemptible. Thus, the testing process would not block the regular service of the target system.

4 Online Testing Service

To enable schedulable online testing, we present the approaches in VM-based real-time embedded system. Although the approaches are applicable to both CLI and JVM, we built a prototype of SOTF in MONO CLI and, in the subsequent text, refer to specific technologies and standard class library of CLI.

4.1 Online Testing Framework

SOTF consists of three tasks (in the application layer) to drive online testing, and two subsystems (in the VM layer) to aid online testing: three tasks including OTTestManager, OTRecordDriver (OTRecordDrv), and OTReconstructDriver (OTReconstructDrv), and two subsystems including a preemptible persistence system and a logging/reconstructing system.

Figure 2 illustrates the architecture of the framework of schedulable online testing. The roles of the three tasks are as follows. OTTestManager is responsible

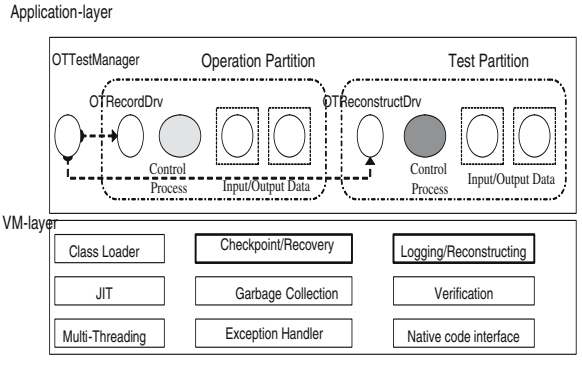


Fig. 2. Online Test Framework

for triggering online testing when software to be tested is ready. As a response of the commands from OTTestManager, OTRecordDrv interacts with SUO to checkpoint a consistent state of SUO and log input data sampled from sensors. Correspondingly, OTReconstructDrv interacts with SUT to direct the recovery and reconstruction operations of the persisted state and the logged input data. In the figure, SUO and SUT are represented with a composition of ControlProcess, InputData, and OutputData, as a simplified model suggested in Section 3.

4.2 An Isolated Testing Environment

As SUT concurrently runs with SUO, the possible faulty behavior of SUT can affect the operation of SUO. To contain the faulty behavior of SUT, it should reside in a runtime environment separated from that of SUO. We employ *Application Domain* [3] in CLI as a facility to provide an isolated runtime environment. The application domain is a lightweight address space designed as a model of scoping the execution of program code and the ownership of resources. Sharing objects between different domains is prohibited: that is, objects in one domain cannot access objects in other domains. Creating multiple application domains by starting assembly[2] with a main entry, is supported at runtime. Additionally, CLI facilitates unloading an application domain at runtime. This allows a dynamically created SUT domain when a testing operation is requested.

4.3 Preemptible Checkpointing and Recovery

If SUO's accumulated state from the computations of preceding periods is preserved, then SUT can start with the known initial state. To transfer the state from SUO to SUT across the domain boundary, we adopt the approach of checkpointing SUO's state and recovering the state for SUT. The challenge of the approach, especially for real-time applications, is that the pause time due to the checkpointing/recovery operation may be unbounded. The unpredictable latency from checkpointing can hinder the timeliness of SUO if SUO is blocked until the checkpointing operation finishes entirely.

To make it possible to bound the pause time due to checkpointing and recovery, our prior work, *schedulable persistence system* (SP system) [7] is adopted with which the persistence service runs concurrently with real-time tasks. The minimal length of the pause time, i.e. the minimal non-preemptible region in the persistence service, can be adjusted to meet the scheduling needs of real-time application tasks.

When the persisted state is deserialized, there may be a question whether the state objects can be useful directly by the SUT. If the state objects of the control process in SUT is the same object of SUO, i.e. the names of persistent classes and persistent fields in SUT is identical to these in SUO, then the persisted objects generated from SUO can be used to initialize SUT. Otherwise, we can apply a transformation script to reconstruct the state objects for SUT based on the persisted SUO objects.

[2] *Assembly* is a minimal unit of reuse, versioning, and deployment in CLI.

4.4 Logging and Reconstructing

After being initialized with the checkpointed state of SUO, SUT is ready to execute. It should receive the sampled input data similar to the one applied to SUO since the checkpoint. As a solution, the access to InputData by SUO's ControlProcess is logged and then the access to InputData by SUT's ControlProcess is sufficed with the logs. This logging/reconstructing requires to intercept the method calls on InputData objects. That is, the method calls by SUO's ControlProcess to read InputData is post-processed to log the sampled data, and the method call by SUT's ControlProcess to read InputData is pre-processed to reconstruct the sampled data based on the logs. The post- and pre-processing on InputData objects are done through the *Context* in CLI which provides an object with an execution scope. Additional services can be augmented during incoming/outgoing method calls on context-bounded objects which are derived from the *System.ContextBoundObject*. This feature has been employed in a logging-based recovery protocol on .NET framework [2,1] to enable the interceptions of messages (to aid logging) on persistent components.

5 Experiments

The experiment of the SOTF prototype on MONO CLI is performed to understand the source of latency on a testing sequence and to examine the concerns of scheduling online testing in an example system. It is conducted with C# benchmarking applications on a PC workstation with 1.5GHz Pentium IV processor and 256MB memory. To have a high resolution timer and preemptive kernel, TimeSys' Linux/RK (real-time kernel v4.1.147) [16] is used. For time measurement, a standard class library *System.DateTime.Now.Ticks* is used, which gives 100ns resolution. The C# language supports five levels of thread priorities, *Highest*, *AboveNormal*, *Normal*, *BelowNormal*, and *Lowest*. The priorities, are implemented using TimeSys RK's POSIX real-time FIFO scheduling policy.

5.1 Cost Analysis for Testing Sequence

SOTF is implemented by integrating a wide range of facilities to satisfy the requirements of online testing such as an isolated testing environment, interceptions of method calls, checkpointing/recovery, and logging/reconstructing. Using the facilities leads to some overhead. Although the amount of overhead or cost depends on the techniques employed, these types of overhead or cost are inevitable. In this experiment, we analyze the cost incurred in every stage constituting the testing sequence.

The benchmarking application, SUO, used in this experiment is a seismic event monitor, which computes the rate of seismic events by using both seismometric data newly obtained and seismometric data accumulated from a preceding duration. The seismic event monitor (SUO) runs periodically every 3ms for 1ms WCET (Worst Case Execution Time) with AboveNormal priority. The seismometric data read from its InputData object is 20Bytes. The seismometric data

accumulated from prior computations, a persistent state of SUO's ControlProcess, will be checkpointed to aid for online testing. The size of the persistent state, consisting of about 1000 composite objects including primitive types' fields, is about 20000Bytes. Its upgrade version, SUT, embodies a slightly different computation approach but generates basically the same results with the operation version SUO. When online testing starts, SUT runs every 1ms with Lowest priority. To just observe the overhead of the operation in a testing sequence, we allow the checkpointing on SUO and the recovery of persisted data on SUT to perform in a nonpreemptible mode. Additionally, the termination condition of testing is set to 300 periods of SUO's operation.

Table 1. Time line of a testing sequence

Stages	Time (ms)
(1) Receive a testing request	0
(2) Turn on checkpointing on SUO	10
(3) Start checkpointing/logging on SUO	12
(4) Complete checkpointing	25
(5) Start SUT	29
(6) Complete initialization for testing on SUT	686
(7) Start recovery on SUT	691
(8) Complete recovery, and start testing on SUT	699
(9) Complete logging on SUO	862
(10) Complete testing on SUT	1015
(11) Start unloading of SUT	1044
(12) Complete unloading of SUT	1109

Table 1 shows the cost incurred in every stage constituting the testing sequence. The result is chosen as one with the longest completion time (e.g. the moment that the unloading of SUT completes since the advent of a testing request) from 20 runs. The time specified at each stage is the elapsed time since OTTestManager received the testing request. We speculate the costs involved to carry out three functions: (1) coordinating SOTF tasks (OTTestManager, OTRecordDrv, and OTReconstructDrv) and transferring information between two different application domains, (2) conducting checkpointing and recovery, and (3) starting and unloading software at runtime.

The cost in function (1) attributes to the coordination of SOTF tasks in different application domains. For instance, the OTTestManager task, receiving an event for testing, informs OTRecordDrv to prepare for testing. What is carried out in this step is that one task fires an event to wake up a dormant thread, and the data specification for testing is transferred from one application domain (where OTTestManager runs) to the other application domain (where

OTRecordDrv runs). To enable the communication between tasks in different application domains, CLI's *AutoResetEvent* class and *AppDomain* class are used. The result, leading to about 10ms delay, which is quite expensive and mostly comes from marshaling (to pass objects between the domains), indicates that the efficient communication mechanism between different domains is desired.

Checkpointing and recovery operations take 13ms ((4)-(3)), and 8ms ((8)-(7)), respectively. The size of final persisted data, including metadata for the serialization protocol, is 33125Bytes. Compared to the experiment results (186ms and 69ms for serialization and deserialization respectively) by standard serialization library, the schedulable persistence system (SP system) in [7] substantially outperforms the standard serialization class libraries.

In Table 1, we also notice the cost for starting and unloading testing software in a new domain at runtime. The operations are implemented using AppDomain class's *CreateDomain*, *ExecuteAssembly*, and *Unload* methods. The delay reaches to 657ms((6)-(5)) to start a new software, and 65ms((12)-(11)) to unload the software, respectively. This noticeable delays comes from loading and compiling not only user classes of the new assembly but also a large portion of system classes referenced by the user classes.

5.2 Scheduling Online Testing and Space Overhead

In SOTF, it is imperative that the timeliness of applications (SUO and other real-time tasks) has to be guaranteed while testing is in progress. The simplest scheduling approach is to treat the testing as a background job so that the testing operations would have a minimal interference to the applications' timeliness. One of the concerns with the background testing job is the nonpreemptible regions caused by SOTF. The other concerns is the overhead of space that is reserved to save the logs. Logs produced by SUO have to remain until they are consumed by SUT. The issue of space overhead also attributes to the characteristic of embedded software testing, which requires to be exposed to the physical world for a long period. Thus, it can encounter all possible input data sets. Here, we consider a schedule example and use it to examine the space overhead in a testing process.

In this experiment, the SUO (a seismic event monitor) and its corresponding SUT (an upgrade version of the seismic event monitor) are same with ones in the previous experiment so that the size of persistent data for checkpointing/recov-

Table 2. Task Sets (CU: CPU Utilization, time unit: ms)

CU	T1	T2	T3	T4
0.5	0.5/5	1.6/8	1/10	1.5/15
0.6	0.5/5	1.6/8	2/10	1.5/15
0.75	1/5	2/8	2/10	1.5/15
Priority	AboveNormal	Normal	Normal	BelowNormal

ery, and the size of each log are same with the previous experiment. Additionally, service launched in an operation partition includes three more tasks besides SUO; that is, an operation partition consists of four tasks. Table 2 specifies three different task sets, and their scheduling parameters according to varying CPU utilization (CU), 0.5, 0.6, and 0.75. The table also specifies WCET, period, and priority of each task, which runs periodically; for example, T1 in the CPU utilization set 0.5 has 0.5ms WCET and 5ms period, and runs with the AboveNormal priority. Among the tasks, T1 is SUO which has a correspondent SUT. SUT, which is not specified as a task in the table because it does not account for CPU utilization, runs as a background task (with the Lowest priority). Checkpointing and recovery are carried out in a preemptible mode with the Highest priority–3ms period, and 2ms WCET. That is, when checkpointing or recovery is initiated, it runs 2ms every 3ms until it completes the requested service. In this experiment, we focus on understanding space overhead during online testing so that we ignore the testing overhead including checkpointing and recovery although it affects the schedulability of the task sets. To ease the termination condition of testing, the testing duration is limited to 300 periods of SUO operation.

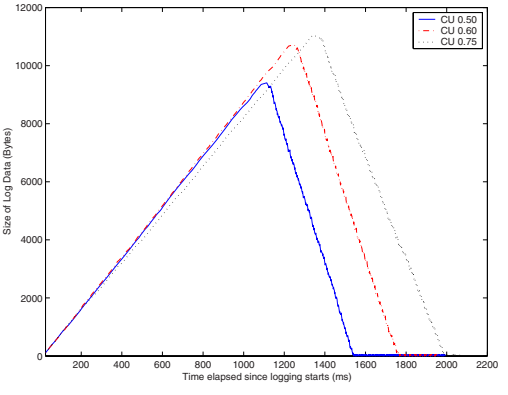


Fig. 3. Space for logs according to varying load

Figure 3 shows the space size of log data for three task sets over time until testing on SUT completes since logging on SUO started. According to the graph, the execution of SUT, conducting testing by actually consuming the logs, starts at around 1122ms, 1257ms, 1342ms for CU 0.5, CU 0.6, and CU 0.75, respectively. This start time is influenced from higher priority tasks' loads and also the overhead of online testing: as we see in the second experiment, SUT can start once the completion of checkpointing by SUO, the launch of SUT by OTTestManager, and the completion of recovery by SUT are accomplished, which take approximately 700ms. Regarding the space for logs, if the duration of SUO logging (producing logs) is not overlapped with the duration of SUT testing (consuming logs), the space for logs including metadata is 16500Bytes. In fact,

the result shows that the maximum space size for logs reaches to 9407Bytes, 10727Bytes, 11057bytes for CU 0.5, CU 0.6, and CU 0.75, respectively. The experiment shows that the space for logs reaches to the maximum during the initial stage of testing; once testing by SUT starts by consuming the logs, the space required for logs becomes less than during the initial stage. It indicates that testing can be conducted for long duration without a severe burden of space if the system can guarantee the maximum space needed in the initial stage.

Besides the space issue, conducting checkpointing and recovery in a preemptible mode shows that it can keep the maximum pause time 2ms and then their response times become 18ms, and 11ms on average, respectively, due to the execution of interleaved mutators. It indicates that checkpointing and recovery do not stop application tasks for 13ms and 8ms as its nonpreemptible mode of the second experiment.

Conclusively, predicting the upper bound of memory space reserved for logs has to consider the cost and overhead of online testing and the workload of applications.

6 Conclusion

In this paper, we depict a VM-based schedulable online testing framework for testing software upgrade of real-time embedded applications. The testing can undergo with actual input data in a target runtime environment. The framework is built by integrating a wide range of mechanisms of VM, including an isolated partition for testing, preemptible checkpointing/recovery, and logging/reconstructing the sampled input data. Meanwhile, in order to prevent the testing from causing adverse effects on the ongoing regular services of the target systems, the testing task runs in the background mode and read in sampled data via a log buffer. The experiment with the prototype of the framework, developed on MONO, demonstrates the feasibility of online testing in VM environment as well as the required capacity of the log buffer.

References

1. Barga, R., Chen, S., Lomet, D.: Improving logging and recovery performance in phoenix/app. ICDE 00, 486 (2004)
2. Barga, R., Lomet, D., Paparizos, S., Yu, H., Chandrasekaran, S.: Persistent applications via automatic recovery. IDEAS 00, 258–267 (2003)
3. Box, D., Sells, C.: Essential.NET vol. 1: The Common Language Runtime, 1st edn. Addison-Wesley, Reading (2002)
4. Dmitriev, M.: The first experience of class evolution support in PJama. In: Proc. of The 8th International Workshop on Persistent Object Systems (POS-8) and The 3rd International Workshop on Persistence and Java (PJW3), pp. 279–296. Morgan Kaufmann Publishers, San Francisco (1998)
5. ECMA. Ecma-335 common language infrastructure (2002)
6. (Mootaz) Elnozahy, E.N., Alvisi, L., Wang, Y.-M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)

7. Goh, O., Lee, Y.-H., Kaakani, Z., Rachlin, E.: Schedulable persistence system for real-time embedded applications in VM. In: EMSOFT, pp. 101–108 (2006)
8. Lee, K., Sha, L.: Process resurrection: A fast recovery mechanism for real-time embedded systems. In: Real Time and Embedded Technology and Applications Symposium, pp. 292–301 (2005)
9. Lee, K., Sha, L.: A dependable online testing and upgrade architecture for real-time embedded systems. In: RTCSA, pp. 160–165 (2005)
10. Lindholm, T., Yellin, F.: The Java Virtual Machine Specification, 2nd edn. Addison-Wesley, Reading (1999)
11. Liu, C., Richardson, D.J.: RAIC: Architecting dependable systems through redundancy and just-in-time testing. In: ICSE 2002 Workshop on Architecting Dependable Systems (2002)
12. Malabarba, S., Pandey, R., Gragg, J., Barr, E., Fritz Barnes, J.: Runtime support for type-safe dynamic Java classes. In: Bertino, E. (ed.) ECOOP 2000. LNCS, vol. 1850, pp. 337–361. Springer, Heidelberg (2000)
13. Sha, L.: Using simplicity to control complexity. IEEE Software 18(4), 20–28 (2001)
14. Soules, C.A.N., Appavoo, J., Hui, K., Wisniewski, R.W., Da Silva, D., Ganger, G.R., Krieger, O., Stumm, M., Auslander, M.A., Ostrowski, M., Rosenburg, B.S., Xenidis, J.: System support for online reconfiguration. In: USENIX Annual Technical Conference, General Track, pp. 141–154 (2003)
15. Stankovic, J.A.: Misconceptions about real-time computing: a serious problem for next generation systems. Computer Magazine, 10–19 (1988)
16. TimeSys Corporation. Timesys linux/real-time user's guide, version 2.0 (2004)
17. Ximian. MONO, http://www.go-mono.com

Scalable Lossless High Definition Image Coding on Multicore Platforms

Shih-Wei Liao[2], Shih-Hao Hung[2], Chia-Heng Tu[1], and Jen-Hao Chen[2]

[1] Graduate Institute of Networking and Multimedia
[2] Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106
{liao,hungsh,d94944008,r94125}@csie.ntu.edu.tw

Abstract. With the advent of multicores in all processor segments including mobile, embedded, desktop and server ones, we are in the new era of multiplying computing power via scaling the number of cores. The multicore approach is more versatile and programmable than the ASIC approach. For instance, the same multicore product can be adapted to the ever-improving potpourri image processing standards. Developing ASIC modules for each standard would pose million-dollar start-up cost and time-to-market disadvantage. However, the multicore approach is a two-edge sword: Unleashing its multiplying power presents significant programming challenges. The harmony between the multiplying power and programming productivity is the holy grail in this field. This paper addresses the challenge in the Digital Cinema domain. This paper presents an oblivious parallelization paradigm in both compressing and decompressing images via JPEG2000 on multicore platforms with maximum productivity. This approach dramatically reduces compression and decompression time in performing JPEG2000 lossless encoding and decoding algorithms on high definition images in almost real time without any extra hardware acceleration. By boosting parallelism coverage, the high resolution images could be compressed and decompressed in near real time: 15 images decoded/encoded per second. To the best of our knowledge, we are the first to propose a software-based coding solution using commodity multicores to achieve near real-time performance result for JPEG2000. This cost-effective approach could be applied to digital cinema on devices with multicores.

Keywords: JPEG2000, Lossless, Multicore SoC, Parallelization, Digital Cinema, Embedded System, Image Compress, Image Decompress.

1 Introduction

Full HD is the standard for the next generation digital TVs. The 2K and 4K digital cinema are also on their way. This kind of huge data flow poses difficult challenges on both storage space and transmission bandwidth. Therefore, compression is a necessity. Furthermore, while consumer product users can tolerate minor image quality distortion, professional production and military and medical customers cannot.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 742–753, 2007.

As a result, their compression schemes must be lossless. Several new standards have been proposed including H.264/AVC for motion picture, JPEG2000 for still image, and AAC for audio. These standards have better compression ratio compared to previous standards, and also support lossless coding scheme. Among these standards, JPEG2000 has been adopted by DCI (digital cinema initiative) for future distribution of movies. Therefore, JPEG2000 is taken as case study in this paper. JPEG2000, compared to previous lossless compression standards such as JPEG-LS, has a average of 30% advantage in compression ratio [1]. But it requires about four times computing power in both encoding and decoding [2]. Hardware (ASIC) and software (multicore) based approaches have been proposed to addressing these complexities. Our paper advocates the latter approach.

1.1 Motivation for the Multicore Approach

The adoption of multicores in all product segments including mobile, embedded, desktop and server is an inevitable trend. The market has witnessed many multicore products such as Sun's UltraSPARC T1 and Intel's Xeon 5100/5355/7100 in the high-end market and ARM's MPCore in the embedded market. More computing power can be obtained for a single task if we can utilize more cores to achieve that. Another incentive for utilizing more cores is that energy per core is more efficient and price per core is cheaper when we move to more cores. For instance, Xeon 5300 series has twice the number of cores as Xeon 5100, yet both the price point and the power consumption are about the same.

In contrast to the ASIC approach, multicore solutions present time-to-market advantages. For instance, multicore solutions do not impose additional hardware development time for each fixed function. In addition, the multicore approach is more flexible in adapting to new standards and configurations. The same hardware resources can be shared among different functions. Thus, better resource utilization across different functions can be achieved. On the other hand, ASIC solutions pose million-dollar start-up cost and time-to-market disadvantage on each standard they would support. Each independent IP can only process one kind of application, resulting poor resource utilization and busy bus traffic among IPs. Furthermore, the energy consumption may increase.

Multicore solutions, as we will show in this paper, are powerful enough to support mainstream applications in the Digital Cinema domain, and the performance scales with the increased number of cores. Thus, the multicore approach brings about a promising platform to support the increasing compute demand and operating modes in embedded environment such as mobile phone and digital TV/Cinema.

1.2 Studying Digital Cinema Solutions

To address the high compute and storage demands of Digital Cinema standard, people have proposed hardware (ASIC) and software (multicore) based solutions. Both approaches use parallelization to speed up processing. The JPEG2000 standard can be roughly divided into three levels of parallelization paradigms.

The highest level is the oblivious parallelization paradigm. In such scheme the image is divided into several segments and coded independently. The middle level is

the tile level parallelization paradigm, in which image is segmented into tiles, and coded altogether into single JPEG2000 code stream. The lowest level is the embedded block coding level parallelization paradigm, which partition each sub-band result from DWT (Discrete Wavelet Transforms) into separate code blocks.

Among hardware solutions, Shirai et al. [3] adopt the oblivious parallelization paradigm to build a system based on the JPEG2000 acceleration boards. There are in total four acceleration boards in the system, connected via PCI-X bus. Each acceleration board is responsible for a quarter of the image; each color component is further delegated to four JPEG2000 ASICs. Using these systems, real-time 4K SHD (4096x2160) quality international conference has been held between Kyoto, Japan and San Diego, USA. Literature [4] shows that embedded code blocking is the most time consuming part of JPEG2000 process. There are several hardware implementations that focus on this portion [5, 6, 7].

Software solutions such as Jasper [8, 9], JJ2000 [10], and Kakadu [11], mostly run in a single process. Threading techniques, used in [11] and [12], are deployed to parallelize embedded code blocking process as hardware solutions do. In this paper, we evaluate Jasper [8] and Kakadu [11] performance with high resolution, real-world pictures. As the result in Figure 1 shows, Kakadu [11] outperforms Jasper in single process with multithreading performance. Furthermore, both coders fail to encode/decode in real-time, 15 frames per second. Based on this result, we hereafter choose Kakadu [11] as our baseline software-based JPEG2000 encoder/decoder. Note that the hardware specification will be given in Section 4.

1.3 Scaling State-of-the-Art Software Solution

Kakadu uses threading to parallelize time-consuming portions. In this paper we investigate Kakadu's scalability in detail. As the figure in Section 0 shows, Kakadu fails to scale beyond three times speedup even on an 8-core machine with 32 *CoolThreads*. This poor scalability makes Kakadu unsuitable for meeting real-time, high-resolution performance requirements. This motivates us to propose a software implementation of oblivious parallelization paradigm on multicore architecture.

Our multicore implementation has three distinct advantages: First, it has time-to-market productivity advantage. It takes less than three man-months to deploy. Second, performance scales with increased number of processor cores. Our result shows that the multicore approach can meet real-time Full HD constraints. Even 2K SHD can be processed in real time. Third, the high parallelism coverage of our implementation removes the limitation induced by the 20% sequential portion in the Kakadu encode/decode process.

In summary, the paper demonstrates the feasibility of multicore solutions in the Digital TV/Cinema domain which was traditionally dominated by proprietary ASICs. In addition, we are the first to propose the software-based oblivious parallelization paradigm on commodity multicores and provide comprehensive evaluation of such scheme for Digital Cinema.

The paper is organized as follows. Section 1 describes the motivation, related works, and overview of the results. Section 2 illustrates the basics of JPEG2000 and

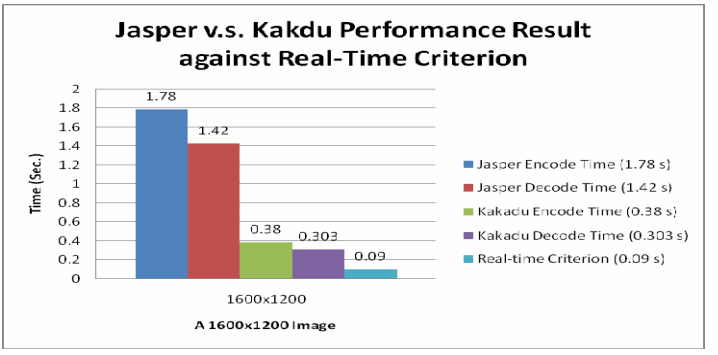


Fig. 1. Performance result of Jasper and Kakadu against Real-Time Criterion, 15 frames per second, on a 1600x1200 image

the traditional approaches to parallelize JPEG2000. We present our approach in Section 3 and its experimental results in Section 4. Section 5 concludes the paper.

2 Traditional Parallelization Approach in JPEG2000

We present the JPEG2000 flow in Section 2.1 and the traditional approaches to parallelize JPEG2000 in Section 2.2.

2.1 JPEG2000 Flow

JPEG2000, as a state-of-the-art image coding system, is a wavelet-based image compression and decompression standard created by Joint Photographic Experts Group committee. Since detailed descriptions of JPEG2000 features, internal algorithms, and functionality set are duly presented in the literature [13], here we only outline the core system and possible applications of JPEG2000 for the completeness.

The fundamental building blocks of JPEG2000 are as follows: pre-processing (component transform), discrete wavelet transform, quantization, tier-1 coding (entropy coding or arithmetic coding), and tier-2 coding (output code-stream creation). These building blocks are mainly processing logical structures of components, tiles, sub-bands, and code-blocks. Now, let us describe the encoding flow of JPEG2000 as an example to understand the relationships among those keywords above. First, the image data is decomposed into components (for example, Red, Green and Blue) and then, the components are further partitioned into (rectangular or non-overlapping) equal-sized tiles. Note that each tile could be coded independently later. Second, the tiles are broken into coefficients for sub-bands by wavelet transform. Each intra-tile's frequency characteristics are kept in the coefficients. Third, wavelet transform is followed by quantization, which is responsible for quantizing and partitioning the coefficients into code-blocks. Quantization is also the stage making decision of rate control (distortion). Fourth, code-blocks, the fundamental entities of tier-1 coding, are independent coded and the result, compressed data, is fed into next stage. Finally, those compressed data generated by previous stage is organized as packets (code-stream) by tier-2 coding.

Due to the diverse functionalities that JPEG2000 have, especially in that JPEG2000 provides both lossless and lossy coding scheme, there are many applications that could benefit from its nature capabilities, such as image distribution through internet, security systems over network, digital photography and medical imaging. From the applications above, we could know that JPEG2000 is an image coding technology for the higher quality and smaller data size. Thus, as long as the demands in seeking for perfect image quality exist, the need for JPEG2000 will not stop.

2.2 Traditional Approach in Parallelizing JPEG2000 Coding

Much work has been done in the literature in either parallelizing or optimizing the performance of JPEG2000 coding [12]. They take advantage of the multi-level parallelization opportunities in JPEG2000 standard, such as tiles, sub-bands, and code-blocks. In this section we would discuss about how far as they go and where the obstacles are. One typical obstacle is the poor scalability with respect to the number of cores.

Based on the literature and our experiments below [12, 14], the maximum speedup obtained from previous work is 3. Also, previous work typically does not focus on high resolution images or real-world images. Thus, we first run tests on our machines and compare the performance of two JPEG2000 coders, Jasper and Kakadu. Next, we determine the maximum speedup we could get from these software implementations and choose the better one as our baseline. Note that the detailed information about the machine environment and tested images will be given in Section 4.

Performance comparison between Jasper and Kakadu is presented in Figure 2. Apparently, Kakadu outperforms Jasper in the entire spectrum of images ranging from 300 mega to 100 mega pixels. Single process and single thread are used in both runs. We will only experiment with Kakadu in the remaining paper, since Jasper is slower.

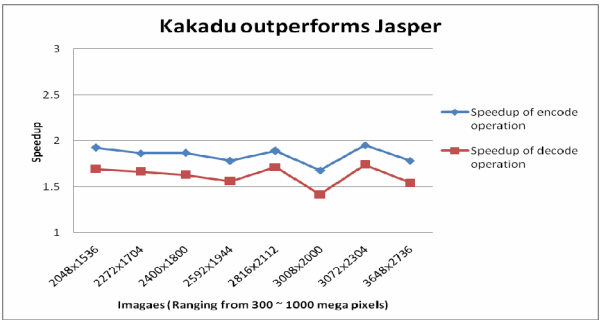


Fig. 2. Performance comparison between Jasper and Kakadu on real world images ranging from 300 to 1000 Mega Pixels

Figure 3 and 4 show the speedup of encoding an image by Kakadu on both HP DL380 with 4 logical processors and SUN Fire T2000 with 32 *CoolThreads*, respectively. The resolution of the image is 3648x2736. The speedup trend shown in Figure 3 and 4 demonstrates the parallelism coverage of Kakadu is not high enough.

That is why speedup shown in Figure 3 remains below 3 while the number of tiles increases. As we introduce more threads, run on SUN Fire T2000 with 32 *CoolThreads* enabled, the speedup shown in Figure 4 remains below 1.15. It is worthy to note that the speedup trend of decoding the image on both machines shares the same characteristics with Figure 3 and 4, respectively.

From the experiments, we conclude that due to low parallelism coverage the performance will not scale even with more hardware resources. In the next section, we propose an oblivious parallelization paradigm to boost parallelism coverage and scale with the increasing number of cores.

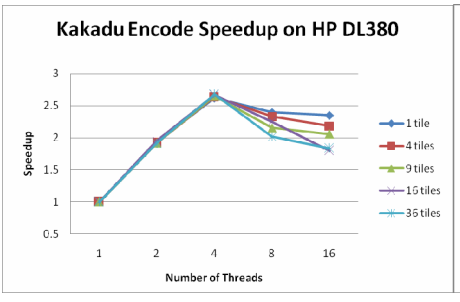


Fig. 3. Kakadu Encode Speedup on HP DL380. (A 3648x2736 image)

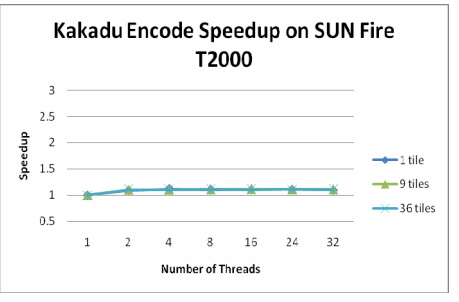


Fig. 4. Kakadu Encode Speedup on SUN Fire T2000 (A 3648x2736 image)

3 Oblivious Parallelization Paradigm

Traditional approach employs hierarchical parallelization from different stages, tiles, sub-bands, and code-blocks. Amdahl's Law states that the speedup from running a program in parallel is limited by the percentage of the program executed in parallel. We call such percentage the parallelism coverage. Section 2 indicates that while the traditional approach may be fine on a uni-processor with rich ILP (Instruction-Level Parallelism) and vector units, it does not scale with the increasing number of cores. Our profile data shows that about 80% of time is spent on decoding an input image into internal code stream. That means the ideal speedup is 5 when only this portion of the code is being parallelized in the traditional approach. This limitation on the traditional JPEG2000 parallelization approaches motivates the oblivious parallelization paradigm.

3.1 Illustration and Rationale

Our proposed approach aims at boosting the parallelism coverage yet helping programmer's productivity at the same time. Intuitively, the speedup would be N if we could process N pieces of the original image in parallel rather than the whole image in serial. Figure 5 illustrates the concept of oblivious parallelization paradigm. Note that the nature of JPEG2000 supports such style of parallel coding.

In comparison, although it is a common and effective way to process image coding in parallel, tiling may lead to the block effect in the resulting image. Besides, the

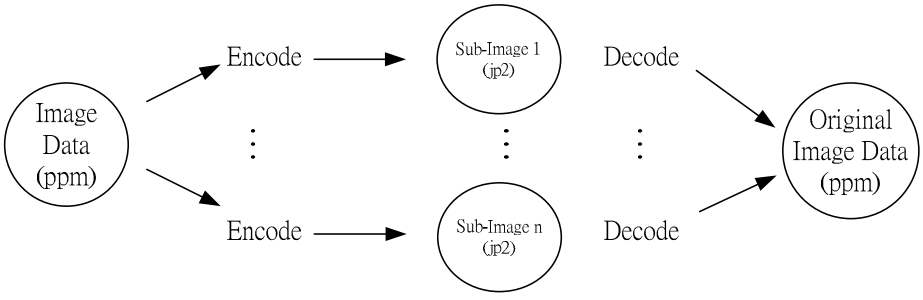


Fig. 5. Oblivious Parallelization Paradigm

parallelism coverage is lower than our oblivious parallelism scheme. To alleviate the block effect, researchers also propose other parallelized regions such as wavelet transform and tier-1 coding. However, the coverage is still lower.

The overhead of dividing and merging data in Figure 5 would be negligible, since the encoding operation (*ppm* format to *jp2* format) occurs mostly when the image capturing device reads out the raw data and stores (transforms) them into memory card or embedded memory in a compressed format, say *jp2*, and the decode operation (*jp2* format to *ppm* format) takes place mostly when the compressed file are going to be displayed on the machine where the *jp2* file located. On either circumstance, the proposed approach will mainly stress the memory system and file system. Oblivious parallelism paradigm will overlap the CPU and memory/file accesses better than the CPU-focused tiling approach.

Intermediate file size in *jp2* format is another possible issue. However, as our experiment in Section 4 shows, only less than 10% space overhead results, even if we divide an image of 2048x1036 into 140 pieces.

Finally, the division and blocking effect would not be a concern if we only focus on lossless JPEG2000 coding. The essence of reversible transformation is that either encoded or decoded data would be the same, regardless of the number of pieces we process concurrently.

3.2 Modeling the Computation Power of Platform

To bound the JPEG2000 coding time for different images, we propose a Random-Generated Image Algorithm. The algorithm generates an image with the same resolution specified at input. Due to the random nature, each pixel is independent of its surrounding pixels. This requires more computation during encoding and decoding. Thus, these images serve to predict how much time it takes to encode/decode images with certain resolutions in the worst case.

It is worth noting that this algorithm provides compute power estimation with respect to JPEG2000 processing power before any input image is used. Finally, we predict speedup using this model while oblivious parallelization paradigm is applied. The model also implies how many sub-images are needed to start with. More results in the next section demonstrate the use of this model.

Algorithm 1. Random-Generated Image Algorithm.

Input: Resolution, *X* and *Y*, of the image to be generated, where *X* represents width and *Y* represents height of the image.

Output: The image, *I*, with resolution, *X* and *Y*, where each pixel is generated based on Uniform Distribution.

for each unit *px* in *X*
  for each unit *py* in *Y*
    Red component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
    Green component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
    Blue component of pixel in coordinate (*px*, *py*) is set to a random number with uniform distribution.
  end for
end for

4 Experimental Results

This section presents the performance results, including the upper-bound execution time from random-generated images, execution time from real-world images, performance speedup on both environments, the performance trend with increasing number of processors, and the size overhead of proposed paradigm. Also, two main contributions, real time capability and scalability, are provided in this section.

Note that throughout this paper, real-world images represent the images captured by different digital cameras with different resolution. Here, we use real-world images as an example to exhibit performance results of oblivious parallelization paradigm because of wide spread use of digital camera in recent years. Our work might not only contribute to digital devices that perform image coding, but also extend to other field, such as medical image coding.

Table 1. Experiment environments

Machine Model	HP DL380	SUN Fire T2000
CPU	Xeon 3.2GHz x 2	UltraSPARC T1 @ 1.0GHz x 8
Main Memory	2048MB	16376 MB
Operating System	Linux 2.6.19	Solaris 11
Threading	4 *logical processors*	32 *CoolThreads*

We first describe the experimental setups and the input images. Table 1 outlines our environments, which are commodity main stream machines on the market. Note that the experiments done on HP DL380 machine are all run with 4 logical processors, while the experiments run on SUN Fire T2000 are with either 32 *CoolThreads*, or varied *CoolThreads*, to observe the relationship between the number of cores and speedup. The mapping between tasks and execution units in multicore is handled either by user level library or operating system, where the tasks refer to sub-images to be encoded or decoded. Table 2 listed the resolution of real-world images used in this

Table 2. Testing images information (captured by different digital cameras)

Resolution	Image size (.ppm)	Pixels
3648x2736	29,241 KB	9,980,928
3488x2616	29,388 KB	9,124,608
3264x2448	25,735 KB	7,990,272
3072x2304	20,736 KB	7,077,888
3008x2000	17,211 KB	6,016,000
2816x2112	17,051 KB	5,947,392
2592x1944	14,416 KB	5,038,848
2400x1800	12,359 KB	4,320,000
2272x1704	11,342 KB	3,871,488
2048x1536	9,000 KB	3,145,728

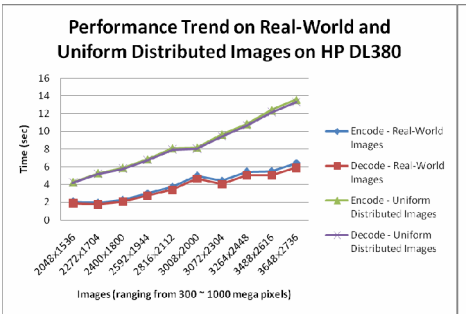


Fig. 6. Performance trend on both Real-World and Uniform Distributed images

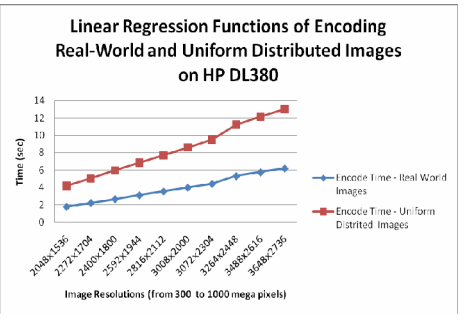


Fig. 7. Linear Regression Functions of encoding Real-World and Uniform Distributed Images on HP DL380

paper. All the images, ranging from 300 mega to 1000 mega pixel (MP), are transformed into ppm file format before the experiment.

Figure 6 depicts the trend while the image resolution increases. It is obvious that the higher the image resolution is the longer execution time it needs to process (decode/encode) the image. As expected, random-generated images (uniform-distributed images) cost more time than the real-world images do. Unrelated pixels result in extra processing time. Also, this leads to larger size of compressed file, jp2 file, than that of the original ppm file. This property could help us analyze the computing power of certain hardware and software combination. In our example, the combination is HP DL380 and Linux operating system. In order to quickly achieve what we have done above rather than testing random-generated images one resolution by one resolution, we model the patterns from the results we have. Interestingly, we obtain a linear curve in Figure 7 and the curve passes most of the points in Figure 6. This suggests that we could gain the upper-bound execution time by simply running few random-generated images instead of those in whole spectrum.

In the remaining paper, we use the smallest image, 2048x1536. Larger images would give better results. Average speedup of each image being coded on HP DL 380 is shown in Figure 8. Furthermore, the speedup of real-world images and random-generated images are shown. Note that this diagram shows significant speedup on a

random-generate images (about 50 to 60 times) faster than the sequential mode run by Kakadu. In contrast to a flat line at 100 sub-images of encoding/decoding real-world images, the speedup of random-generated images continues to increase. Note that the results suggest that the best sub-image size to perform oblivious parallelization paradigm for real-world images is about 90KB (9,000/100 KB). Furthermore, the result shows that real time encoding/decoding, 15 frames per second, is possible if the hardware resources are available. While it takes about 2.05 second to encode the 2048x1536 image in serial, the oblivious parallelization paradigm yields a speedup of 29 in the case of 44 sub-images. In addition, it takes about 0.04 second to encode the image when the speedup of 100 sub-images is 51.25.

To show the performance speedup with different number of *CoolThreads*（Virtual Processors）, in SUN Fire T2000, we use minimum and maximum number of sub-images. As shown in Figure 9, the speedup remains at 6 while 140 sub-images are used. Alternatively, the speedup rises while the numbers of Virtual Processors

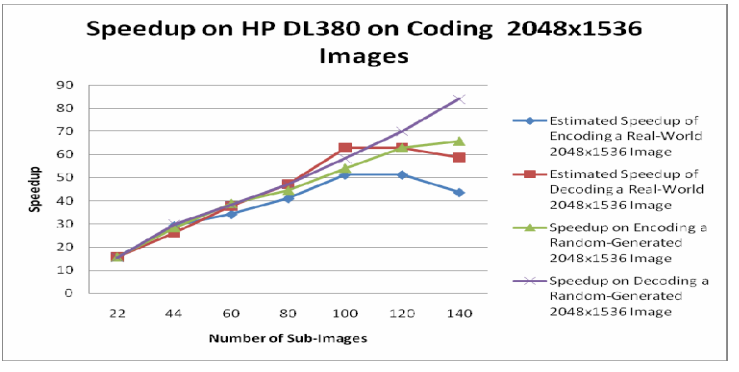


Fig. 8. Average execution time speedup on Encoding/Decoding 2048x1536 images, Real-World and Random-Generated images, on HP DL 380

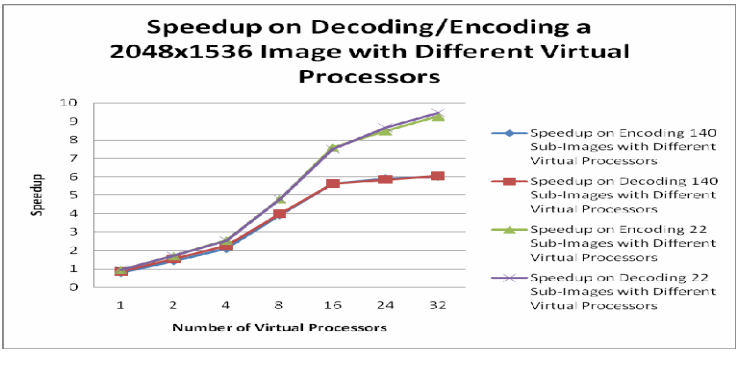


Fig. 9. Speedup on Encoding/Decoding a 2048x1536 image with Different Virtual Processors on HP DL 380 and SUN Fire T2000

increase. This shows that our paradigm is ready for multicore environment and with the maximum productivity.

The size overhead is shown in Figure 10. Apparently, the diagram shows that little overhead is introduced by our paradigm. Only less than 10% extra size is required while the numbers of sub-image are 140.

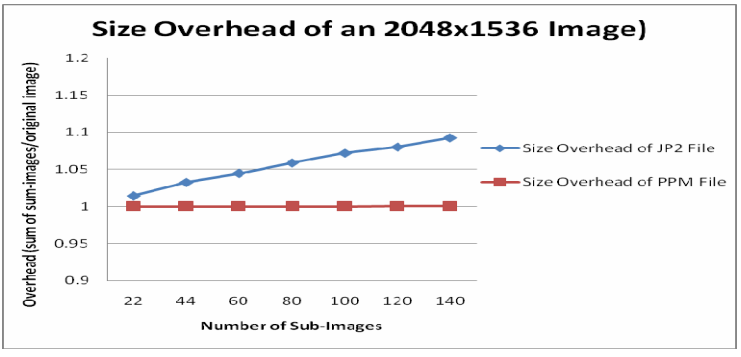


Fig. 10. Size overhead introduced by Oblivious Parallelization Paradigm

5 Conclusion and Future Work

This paper presents an oblivious parallelization paradigm to boost parallelism coverage in high definition image compression and decompression for Digital Cinema domain. To the best of our knowledge, we are the first to propose a pure software-based solution on JPEG2000 image coding. Near real-time performance could be obtained from proposed method.

The contributions of this paper are three-fold. First, we show that images for 2K Digital Cinema could be compressed or decompressed in real-time, 24 frames per second, if the hardware resources are adequate. Second, the potential scalability and overhead of oblivious parallel paradigm are well presented. Finally, we have also proposed the Random-Generated Image Algorithm to predict image coding performance for specific hardware and software combination. Furthermore, according to our results, this modeling could be done faster by running few numbers of random-generated images. This could help to shorten the time for finding upper-bound execution time of image coding on specific platform.

In the future, we plan to set up an environment to do real-time video streaming system based on proposed algorithm. Also, we would like to extend our work to medical image coding, since lossless image coding is vital in medical image area. Finally, we would like to add hand-held devices into our system to play real-time video streaming films.

Acknowledgments. We would like to thank Dr. Chung-Jr Lian for his valuable comments and suggestions, Yu-Hong Liao and Yi-Di Lin for technical support on this project and to the unknown referees for their valuable suggestions to improve the quality of this work.

References

1. Novosel, D., Kovac, M.: Still image compression analysis. In: International Symposium Electronics in Marine, pp. 567–572 (2004)
2. Santa-Cruz, D., Grosbois, R., Ebrahimi, T.: PEG 2000 performance evaluation and assessment. Signal Processing: Image Communication 17, 113–130 (2002)
3. Shirai, D., Yamaguchi, T., Shimizu, T., Murooka, T., Fujii, T.: 4K SHD Real-Time Video Streaming System With JPEG 2000 Parallel Codec. In: IEEE Asia Pacific Conference on Circuits and Systems, pp. 1855–1858 (2006)
4. Lian Jr., C., Chen, K.-F., Chen, H.-H., Chen, L.-G.: Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000. IEEE Transactions on Circuits and Systems for Video Technology 13, 219–230 (2003)
5. Chang, Y.W., Cheng, C.C., Chen, C.C., Fang, H.C., Chen, L.G.: 124 MSamples/s Pixel-Pipelined Motion-JPEG 2000 Codec Without Tile Memory. IEEE Transactions on Circuits and Systems for Video Technology 17, 398–406 (2007)
6. Fang, H.-C., Chang, Y.-W., Wang, T.-C., Lian Jr., C., Chen, L.-G.: Parallel embedded block coding architecture for JPEG2000. IEEE Transactions on Circuits and Systems for Video Technology 15, 1086–1097 (2005)
7. Andra, K., Chakrabarti, C., Acharya, T.: A high-performance JPEG2000 architecture. IEEE Transactions on Circuits and Systems for Video Technology 13, 209–218 (2003)
8. Adams, M.D., Kossentini, F.: JasPer: a software-based JPEG-2000 codec implementation. In: IEEE International Conference on Image Processing, pp. 53–56 (2000)
9. JasPer JPEG2000 codec. http://www.ece.uvic.ca/ mdadams/jasper
10. JJ2000, A JAVATM implementation of JPEG 2000. http://jj2000.epfl.ch
11. Kakadu JPEG2000 codec. http://www.kakadusoftware.com
12. Meerwald, P., Norcen, R., Uhl, A.: Parallel JPEG2000 Image Coding on Multiprocessors. In: IEEE International Parallel and Distributed Processing Symposium, pp. 2–7 (2002)
13. Skodras, A., Christopoulos, C., Ebrahimi, T.: The JPEG 2000 still image compression standard. IEEE Signal Processing Magazine 18, 36–58 (2001)
14. Hong Man, A.D., Kossentini, F.: Performance Analysis of the JPEG 2000 Image Coding Standard. Multimedia Tools and Applications 26, 27–57 (2005)

Self-stabilizing Structure Forming Algorithms for Distributed Multi-robot Systems

Yansheng Zhang, Farokh Bastani, and I-Ling Yen

University of Texas at Dallas
{yxz037000,bastani,ilyen}@utdallas.edu

Abstract. Object transportation is an important emerging application of multi-robotic swarm systems. It requires a large number of robots to be dynamically coordinated in real-time to form structures around any given rigid body that is to be lifted and transported. This paper systematically investigates the major issues that need to be addressed in methods of dynamically forming robust transportation structures. Two self-stabilizing algorithms are developed to enable a swarm of robots to form a structure to handle object transportation. Even with only a limited localized view of the environment, the algorithm enables individual robots to cooperatively form a safe structure. The stability and fault-tolerance properties of the algorithms are formally proven. The performance of the self-stabilizing algorithms, in terms of their efficiency of convergence, is evaluated via experimental studies and the results show that the system can achieve the goals in real-time.

Keywords: Swarm robotic systems, self-organizing, self-stabilizing systems.

1 Introduction

A robotic swarm system is a collective multi-agent system composed of multiple autonomous robots, interacting and cooperating with each other to accomplish some specified task. In recent years, a lot of research efforts have been devoted to the development of swarm system models and techniques [4]. Many application systems, such as large-scale unmanned space exploration, underwater missions [2], etc. can potentially take advantage of swarm system techniques. Compared to the traditional individual robot control approach, the swarm robot system has better potential in achieving more complicated tasks and can better adapt to failures or unexpected situations.

One major direction in swarm systems investigates the transportation of large objects, which is frequently required in many robotic applications. For example, in space exploration task, thousands of mobile robots may be launched into asteroid belt to collect data for further analysis. Some interested big object need to be transported by multiple robots back to the spaceship. Another example is pathway clearing. There may be some large trash that can only be removed by group of robots through tight cooperation. To build such a system, we usually divide the whole task into smaller subtasks. The first task is the coalition formation [3] which requires multiple robots form a group to accomplish a common goal. Specifically, the group of robots need to

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 754–766, 2007.

form a suitable structure [5] with which the weight of the load can be distributed evenly on the robots and the balance of the rigid body can be easily maintained in spite of some disturbance during the transportation. Thus the structure should be flexible such that it can be easily adjusted under different situations during the mission.

Self-organizing or self-assembling is the fundamental technique in swarm robotic systems for achieving structure formation. Assume that a group of robots are initially located randomly in an environment. They should automatically assemble to form desired structures to facilitate object movement. The structure should be adjusted dynamically and autonomously during the execution of the transportation tasks. The coordination should be implicit, i.e., the behavior of an individual robot is based only on its local information and interactions with neighboring robots while achieving the desired behavior of the system. Moreover, individual robots should be allowed to dynamically join or leave the group without significantly affecting the emergent system behavior. Thus, self-organizing approach achieves better reliability, extensibility, robustness, and fault tolerance.

Many self-organizing techniques have been developed in the literature. One major approach is to apply artificial evolution techniques to synthesize the individual robot controller so as to achieve a collective behavior of the system. In [5], a group of mobile robots are trained to approach a prey, self-assemble into structures and pull or push the prey towards a target location. The system is capable of coping with different structures and shapes of the prey. But it takes a long time to evolve a good controller. Moreover, the neural network controller may result in multiple structures given the same initial prey shape and robot's position. In some of these structures, all the robots aggregate at one side of the rigid body, which make the robot improperly hold and support the body. Another approach is based on the concept of force field [6]. The attractive potential and repulsive potential are used to manipulate the robots to move toward the target and/or to distribute evenly around a rigid body. Though the system can form flexible structures, every robot is required to get the other robots' position information periodically to calculate the potential. This is not practical since most of the robot being used in the swarm system only have local and limited sensing and communication abilities. Moreover, it takes a long time for these algorithms to converge.

In this paper, we design self-stabilization algorithms [1] for a swarm of robots to form a structure to handle object transportation. We focus on efficient convergence of the algorithm such that the system can fulfill the specified tasks in real-time. Also, the constraint on the number of robots is carefully studied to ensure a safe structure that helps the rigid body keep balance in transportation. Moreover, in the process of transportation, the self-stabilization algorithm supports autonomous structure adaptation to cope with failures or environment changes. Hence, the system is robust and can tolerate failures of individual robots.

The rest of the paper is organized as follows: In Section 2, we present our model for individual robots as well as the problem specification. Section 3 presents two algorithms for the lower level planners to cooperatively form a safe structure. Section 4 describes simulation result and Section 5 summarizes the paper and outlines some future areas of research.

2 System Model and Problem Specification

Here we discuss the system model for object transformation by a swarm of robots. A detailed description about the robot capabilities is discussed in Section 2.1. Section 2.2 defines some basic definition about the rigid body and system configuration. Problem specification and the constraint on the formed structure are discussed in Section 2.3.

2.1 Mobile Robot Model

We consider mobile robots with sensors and actuators and wireless communication devices (as shown in Fig. 1). Initially, we assume that n robots form a coalition as one group. Let r_i denote the i'th robot. To transport a rigid body, the robot should have the following capabilities and properties.

1) The robot has two wheels and their relative positions to the center of the robot are fixed. Each wheel can move independently at the user specified speed. Positive speed yields forward move and negative speed yields backward move. Rotation can be achieved by assigning different speed to two wheels.
2) The robot maximum linear speed is denoted by ϖ.
3) Each robot has a set of sonar and color sensors, which can detect obstacles and other robots around it. Furthermore, the robot can determine the distance of an object and can differentiate an obstacle from other robots in the group.
4) Each robot is equipped with a GPS device which allows it to obtain its current position.
5) Each robot is equipped with a gripper that allows the robot to grasp and lift the rigid body at all angles. The gripper can be used to enforce a fine-grained coordination to a certain extent. Slightly uncoordinated behaviors from some individual robots due to physical discrepancies will be forcefully adjusted to the desired behaviors by the force feedbacks from the grippers. Thus, we assume that during motion, the relative positions of the robots are maintained.

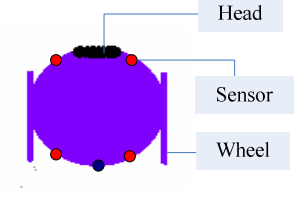


Fig. 1. Structure of Each Individual Robot

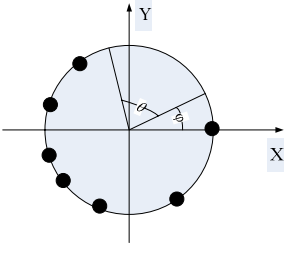


Fig. 2. An Example of Unsafe State

2.2 Basic Definitions

Let O denote the object to be transported. Assume that O is a convex polygon and let graph $G = (V, E)$ denote the polygon, where V is the set of vertices and E is the set of edges. Let gc denote the center of gravity and rad the radius of O, i.e. $rad = \text{Max}\ \{dist\ (gc, v_i) \mid \text{for all } v_i \text{ in } V\}$, where $dist$ is the distance between two points.

As discussed earlier, there are n mobile robots in the system. For simplicity, each robot is considered as a dot. Let p_i represent robot r_i's angular position relative to the center of gravity. The system state or configuration can be defined by vector $cf = [p_1, p_2, \ldots, p_n]$. Configuration vector cf is time variant, so sometimes we use $cf(t)$ to denote the system state at time t.

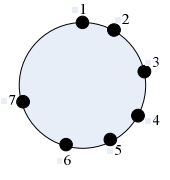
2.3 Problem Specification and Analysis

We consider the problem of n robots, $r_1 \ldots r_n$, distributing themselves around the rigid body O to support the transportation of O in a balanced manner. The most basic requirement to maintain the balance is: when we arbitrarily draw a line that crosses the center of gravity (gc) of the rigid body, there must exist some robots on either side of the rigid body or simply on the line. To allow strengthening the balance requirement and make it flexible, we require that for a given θ, when we arbitrarily draw an angle of size θ with the tip on gc, there always exists at least one robot within the angle. Here θ can be specified by the system designer to control how evenly the robots should distribute. In the following we give a formal definition for the safe state based on θ.

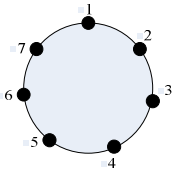
Definition 1. Safe State S: $\forall\varphi$, there always exists a robot that is within the angle range $[\varphi, \varphi+\theta]$ of the rigid body. □

When $\theta = \pi$, the safe state constraint becomes the most basic requirement. Also, θ should always be less than or equal to π. In Fig. 2, we give a counter example to illustrate an unsafe state. In this configuration, there exists an angular range $[\varphi, \varphi+\theta]$ that does not have any robot.

Generally, the system is expected to not only converge to a safe state, but also remain in the safe state in spite of some individual robot failures so that the group of robots can keep transporting the rigid body without stopping for structure reformation. This is especially important for time-critical rigid body transportation tasks. Thus, it is necessary to consider additional system requirements beyond the safe state requirement. Consider a rigid body that requires at least 3 robots to safely transport it (due to its weight). Thus, the safety requirement have $\theta = 2\pi/3$. Assume that the system actually has 7 robots which form a structure as shown in Fig. 3(a). Note that the system satisfies the safe state constraint. However, if robot 7 fails, the system will no longer be in a safe state since the angle between robot 1 and robot 6 is greater than $2\pi/3$. But if the robots form a structure as shown in Fig. 3(b), then even if one robot fails, the system still stays in the safe state.



(a). Distribution That Cannot Tolerate Failure.



(b) Distribution That Can Tolerate Failure.

Fig. 3. Two Examples of 7 Robots Distributing around a Circle

The above analysis leads to a general model of the desired system design. The system is expected to converge to a goal state which always implies the safe state. When failure occurs, the system may no longer be in the goal states, but should remain in the safe states (for example in Fig. 4, the system is driven from state 3 to state 4). Since the system continuously adjusts its state, it will automatically recover and transfer back to a goal state (state 4 back to state 3). Continuous failures may finally bring the system from a safe state to an unsafe state (for example in Fig. 4, the system is driven from state 1 to state 2), but the desired system design is to let this happen with a very small probability.

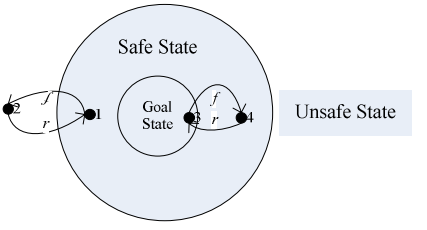


Fig. 4. Desired State and Safe State (f=fault, r=recovery)

Mapping the general system design model to our object transportation problem, the goal state is to have robots evenly distributed around the rigid body. When a robot fails, the remaining robots still form a structure that satisfies the safe state constraint. The robots will quickly recover to the goal state and during the recovery process, the system is always in the safe state. In a rare case when the system, after multiple robot failures or do not have a sufficient number of robots, may go into an unsafe state. In this case, if the system can still recover to a safe state, the robot will have to release the rigid body and reform a safe structure and, then, continue the transportation task.

Here we need to define the goal state of the system. The ideal goal state is to have robots evenly distributed around the rigid body. Specifically, the angular distance between any two adjacent robots, denoted by α, should have $\alpha = 2\pi/n$ (n is the total number of robots). But this will not be possible in physical systems. Thus, we need to define a tolerance bound without sacrificing safety even under robot failures.

Definition 2. Goal state *G*: The angular distance between any two adjacent robots, α, should satisfy $\alpha \leq 2\pi/n + \delta$. □

Here δ is the tolerance bound and it is the control parameter of the goal state. Smaller δ indicates better robot distribution (more even) structure and potentially better system fault tolerance. In the case of the safe state, we have $\alpha = \theta$.

The problem we consider is that n robots, $r_1 \ldots r_n$, should distribute themselves around the rigid body O such that the goal state is satisfied. When failure occurs, the system should still satisfy the safe state constraint if there is a sufficient number of working robots. When a robot is in motion, in case it cannot coordinate properly with other robots due to partial failures or some obstacles, we assume that the robot fails. The continuous execution of the algorithm should guarantee that the system converges to the goal state after failures.

The system, either starting from an initial state, or being in an intermediate state after failure, should converge to a goal state. This stability requirement is given in the following.

Definition 3. Stability property: If there exists a time t, such that $cf(t) \notin G$, then, there exists a time $t' < t + T$, such that $cf(t') \in G$. □

The stability property requires that, from any state, the system will converge to the goal state G in a bounded time T.

3 Self-stabilizing Structure Forming Algorithms

In a self-stabilization system, the system may start from an arbitrary state and will converge to a goal state within a bounded time. Self-stabilization concept is very suitable for the design of swarm systems so that the system can converge from an arbitrary state (potentially an unsafe state due to the initial condition or after failures) to a goal state. We apply the self-stabilization principle to the design of the rigid body transportation algorithm for a swarm of robots. In Subsection 3.1, we first try to unify the problem of considering different rigid body shapes by mapping any convex polygons to a circle. Two algorithms for the rigid body transportation problem are presented in Subsections 3.2 and 3.3. For each algorithm, the stability property is analyzed.

3.1 Shape Mapping

As defined in Section 2.2, we consider a group of robots, r_1, …, r_n, transporting a rigid body, O, that has a convex polygon shape. For structure formation, we first map O to a circle, then compute the robot movements on the circle, and finally map the movements to polygon positions.

Let C=(gc, rad) denote the circle that encloses the polygon O in which gc is the center and rad is the radius of C. We project a robot r_i next to O to the circle C (with a light source on gc) and call the projection of r_i the shadow robot sr_i. Fig. 5**.** gives an example to illustrate the mapping of a polygon to a circle and mapping of robots next to the polygon to shadow robots (or vise versa).

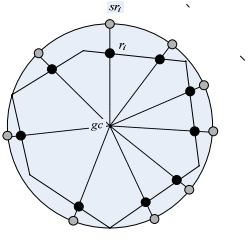


Fig. 5. Rigid body and robot mapping

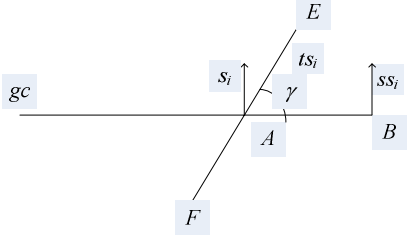


Fig. 6. Robot speed mapping

We use angular coordinates to represent robot positions and movement speeds. Note that the definition of the safe state and goal state are also based on the angular coordinates. As can be seen, sr_i and r_i has the same angular position. Thus, a safe shadow structure implies that the corresponding real structure is safe. So, we only need to consider the movement of the shadow robot for the structure formation problem. Assume that a shadow robot only has two types of simple movements, move around C clockwise or anticlockwise. Also assume that clockwise speed is positive and anticlockwise speed is negative. Here we discuss how to map the shadow robot angular speed back to the real robot translational speed.

Without loss of generality, consider that a real robot r_i at position A and r_i is moving along EF, the edge of the rigid body (as shown in Figure 6). Let γ denote the angle of the intersection between rigid body edge E-F with the circle radius gc-B. Let s_i denote the angular speed of r_i and ss_i denote sr_i. Note that the real robot and the shadow should have the same angular speed, i.e. s_i=ss_i. Also, let ts_i denotes the translational speed of r_i along edge EF. We have: $ts_i \times sin(\gamma) = s_i \times dist(gc, A)$. Thus, the speed of the real robot is $ts_i = s_i /(rad \times sin(\gamma))$. For convenience, we only consider s_i in the algorithms and ts_i can be computed accordingly. Similarly, the maximum translational speed of the robots, ϖ defined in Section 2 can be transformed into maximum rotational speed τ, where $\tau = \varpi/rad$.

3.2 First Algorithm

The simplest strategy for the robots to distribute evenly around the rigid body is to let each robot move to the relative center of its two neighbors. Assume that robot r_i is the current robot. After determining the LR and RR angles, the robot will rotate anticlockwise to reduce the angular difference between LR and RR. However, it is well known that this strategy can result in oscillation. Thus, it is necessary to control the speed of the robots in order to assure system convergence. Also, we consider an ideal physical system here and make the following assumptions to ensure the steady convergence to a goal state.

1) We assume that sensors of the robots are accurate and sensitive. It can provide the individual robots with the global information, including all the individual robot positions and the exact shape and position of the rigid body.
2) The planning process of each robot is assumed to be instantaneous.
3) The actuator of each mobile robot is accurate. It can move to any location precisely as the planner specifies. Moreover, speed acceleration and deceleration of each robot is instantaneous.

Assume that the robots are numbered sequentially based on their positions (in clockwise order), i.e., the immediate neighbors to the right and left of robot r_i are r_{i-1} and r_{i+1}, respectively. Let ξ_i represent the angular interval between r_i and r_{i+1}. ξ_i can also be viewed as the angle formed by r_i, gc, and r_{i+1} and $\xi_i = p_{i+1} - p_i$. ξ_n is the angular interval between r_n and r_1. Robot r_i and r_{i+1} are considered as the boundary robot of interval ξ_i. For robot r_i, ξ_i and ξ_{i-1} are the left and right angular intervals of r_i. Given any two robot r_i and r_j, $i > j$, r_j is considered as the $(i-j)$th right neighbor of r_i, denoted by $LN_i(i-j)$. Similarly $RN_j(i-j)$ denotes that robot r_i is the $(i-j)$th left neighbor of r_j.

Specially, $LN_i(0)=RN_i(0)=r_i$. Each robot has a local state. Let $state_i$ denote the state of robot r_i. $state_i$ can be either UnEven (*UE*), when the robot's left and right angular intervals are different, or LocalEven (*LE*), when the intervals are equal. If all the robots are in the local even state, then the system is in a global even state. The self-stabilizing structure forming algorithm for the robots is given as follows:

Algorithm 1

Robot r_i:
  $state_i = UE$;
  Periodically execute the following command;
    obtain ξ_{i-1} and ξ_i from the sensors;
    if ($\xi_i > \xi_{i-1}$) move the robot clockwise at speed τ;
    else if ($\xi_i < \xi_{i-1}$) move the robot anticlockwise at speed $-\tau$;
    else //$\xi_i = \xi_{i-1}$
    { $state_i = LE$;
      $minL$ = smallest k such that $LN_i(k)$'state is *UE*; //if no robot in *UE*, set to −1
      $minR$ = smallest k such that $RN_i(k)$'state is *UE*; //if no robot in *UE*, set to −1
      $s_L \leftarrow$ speed of $LN_i(minL)$; // set to 0 if $minL$ equals −1
      $s_R \leftarrow$ speed of $RN_i(minR)$; // set to 0 if $minL$ equals −1
      $s_i = (minL \times s_R + minR \times s_L) / (minL+minR)$;
    }

Theorem 1. Given any initial state, Algorithm 1 can stabilize within time π/τ.

Theorem 2. The system can reach a safe state if $\theta \geq \theta_i$, $1 \leq i \leq n$, that is $n \geq 2 \times \pi/\theta$.

Theorem 3. The system can reach a goal state.

3.3 Second Algorithm

In Algorithm 1, idealistic assumptions were made. In practice, robots generally do not have powerful sensors that can provide global system information. Also, due to physical constraints, it takes some time for the robots to execute the planned command. For example, it takes some time for a robot to accelerate to the computed speed. To address these limitations, we revise some assumptions in Algorithm 1 and design the second algorithm. The assumptions for the second algorithm are as follows.

1) Each robot maintains a clock, which initially is synchronized and accurate throughout the self-stabilization algorithm.
2) The planning task takes a fixed number of clock ticks and the clock tick is an atomic timing unit, denoted by λ. Planning task a periodical task. Different robots may have different periods and their initial phase may vary.
3) We consider that sensor reading and planning takes almost no time compared with command execution.
4) Each robot is going through a loop: first detecting the environmental changes, planning, and adjusting the robot position. Let Δt denotes the maximum time interval between robot sensor detecting and robot actuator actuating. Let ψ denote the angle difference threshold and its value satisfies the following constraint: $\psi > 4 \times \tau \times \Delta t$. This assumption is based on empirical data from analysis of the real system. From our empirical experience with

multi-robot systems, sensor reading and planning consume negligible time compared with executing a command since command execution phase involves adjusting the real actuator speed and keep moving for a time period. Thus, we assume that reading sensors and the planning phase start at the beginning of each round and their execution time is zero.

Oscillation may occur due to the local view of the environment. Thus we consider the angle threshold in the process of determining the robot execution command. The robot's state transition is triggered only when the environmental variation has exceeded some threshold.

The self-stabilizing structure forming algorithm for the robots is given as follows:

Algorithm 2

Robot r_i:

Periodically execute the following command.
obtain ξ_{i-1} and ξ_i from the sensors;
if ($\xi_i > \xi_{i-1} + \psi$) move the robot clockwise at speed τ;
else if ($\xi_i < \xi_{i-1} + \psi$) move the robot anticlockwise at speed $-\tau$;
else robot stops.
if (robot is moving) Wait(Δt)
else Wait(λ).

From Algorithm 2, it's necessary for the planner to wait Δt time for the actuator to adjust its speed and move before next round of planning.

Lemma 1. For any robot r_i, if $\xi_i(t) > \xi_{i+1}(t) + \psi$, then for any t_0, $t_0 < \Delta t$, $\xi_i(t+t_0) > \xi_{i+1}(t_0)$

Proof. To prove this, we only need to show that the difference between a moving robot's left angle and right angle will not change in one round once this robot plans to move. Assume that at the start of round k, robot r_i does the planning and moves in the clockwise direction. This means: $\xi_i(k) > \xi_{i+1}(k) + \psi$.

With adversary strategy, we consider robot r_{i+1} and r_{i-1} cooperating to maximally reduce the angular difference between ξ_i and ξ_{i+1}, trying to make ξ_{i+1} larger than ξ_i at some time within round k of robot r_i. It is easy to see that in the worse case, robot, r_{i+1} and r_{i-1} should move anticlockwise throughout round k. Then we have the following fact:

$\xi_i(k+1) \geq \xi_i(k) - 2\times\tau\times\Delta t$. $\quad \xi_{i+1}(k+1) \leq \xi_{i+1}(k) + 2\times\tau\times\Delta t$.

Thus, we have the following fact:

$\xi_i(k+1) - \xi_{i+1}(k+1) \geq \xi_i(k) - 2\times\tau\times\Delta t - \xi_{i+1}(k) - 2\times\tau\times\Delta t > 0$ □

Similarly, we can proof that if $\xi_i(t) < \xi_{i+1}(t) + \psi$, then for any t_0, $t_0 < \Delta t$, $\xi_i(t+t_0) < \xi_{i+1}(t_0)$.

Lemma 2. Given any number of robots and initial state, Algorithm2 can successfully terminate.

Proof. To analyze the property of the algorithm, we first define var_i as $var_i = (\xi_i - \varepsilon)^2$. Then we can obtain the the summation of all the vars, denoted by var as follows:

$$\text{var} = \sum_{i=1}^{n} \text{var}_i = \sum_{i=1}^{n} (\xi_i - \varepsilon)^2 = \sum_{i=1}^{n} (\xi_i^2 - 2\varepsilon\xi_i + \varepsilon^2) = \sum_{i=1}^{n} \xi_i^2 - n\varepsilon^2$$

Since self-stabilization phase occurs after member recruiting phase, n and ε are constant when the self-stabilization algorithm starts running. Thus, *var*'s value only depends on the summation of the square of ξ_i. Similarly, in the following paragraph, we will show that *var* keep decreasing until the self-stabilization algorithm terminates. We consider the situation in which some robots are moving during time slot k in the process of self-stabilization.

In Fig. 7, at time $k-1$, we assume that robot r_{m1}, r_{m2}, …, r_{ml} ($m1, m2 \ldots ml \in [1,n]$) are moving. Since each robot's period time is an integer of clock ticks, these robots will keep moving throughout the time slot k. Let $[p_1 \ldots p_{m1} \ldots p_{m2} \ldots p_{ml} \ldots, p_n]$ denotes the system configuration at time $k-1$ and $[p_1 \ldots p'_{m1} \ldots p'_{m2} \ldots p'_{ml} \ldots, p_n]$ the configuration at time k. Then, we can transform the concurrent robots movement at time slot k into sequential movement as follows:

Consider a moving robot r_{mi} in Fig. 8. From the above property, we can see that its movement is consistent with the environment within time slot k, independent of its neighbor's movement. Assume that the left angle and right angles of robot r_{mi} are ξ_{mi} and ξ_{mi+1} in the sequential configuration that robot r_{mi} will move. Without loss of generality, we assume that ξ_{mi} is greater than ξ_{mi+1}. Then, after robot r_{mi} moves, the angles will be changed as follows:

$$\xi'_{mi} = \xi_{mi} - \eta \qquad \xi'_{mi+1} = \xi_{mi+1} + \eta.$$

where η is the minimum angle change of robot r_{mi} in one clock tick, i.e equals $\tau \times \gamma$. Then, the impact of the robot r_{mi}'s movement on the *var* is as follows:

$$\begin{aligned} var' - var &= \xi'^{2}_{mi} + \xi'^{2}_{mi+1} - \xi^{2}_{mi} - \xi^{2}_{mi+1} \\ &= \xi'^{2}_{mi} + \xi'^{2}_{mi+1} - (\xi'_{mi} + \eta)^2 + (\xi'_{mi+1} - \eta)^2 = = 2\eta\,(\xi'_{mi+1} - \xi'_{m} - \eta). \end{aligned}$$

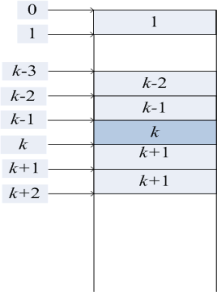


Fig. 7. System configuration

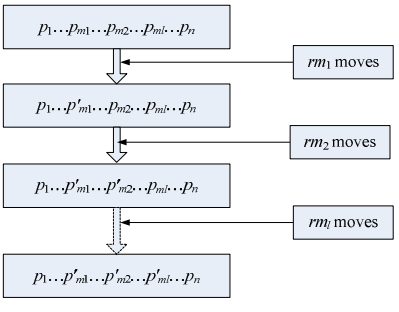


Fig. 8. System configuration decomposition

From 0, we know that $\xi'_{mi+1} - \xi'_{mi} < 0$, so we have $var' - var < -2\eta^2$. Thus, after clock ticks k, *var* will decrease at least by $2 \times l \times \eta$. Since the largest period of all the robots is Δt, it is easy to see that the self stabilization algorithm will terminate if the robots stay stationary within time period $2\Delta t$. Thus, the algorithm will terminate. □

Let κ_{max} denote the maximum task period of the robot and κ_{min} denote the minimum period. Then for any time interval $[t, t + \kappa_{max} \times \gamma]$, every robot planning task is triggered

at least one time. Hence, if all the robots stay stationary, the algorithm terminates. Thus we have the following theorem:

Theorem 4. Given any initial state, Algorithm II can successfully stabilize within time $2(\kappa_{max}+\kappa_{min})\pi^2/(\kappa_{min}\times\tau^2\times\gamma)$.

Proof

Omitted.

The introduction of the threshold in the process of determining the robot movement may make the stable distribution uneven. The worst case is as follows:

$$\xi_2=\xi_1+4\times\tau\times\Delta t \qquad \xi_3=\xi_2+4\times\tau\times\Delta t \qquad .....$$

$$\xi_{\frac{n}{2}}=\xi_{\frac{n}{2}-1}+4\times\tau\times\Delta t \qquad \xi_{\frac{n}{2}+1}=\xi_{\frac{n}{2}}-4\times\tau\times\Delta t \qquad .....$$

Thus, the system can reach a safe state if $\theta\geq\frac{2\pi}{n}+\frac{n}{4}\times4\times\tau\times\Delta t$. Consequently, the following theorems hold.

Theorem 5. The system can reach a safe state if $\frac{2\theta-\sqrt{\theta^2-8\pi\tau\Delta t}}{2\tau\Delta t}\leq n\leq\frac{2\theta+\sqrt{\theta^2-8\pi\tau\Delta t}}{2\tau\Delta t}$.

Theorem 6. The system can reach a goal state if $\delta> n\times\tau\times\Delta t$.

4 Experimental Study

To illustrate the effectiveness of the proposed approach to the self-organizing behavior of the multi-robot system, we test the algorithm in different cases. The algorithm is performed on a conventional 2G RAM, Intel Core 2 Duo 7600 PC with Windows XP.

The case study considers the even distribution of groups of homogeneous robots around a circle. The radius of the rigid body is 2m (meters). The parameters of the mobile robot are as follows:

Table 1. Parameter of the robot (*m*=meter, *s*=second)

Maximum translational speed (ϖ)	Maximum rotational speed (τ)	Clock Tick	Task period	Threshold (ψ)
0.2m/s	0.1	0.1s	5 (0.5s)	0.2

We test the Algorithm II in 100 rounds. In each round, a unique random seed is selected to generate the initial distribution for the robot group of size 0 to 39. Convergence time is illustrated in Fig. 9. The *x* axis denotes the number of rounds; the *y* axis denotes the convergence time for each distribution. Note, for each round, we generate 40 distributions with same random seed and all the distributions have successfully converged.

There are 89.625% initial distributions that converge within the time interval [8*s*-18*s*]. Comparing with the maximum convergence time 25 seconds, the upper bound of convergence time is π/τ=31.4s from algorithm I. The convergence time comparison between different numbers of robots is illustrated in Fig.10. The *x* axis denotes the

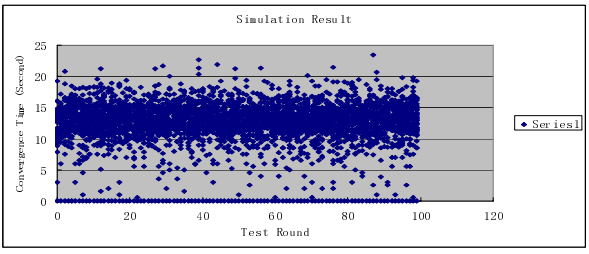


Fig. 9. Convergence Time

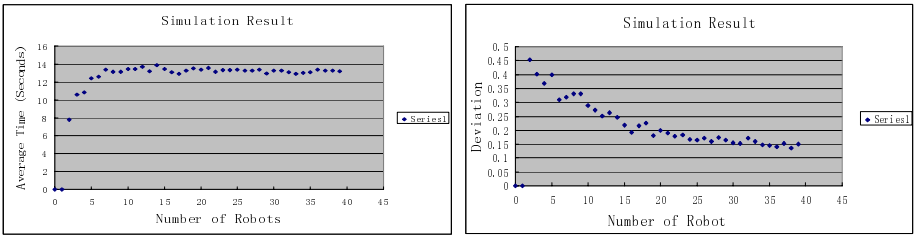


Fig. 10. Average Time and Standard Deviation

number of robots; the *y* axis denotes the time (seconds). Series 1 is the average convergence time and Series 2 is deviation of the average time.

From Fig. 10, we can conclude that 92.106% distributions converge within time interval [120, 140] in our test. Also the convergence time is irrelevant with the numbers of robot when the latter is above 5. For a general case, assume that the convergence time follows the normal distribution and the average convergence time is less than 14 and standard deviation is less than 0.5. Then 99.8% of the distributions converge within 15.5 seconds.

5 Conclusion

We have presented two self-organizing algorithms to adaptively form a safe structure for multiple robots transporting rigid objects. The algorithms are developed based on a safe self-stabilization model, where failures of some robots may bring the system to a non-goal state but the system will always stay in the safe states. The convergence of the algorithms has been formally proven and the convergence speed has been analyzed. The simulation study shows that our algorithm can efficiently converge and the convergence speed is independent of the number of robots. This is especially useful in real-time swarm-robot application since the timing property of the system behavior is predictive.

Our approach is an attempt in forming a structure for swarm-robotic rigid body transportation system. Following this paper, there are several research directions that can be investigated. We will consider more complex structure formation for other real time applications. Also, new self-organizing algorithms which deal with obstacles

during rigid body transportation will be developed. Moreover, we plan to test the algorithms using real robots to experimentally validate our approach.

References

[1] Dolev, S.: Self-stabilization. The MIT Press, Cambridge, Massachusetts, London, England (2000)
[2] Valavanis, K.P., Gracanin, D., Matijasevic, M., Kolluru, R., Demetriou, G.A.: Control Architecture for Autonomous Underwater Vehicles. IEEE Contr. Syst. 48–64 (1997)
[3] Vig, L., Adams, J.A.: Multi-Robot Coalition Formation. IEEE Transactions on Robotics 22(4) (2006)
[4] Parker, L.E: Current Research in Multi-Robot Systems. Journal of Artificial Life and Robotics (2003)
[5] Dorigo, M., Trianni, V., Sahin, E., Groß, R., Labella, T.H., Baldassarre, G., Nolfi, S., Deneubourg, J.-L., Mondada, F., Floreano, D., Gambardella, L.M.: volving Self-Organizing Behaviors for a Swarm-bot. Auton, Robots 17(2-3), 223–245 (2004)
[6] Wang, Y.F., Chirikjian, G.S.: A new potential field method for robot path planning. In: Proceedings of IEEE International Conference on Robotics and Automation, San Francisco, USA, pp. 977–982

Author Index