Hermann de Meer
James P. G. Sterbenz (Eds.)

LNCS 4124

# Self-Organising Systems

First International Workshop, IWSOS 2006, and Third International Workshop on New Trends in Network Architectures and Services, EuroNGI 2006
Passau, Germany, September 2006, Proceedings

Springer

Hermann de Meer   James P. G. Sterbenz (Eds.)

# Self-Organizing Systems

First International Workshop, IWSOS 2006
and Third International Workshop on New Trends
in Network Architectures and Services, EuroNGI 2006
Passau, Germany, September 18-20, 2006
Proceedings

Springer

Volume Editors

Hermann de Meer
University of Passau
Faculty for Mathematics and Informatics
Innstr. 33, 94032 Passau, Germany
E-mail: demeer@fmi.uni-passau.de

James P.G. Sterbenz
University of Kansas
Department of Electrical Engineering and Computer Science
Information and Telecommunication Technology Center
209 Nichols Hall, 2335 Irving Hill Rd, Lawrence, Kansas 66045-7612, USA
E-mail: jpgs@ittc.ku.edu, jpgs@comp.lancs.ac.uk

Library of Congress Control Number: 2006931005

CR Subject Classification (1998): C.2.4, C.2, D.4.4, D.2, I.2.11, H.3

LNCS Sublibrary: SL 5 – Computer Communication Networks and Telecommunications

ISSN 0302-9743
ISBN-10 3-540-37658-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-37658-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11822035 06/3142 5 4 3 2 1 0

# Preface

We welcome you to the proceedings of the workshop on self-organizing systems, held at the University of Passau, located at the confluence of the Danube, Inn, and Ilz rivers, in the beautiful state of Bavaria, Germany! We hope you enjoyed your time in this ancient and historic city.

Self-organizing systems emerge as an increasingly important area of research, particularly for computer networks. Auto-configuration and self-organization are key enablers for network optimization, self-management, self-diagnosis, self-repair, and autonomic networking in support of the increasing complexity and demands on the global Internet, as well as for emerging technologies such as ad-hoc, sensor, and peer-to-peer overlay networks.

In recognition of this, we created a cross-disciplinary program at the First International Workshop on Self-Organizing Systems (IWSOS 2006). This event was supported by the European Next Generation Internet (EuroNGI) Network of Excellence, which gathers major leading European institutions and researchers. Previously, two EuroNGI workshops on "New Trends of Network Services and Architectures" took place in Wuerzburg, Germany, and then in Villa Vigoni at Lake Como, Italy. The third edition of this workshop emerged as the first IWSOS 2006.

We hoped to seed connections between different disciplines and between industry and academia. The technical program consisted of six sessions: dynamics of structured and unstructured overlays; self-organization in peer-to-peer networks; self-organization in wireless environments; self-organization for network management and routing; self-organization in grid computing; and self-managing and autonomic computing.

Additionally, two sessions consisted of work-in-progress and position papers and featured ideas and early research that is still open for discussion and commentary. Furthermore, a poster session permitted participants to interact with members of the European Network of Excellence EuroNGI, to discuss research in an informal setting.

In addition to the reviewed paper sessions, we were exceptionally pleased to present Robbert van Renesse, Cornell University, USA, as our keynote speaker with the challenging topic on "Making Self-Organizing Systems Secure". An international panel session entitled "Self-Organising Networks: Panacea or Pandora's Box?" considered the benefits, complexities and prospects for future deployment of these emerging technologies. Two half-day tutorials on the topics of resilient and survivable networks, as well as on peer-to-peer networking, completed the attractive program.

As always, a great deal of effort has gone into creating this program. More than 70 paper submissions were received from 21 countries. We were particularly pleased with the relatively large number of papers received from Asia. The best

16 full papers were selected after a thorough peer reviewing process, in which each paper was independently evaluated by at least three reviewers. In addition to the full papers, six short papers and a preselection of posters were chosen based on their merit for the respective session and their general quality.

We wish to thank the Technical Program Committee for their hard work to ensure that high-quality papers were accepted and that new research was viewed with an open mind. Many thanks also to Amine Houyou, Patrick Wüchner, Richard Holzer, Christopher Auer, Michael Straßer, the student volunteers and many other people who helped with the workshop organization during various phases. Finally, the authors are to be thanked for their submissions and continuing excellence.

June - September 2006

Hermann de Meer and James P.G. Sterbenz
Program Chairs
IWSOS 2006

# Organization

## Program Chairs

## Steering Committee

## Program Committee

## Organization Committee

## Reviewers

Augusto Casaca
Vicente Casares-Giner
Claudio Casetti
Sumir Chandra
Costas Courcoubetis
Philippe Cudré-Mauroux
Vasilios Darlagiannis
Anwitaman Datta
Hermann de Meer
Ivan Dedinski
Zoran Despotovic
Giovanna Di Marzo Serugendo
María-José Doménech-Benlloch
Alessandro Duminuco
Julian Eckert
Markus Fiedler
Stefan Fischer
Michael Fry
Wojciech Galuba
David García Roger
Jan Gerke
José Manuel Giménez Guzmán
Šarūnas Girdzijauskas
Christos Gkantsidis
Martin Greiner
Indranil Gupta
Günter Haring
Hasan Hasan
David Hausheer
Oliver Heckmann
Helmut Hlavacs
Richard Holzer
Tobias Hösfeld
Amine M. Houyou
Karin A. Hummel
David Hutchison
Oana Jurca
Wolfgang Kellerer
Anne-Marie Kermarrec
Bernhard Klein
Fabius Klemm
Andre König
Rajesh Krishnan
Tronje Krop
Daniela Krüger
Geng-Sheng Kuo
Pascal Kurtansky
Vu Le Hung
Baochun Li
Nicolas Liebau
Martin Lipphardt
Luis Loyola
J.P. Martin-Flatin
Cristian Morariu
Paul Müller
Jens O. Oberender
Simon Oechsner
Melek Önen
Krishna Pandit
Manish Parashar
Dennis Pfisterer
Vicent Pla
Bernhard Plattner
Christian Prehofer
Andrés Quiroz Hernández
Idris Rai
Nicolas Repp
Ali Salehi
Daniel Schlosser
Stefan Schmidt
Roman Schmidt
Johannes Schmitt
Paul Smith
Stella Stars
Ralf Steinmetz
James P.G. Sterbenz
Burkhard Stiller
Zhili Sun
Fadi Tirkawi
Kurt Tutschku
Maarten van Steen
Martin Waldburger
Thomas Walter
Axel Wegener
Klaus Wehrle
Christian Werner
Patrick Wüchner
Linlin Xie

## Organizers

UNIVERSITÄT PASSAU

Computer Networks
& Communications
Prof. Hermann de Meer

euro ngi
NETWORK OF EXCELLENCE

# Table of Contents

# Invited Program

# Making Self-organizing Systems Secure

Robbert van Renesse

Department of Computer Science, 4105C Upson Hall,
Cornell University, Ithaca, NY 14853-7501
rvr@cs.cornell.edu

**Extended Abstract.** Network overlays provide important routing functionality not supported directly by the Internet. Such functionality includes multicast routing, content-based routing, and resilient routing, as well as combinations thereof. As network overlays are starting to be deployed for critical applications such as Internet telephony (e.g., Skype), web casting/distance education, web conferencing (e.g., NetMeeting), and even DNS replacements (CoDons), efficiency and security are becoming important attributes. For example, a web cast of a political conference may be an attractive target. Alas, most current network overlays are built from Distributed Hash Tables and spanning trees, resulting in infrastructures that are easily compromised. But traditional protocols based on Byzantine agreement do not scale to the sizes required.

We are exploring the use of randomized protocols for network overlays. Such protocols are often highly tolerant of benign failures such as crashes and message loss. We modify these protocols in non-trivial ways in order to make them tolerant of intrusions. In particular, we use epidemic protocols to build a pseudo-random mesh of participants, and use controlled flooding for disseminating information efficiently and reliably in the face of compromised participants. Note that we do not attempt to detect (Intrusion Detection, Reputation) or prevent intrusions (Access Control). Doing so would lead to an arms race that may not be productive. Instead, we only tolerate intrusions. Unlike Byzantine protocols, our protocols degrade gracefully as a larger percentage of participants is compromised.

We have built two protocols to date. The first, Fireflies, is an epidemic group membership protocol that maintains a pseudo-random mesh among its participants. Fireflies has been deployed and extensive evaluated on Planetlab. We used various forms of attack, with percentages of intruders of up to 25%. At the scale of Planetlab, Fireflies performs similarly to Chord, a well-known DHT, in terms of use of CPU, memory and bandwidth resources, but all correct nodes are guaranteed to be able to communicate with high probability. We have also developed a correctness proof of the Fireflies protocol. The second, SecureStream, is an audio and video streaming protocol. Currently, SecureStream has only been deployed on Emulab for testing scenarios, where it performs similarly to well-known protocols like SplitStream, even in the face of 20% or more intruders.

On top of these, we are currently developing a software distribution service that can reliably disseminate and install security updates without giving hackers an opportunity to reverse engineer patches and launch viruses that exploit the corresponding security holes, as well as a webcast service.

# Self-organising Networks: Panacea or Pandora's Box?

James P.G. Sterbenz[1,2]

[1] University of Kansas, USA
jpgs@ittc.ku.edu
[2] Lancaster University, UK
jpgs@comp.lancs.ac.uk

**Extended Abstract.** Self-organisation is a key enabler for a number of advanced networking scenarios. Algorithms for auto-configuration of nodes and self-organisation of networks have been traditionally applied to ad hoc networks and more recently to peer-to-peer overlays and sensor networks, in which it is critical to avoid human intervention to configure and reconfigure the network. As the complexity of the Global Internet and other attached and overlay networks increases, the ability for humans to understand, configure, and manage these networks is becoming increasingly difficult and expensive. Self-organisation appears to be a key technology to enable self-organising and self-managing autonomic networks.

This panel will consider the future prospects of self-organisation to help solve these problems. There are significant research challenges in achieving the vision of resilient, secure, policy-driven, large-scale self-organisation and self-management. Furthermore, there is the risk that the additional complexity of the new mechanisms will cause more problems than they solve. Finally, this panel will consider whether self-organisation will be a field in danger of a large amount of research activity having little impact on real network deployment as has been the case with research in QoS and multicast.

# Full Papers

# The Challenges of Merging Two Similar Structured Overlays: A Tale of Two Networks[*]

Anwitaman Datta and Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
anwitaman.datta@epfl.ch, karl.aberer@epfl.ch

**Abstract.** Structured overlay networks is an important and interesting primitive that can be used by diverse peer-to-peer applications. Multiple overlays can result either because of network partitioning or (more likely) because different groups of peers build such overlays separately before coming in contact with each other and wishing to coalesce the overlays together. This paper is a first look into how multiple such overlays (all using the same protocols) can be merged - which is critical for usability and adoption of such an internet-scale distributed system. We elaborate how two networks using the same protocols can be merged, looking specifically into two different overlay design principles: (i) maintaining the ring invariant and (ii) structural replications, either of which are used in various overlay networks to guarantee functional correctness in a highly dynamic (membership changes) environment.

Particularly, we show that ring based networks can not operate until the merger operation completes. In contrast, from the perspective of individual peers in structurally replicated overlays there is no disruption of service, and they can continue to discover and access resources that they could originally do before the beginning of the merger process, even though resources from the other network become visible only gradually with the progress of the merger process.

## 1 Introduction

In the recent years there has been an increasing trend to use resources at the edge of the network - typically desktop computers interconnected across the internet provide services and run applications in a peer-to-peer manner, as an alternative to the traditional paradigm of using dedicated infrastructure and centralized coordination and control. Many peer-to-peer applications need some basic functionalities, particularly that of locating resources efficiently in a distributed large-scale environment in a decentralized manner. *Structured overlay*

[*] The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

*networks* have come to be recognized as a generic substrate which can facilitate resource discovery in a decentralized fashion [3,9,12,13,14,16,17,18].

Even while the peer-to-peer research community prides itself to be pushing the limits of networked distributed systems, it has so far ignored a fundamental and realistic problem for structured overlays that any distributed system needs to deal with - that of making two partitions of such a system merge to become one. One can speculate several reasons for such omissions in structured overlay network research. (i) Merger of isolated overlays is trivially resolved in unstructured overlays, which is where most of the empirical information of P2P research so far has been derived from. (ii) Until recently, there has not been any real structured overlay implementations deployed and hence the problem not identified. (iii) The recent deployments and experiments have typically [1,15] been under a controlled setting, where some central coordination like the use of a common set of bootstrap nodes has been used with the intention and sufficient coordination to construct only one overlay, making sure that independent and distinct overlays are not created. Moreover, none of these experiments with real implementations looked specifically for, or even accidentally, encounter network partitioning problems.

Apart network partitioning which can lead to the creation of two disjoint overlay networks there is a more likely scenario. It may so happen that disjoint overlay networks (using the same protocols) are formed over time by disjoint group of users. One may imagine that an overlay P2P network caters to a specific interest group from a particular geographic area who participate in an overlay network. At a later time, upon discovering a hitherto unknown group of like-mind users from a different part of the world, who use their own "private" network (using same protocols), these two groups may want to merge their networks in order to benefit from their mutual resources (like content or knowledge). In fact, such isolated overlay networks may result because of initial isolation of groups because of various reasons including geographic, social or administrative - a large company or country, which may originally restrict their users from interacting with outsiders in the overlay, and changes the policy at a later time - or purely because of partitioning of the physical infrastructure.

Structured overlays have often been touted as a generic substrate for other applications and services. Ideally, there will be one or few such universal overlays [5] which will be used by a plethora of other P2P applications. Realizing such an universal service too will need the possibility to merge originally isolated networks. Small overlays can be built independently, which may later be all merged together incrementally into a single overlay network.

One can thus imagine isolated islands of functional overlays catering to their individual participants. Someday, some member from one of these overlays may discover a member from another overlay. The natural thing to do then would be to merge the two originally isolated overlays into a single overlay network. In simple file-sharing networks, the motivation of doing so will be to make accessible content from both the networks to all the users. Similar conclusions can be drawn for various other conceivable applications of overlay networks.

In unstructured overlay networks (like Gnutella), merger of two originally isolated overlays happens trivially. Whichever peers from two originally isolated networks come in contact with each other need to establish mutual neighborhood relationship, and then onwards just need to forward/route messages to each other as they do with all other neighbors. That's all! Likewise, hierarchical (super-peer based) unstructured overlays also merge together trivially. This is because no peer has any specific responsibility and can potentially be responsible for any and all resources in the network.

Recent years have seen an increasing advocacy of structured overlays, because of the efficiency and completeness[1] their guarantees. There has been intensive study of structured overlays, looking primarily in three important aspects of such an infrastructure - (i) the topology of the routing network and the corresponding routing algorithms, (ii) resilience of such network under churn (membership dynamics) and (iii) load-balancing. These are critical issues that needed to be addressed in order to build practical structured overlay networks which can be deployed over the internet. The last five years of overlay related research concentrated on these issues. However, the issue of merger of structured overlays has so far not only not been addressed but has even hardly been recognized,[2] possibly because of the preoccupation of the P2P community with the other above mentioned infrastructure issues, which were necessary even to begin with development of actual software that could be deployed.

In this paper we take a first look at how and whether such merger can be achieved. And at what cost in terms of algorithmic complexity and development, as well as in terms of operational cost (bandwidth) and performance (interruption of service).

We do a case study for two structured overlays - ring based overlay (like Chord) and P-Grid, in order to identify the properties of an overlay network which would facilitate or hinder successful merger of distinct (but using the same protocols) overlay networks. These two networks, apart from being two of the few structured overlays which have really been implemented, benchmarked and deployed, also are representative of two very different design principles. Chord relies on the principle of maintaining what is called the ring invariant in order to guarantee functionality of the overlay network in presence of membership dynamics (churn). Maintaining the ring under churn is relatively straightforward and provides good resilience [8]. P-Grid uses prefix-based routing (PRR [13] topology). Such topology has been shown to have poorer static resilience than a ring based topology [8]. P-Grid alleviates the problem, and still avoids the use of maintaining a ring by using a different way to provide redundancy - which we call *structural replication.* This has been shown to provide very good dynamic resilience [2].

In a ring, there is a strict notion of ordering among all the peers, and the key-space partition these peers are responsible for. This strict ordering is exploited in

[1] Recall in terms of information retrieval terminology.

[2] The technical report version of the original Chord proposal makes a passing remark on the merger of isolated overlays. SkipNet [9] addresses a very special case of network partitioning which is not usable in the general case.

defining the overlay topology and keeping it connected and to guarantee routing. In contrast, structural replication explicitly allows multiple peers to be responsible for the same key-space partition.

By ***structural replication*** we mean that (i) multiple peers can be responsible for the exactly same key space partition, i.e., these peers are mutual replicas; and (ii) each peer has multiple (functionally redundant) routing references which reduce the distance between source and destination by the same magnitude (probabilistically). CAN [14] uses similar key-space partition replication and calls it zone replication.

## 2 Background

In recent years the concept of structured overlays have attracted a lot of attention because of its potential to become a generic substrate for internet scale applications - as diverse as locating resources in a wide area network in a decentralized manner, address independent and robust and flexible (group) communication - e.g., application layer multicast and internet indirection infrastructure and content distribution network to name a few.

Structured overlay networks comprise of the three following principal ingredients:

(i) Partitioning of the key-space (say an interval or circle representing the real number between the range $[0, 1]$) among peers, so that each peer is responsible for a specific key space partition. By being responsible, we mean that a peer responsible for a particular key-space partition should have all the resources which are mapped into keys which are in the respective key-space partition.

(ii) A graph embedding/topology among these partitions (or peers) which ensures full connectivity of the partitions, desirably even under churn (peer membership dynamics), so that any partition can be reached from any partition to any other - reliably and preferably, efficiently.

(iii) A routing algorithm which enables the traversal of messages (query forwarding), in order to complete specific search requests (for keys).

Various applications can use transparently the (dynamic) binding between peers and their corresponding key-space partitions as provided by the overlay for resource discovery and communication purposes in a wide area network.

A structured overlay network thus needs to meet two goals to be functionally correct:

(i) *Correctness of routing*: Starting from any peer, it should be possible to reach the correct peer(s) which are responsible for a specific resource (key).

(ii) *Correctness and completeness of keys-to-peers binding*: Any and all peers responsible for a particular key-space partition should have all the corresponding keys (and none other).

Correctness of routing is achieved by maintaining the peers' routing tables correctly and using a proper routing algorithm. Correctness and completeness of binding is achieved by moving the corresponding keys (content) as and when the partition a particular peer is responsible for changes, and synchronizing the content among replica peers.

Most structured overlays use the ring topology, or a hybrid one [16], relying on a strongly connected ring for functional correctness of routing in the overlay, while the rest of the connections among peers provide optimization, i.e., efficiency. Replication is done in immediate neighbors on the ring and hence is deterministic as long as the ring is maintained correctly. This ensures the correctness (and completeness) of keys-to-peers binding. Hence, in our case study, one candidate we consider is the Chord [17] overlay - which not only pioneered the ring topology in the context of structured overlays, but also is one of the most extensively studied and developed system.

Both because of the well developed algorithms of Chord to maintain the ring topology, as well as the relative ease in doing so and better static resilience than other topologies [8] of the ring topology, it is predominantly used in other overlays.

We'll show that when it comes to merger of two networks, the reliance on the ring for functional correctness of the overlay is in fact a liability - hard to achieve - very slow and costly in terms of communication complexity and in terms of the required coordination. Merger of two ring based networks disrupt completely the operations of the overlay, and hence the functioning of other applications and services using it.

The other overlay we consider is the P-Grid [3] network, which apart using prefix based routing [13] uses an (almost) unique feature - *structural replication* - in order to provide resilience against churn [2], instead of relying on the ring invariant used by most other overlays. With the use of structural replication, the correctness of routing is never violated even under network mergers. However completeness of keys-to-peers binding is harder to achieve. Thus merger of structurally replicated overlays is graceful because the applications running on top of the original overlays will always have access to all the resources they had access to in their isolated overlays, but discovery (and hence, access) to resources from the originally other overlay will be possible only when the peers (replicas) for the corresponding key-space partition(s) have synchronized.

We'll like to emphasize that this case study is not a quantitative evaluation to come out with a final verdict on any specific overlay - each of Chord and P-Grid networks have many nice, sometimes complimentary features as well as shortcomings - making each better suitable for different application requirements. The essential goal here is to explore the design space to better identify the features of overlay networks that can either facilitate or hinder merger of overlays - and hence get a better insight for (re-)designing such systems.

## 3 Network Merger Case-Study: Chord

### 3.1 Chord (Recapitulation)

Chord uses SHA-1 based consistent hashing to generate $m$-bit identifier for each peer $p$, which is mapped onto a circular identifier space (key-space). Irrespective of how the peers' identifiers and keys are generated in a ring based topology, what is essential is that the peer identifiers are distinct. Similarly, unique keys are generated corresponding to each resource. Each *key* on the key-space is mapped to the peer with the least identifier greater or equal to *key*, and this peer is called *key*'s successor. Thus to say, this peer is responsible for the corresponding resource.

What is relevant for our study is how keys from the key-space are associated with some peer(s) and how the peers are interconnected (in a ring) and communicate among themselves.

**Definition 1.** *A ring network is (1) weakly stable if, for all nodes $p$, we have $predecessor(successor(p)) = p$; (2) strongly stable if, in addition, there exists no peer $s$ on the identifier space where $p < s < q$ where $successor(p) = q$; and (3) loopy if it is weakly but not strongly stable.*

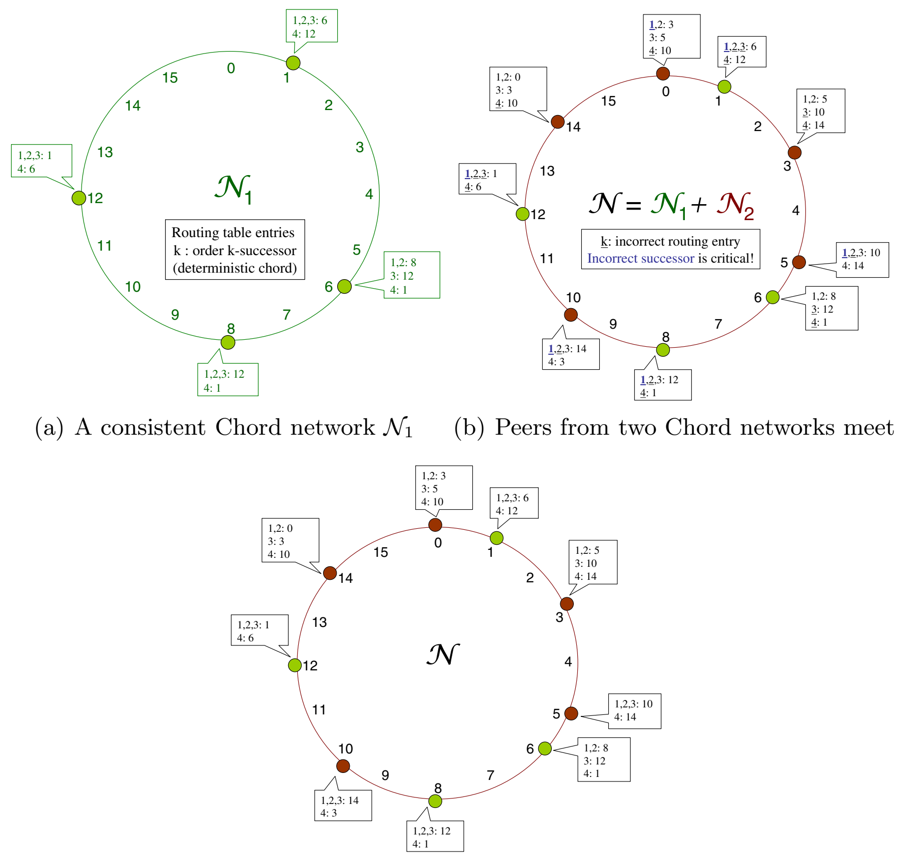
Condition (2) that there exists no peer $s$ on the identifier space where $p < s < q$ if $p$ and $q$ know each other as mutual successor and predecessor determines the correctness of the ring structure. Figure 1(a) shows one such consistent ring structure (peer's position in the ring and its routing table). The order-1 successor known also just as "successor" of each peer is the peer closest (clock-wise) on the key-space.

If at any time such a $s$ joins the system, the successor and predecessor information needs to be corrected at each of $p$, $q$ and $s$. Maintaining the ring is basically to maintain the correctness of successors for all peers - this in turn provides the functional correctness of the overlay - i.e., successor peer for any identifier *key* can be reached from any other peer in the system (by traversing the ring). For redundancy, $f_s$ consecutive successors of each peer is typically maintained, so that the ring invariant is violated only when any $f_s$ consecutive peers in the identifier space all depart the system before a ring maintenance mechanism - Chord's self-stabilization algorithm - can amend for the changes.

In addition to the successor/predecessor information, each peer maintains routing information to some other distant peersin order to reduce the communication cost and latency. It is the way these long ranges are chosen which differ in many ring topology networks. It has no critical impact on the functional correctness of the overlay. The original Chord proposal advocated the deterministic use of the successor of the identifier $(p + 2^{k-1})$ modulo $2^m$ as an order-$k$ successor of peer $p$ or a finger table entry.Many other variants for choosing the long range links exist - e.g., randomized choice from the interval $[p + 2^{k-1}, p + 2^k)$ or exploiting small-world topology [11,4] to name a few.

**Ring Self-stabilization Highlights.** The ring invariant is typically violated when new peers join the network, or existing ones leave it. If such events occur simultaneously at disjoint parts of the ring, the ring invariant can easily be reestablished using local interactions among the affected peers. Note that these events do not lead to a loopy state of the network.

Apart looking into the simple violations of the ring invariant which are relatively easily solved, the original Chord proposal (technical report version) also provides mechanisms to arrive at a strongly stable network starting from a loopy



(a) A consistent Chord network $\mathcal{N}_1$ (b) Peers from two Chord networks meet

(c) Ideal Chord network comprising peers from both networks

**Fig. 1.** When (peers from) two ring-based overlays meet

network (whichsoever reason such a loopy state is reached). We summarize the results of stabilizing a loopy network here.

Any connected ring network with $N$ peers becomes strongly stable within $O(N^2)$ rounds of strong stabilization if no new membership changes occur in the system. Starting from an arbitrary connected state with successor lists of length $O(logN)$ if the failures rate is such that at most $N/2$ nodes fail in $\Omega(logN)$ steps then, whp, in $O(N^3)$ rounds, the network is strongly stable.

## 3.2 Merger of Two Ring Based Networks

Consider two Chord networks $\mathcal{N}_1$ and $\mathcal{N}_2$ with $N_1$ and $N_2$ peers respectively (e.g., shown superimposed in Figure 1(b)).

**When peers from different overlays meet:** When peers from the two different overlays meet each other (by whatsoever reason - accidentally or deliberately), in a decentralized setting there is no way for them to ascertain that they belong to two completely different systems. This is so because overlay construction always relies on such peer meetings to start with. As a consequence, if the peer pair that meets have identifiers such that they would replace their respective successor and predecessor, then they will indeed do that.For our example from Figure 1(b) lets say peer 1 from $\mathcal{N}_1$ meets peer 0 from $\mathcal{N}_2$. Then peer 1 will treat 0 as its new predecessor, and 0 will treat 1 as its new successor, instead of 12 and 3 respectively. However, if they only change their local information, then the ring network will no more be strongly stable (may in-fact not even be weakly stable). In-fact such a reconfiguration will need and lead to a cascading effect, so that all members of both the original network try to discover the appropriate immediate neighbors (successor/predecessor) - requiring coordination among all the peers.

**Estimation of the probability that a peer's predecessor changes:** From the perspective of any peer in $\mathcal{N}_1$, the successor will change, if at least 1 out of the $N_2$ peers have identifier within the next $1/N_1$ stretch of the key-space (for which its present successor is responsible, on an average). Any particular peer from $\mathcal{N}_2$ has an identifier for this stretch with probability $1/N_1$. The number of peers falling in this stretch is thus distributed as $Binomial(N_2, 1/N_1)$, which approaches to a Poisson distribution with expectation $N_2/N_1$ as $N_2 \rightarrow \infty$. Hence, a peer from $\mathcal{N}_1$ will have its successor unchanged with probability $e^{-\lambda_1}$ where $\lambda_1 = N_2/N_1$. Thus each of the $N_1$ peers will have their successor node changed with a probability $1 - e^{-\lambda_1}$ i.i.d. Peers in $\mathcal{N}_2$ will be affected similarly with a parameter $\lambda_2 = N_1/N_2$ (symmetry).

**Estimate of the number of peer pairs which will have their immediate neighbors (either successor and/or predecessor) changed:** The number of peers whose successor will change in $\mathcal{N}_i$ is then distributed binomially $Binomial(N_i, 1 - e^{-\lambda_i})$ for i =1,2. Hence the expected number of nodes which will need to correct their successor nodes (and predecessor nodes) is $N_1(1 - e^{-\lambda_1}) + N_2(1 - e^{-\lambda_2})$.

The basic idea of how the ring can be reestablished is that when two peers from different networks meet so that they replace each other's successor and predecessor (immediate neighbor), then this information needs to be communicated to the original immediate neighbors, and the process continues.

There are several combinations of how the neighborhoods of the peers are affected after their interaction, each of which needs to be accounted for the actual network merger algorithm. Moreover, different combinations of faults (single or multiple peers crashing or leaving) can happen during the ring merger, and these too need to be dealt with. The specifics of such algorithms, and evaluation of the actual ring network merger algorithm is currently underway.

We'd like to admit at this juncture that without proper and exhaustive evaluation of the exact algorithms for merging two ring networks, it is difficult to see whether a strongly stable ring can be directly achieved, or whether a sequence of faults during the merger of two rings can even lead to a loopy network, which would then require even more effort to converge to a strongly stable state using Chord's already existing self-stabilizing mechanisms.

Thus the back of the envelope analysis above just provides the expected lower-bound of the ring reestablishment process in terms of correction of successor/predecessors. The latency of such a process started because of two peers from the two networks will be $O(N_1 + N_2)$ even in there is no membership changes during the whole merger process - this is the time required to percolate the information that the ring neighborhood has changed and to discover the correct neighbor when peers from both the original networks are considered together.

**Ring Loses Bearing During the Merge Process.** Above we provided a sketch of how to only reestablish the ring topology - which only guarantees the functional correctness of the routing process - i.e., the query will be routed to the peer which is supposed to be responsible for the key-space to which the queried key belongs.

Reestablishing the ring will be necessary in order to be able to query and locate even the objects which were accessible in the original network of any individual network. Hence, such a merger operation of ring topology based overlay will typically cause a complete interruption of the overlay's functioning.[3]

**Managing Keys on the Merged Ring.** Establishing the ring in itself is however not sufficient in an overlay network based index. In order to really find all keys (which originally existed in at least one of the two networks) from any peer in the merged network, it will still be necessary to transfer the corresponding key/value data to the "possibly" different peer which has become responsible for the new overlay. To make things worse, in a ring based network the queries will be routed to the new peer which is responsible for a key, so that even after the reestablishment of the ring itself, some keys that could be found in the original networks may not be immediately accessible, and will need to wait until the keys are moved to the new corresponding peer.

Lets consider that before the networks started merging, network $\mathcal{N}_i$ had key set $\mathcal{D}_i$ such that $|\mathcal{D}_i| = D_i$. Furthermore, if we consider that $\alpha$ fraction of the keys in the two networks is exclusive, that is $|\mathcal{D}_1 \cap \mathcal{D}_2| = \alpha|\mathcal{D}_1 \cup \mathcal{D}_2|$, then on an average, if a $\mathcal{N}_i$ node's successor changes, it will be necessary to transfer on an average $\alpha$ fraction of the data from network $\mathcal{N}_j$'s $\frac{1}{N_1+N_2}$ stretch of the key-space. Thus, on an average, the minimum[4] required transfer of unique data from members of original networks $\mathcal{N}_j$ to $\mathcal{N}_i$ will be $D_{tx}^{j \to i} = N_1(1 - e^{-\lambda_1})\alpha\frac{D_2}{N_1+N_2}$.

Apart from assigning the data corresponding to a key on the key-space to the peer which is the successor for that key, ring based topologies provide fault-tolerance by replicating the same data at $f$ consecutive peers on the ring.[5]

[3] We'd like to note that such a vulnerability may expose ring topologies to a new kind of "throwing rings into the ring" distributed denial of service (DDoS) attack, though the implications of such an attack and the amount of resources an adversary will require to make such a DDoS attack needs to be studied in greater detail.

[4] The actual implementation of such a data transfer will need to identify the distinct data in the two networks and transfer only the non-intersecting one, in order to achieve this minimal effort. This is an orthogonal but important practical issue that any implementation will need to look into.

[5] The parameter $f$ is a predetermined global constant determined by the system designer.

Given the strict choice of $f$ as neighborhood changes, the transferred data will in-fact have to be replicated at the precise $f$ consecutive peers of the merged network, determining the actual minimal bandwidth consumption. Similarly, some of the original $f$ replicas will need to discard the originally replicated content.

## 4 Network Merger Case-Study: P-Grid

### 4.1 The P-Grid Routing Network

P-Grid divides the key-space in mutually exclusive partitions so that the partitions may be represented as a prefix-free set $\Pi \subseteq \{0,1\}^*$. Stored data items are identified by keys in $\mathcal{K} \subseteq \{0,1\}^*$. We assume that all keys have length that is at least the maximal length of the elements in $\Pi$, i.e.,

$$\min_{k \in \mathcal{K}} |k| \geq \max_{\pi \in \Pi} |\pi| = \pi_{max}$$

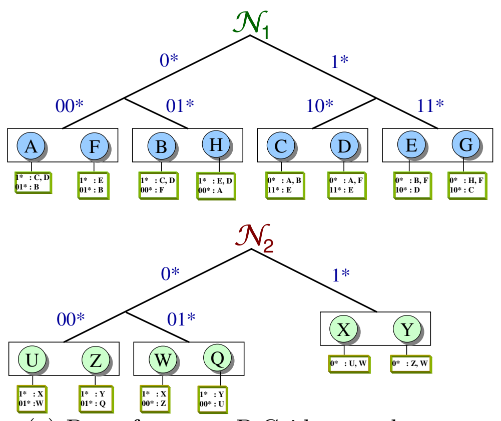
Each key belongs uniquely to one partition because of the fact that the partitions are mutually exclusive, that is, different elements in $\Pi$ are not in a prefix relationship, and thus define a radix-exchange trie.

$$\pi, \pi' \in \Pi \Rightarrow \pi \nsubseteq \pi' \wedge \pi' \nsubseteq \pi$$

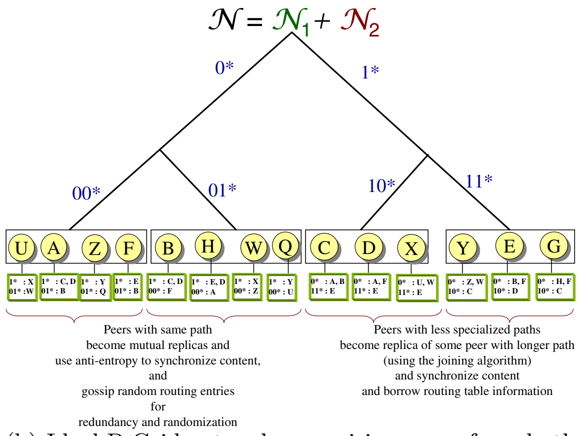
where $\pi \subseteq \pi'$ denotes the prefix relationship. These partitions also exhaust the key-space, so to say, the key-space is completely covered by these partitions so that each key belongs to one and only one (because of exclusivity) partition.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree, and each leaf has at-least one peer associated to itself. Each leaf corresponds to a binary string $\pi \in \Pi$, also called the *key-space partition.* Thus each peer $p$ is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length $l$ a set of references $\rho(p,l)$ to peers $q$ with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string $\pi$ with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The whole routing table at peer $p$ is then represented as $\rho(p)$ Moreover, the actual instance of the P-Grid is determined by the randomized choices made at each peer for each level out of a much larger combination of choices. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree. This also bounds the search time and communication cost. Figure 2(a) shows instances of P-Grid network (peer's path and routing table). e.g., in $\mathcal{N}_1$, peers $A$ and $F$ are mutual replicas and are responsible for the key-space with prefix 00. Peer $A$'s routing table comprise of peers $C$ and $D$ from the partition with prefix 1 and peer $B$ from the partition with prefix 01.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $\kappa(d)$ is calculated using an order-preserving hash function. $\kappa(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a



(a) Peers from two P-Grid networks meet


$\mathcal{N} = \mathcal{N}_1 + \mathcal{N}_2$
0*
1*
00*
01*
10*
11*
U
A
Z
F
B
H
W
Q
C
D
X
Y
E
G
Peers with same path become mutual replicas and use anti-entropy to synchronize content, and gossip random routing entries for redundancy and randomization
Peers with less specialized paths become replica of some peer with longer path (using the joining algorithm) and synchronize content and borrow routing table information

(b) Ideal P-Grid network comprising peers from both networks

**Fig. 2.** When (peers from) two structurally replicated overlays meet

peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (*structural replication*). $\Re(\kappa)$ represents the set of peers replicating the object corresponding to key $\kappa$. Peers additionally also maintain references to peers with the same path, i.e., their replicas $\Re(\pi(p))$, and use epidemic algorithms to maintain replica consistency. Routing in P-Grid is greedy.

### 4.2 Merger of Two Structurally Replicated Networks

**When peers from different overlays meet:** A resulting merged overlay network when peers from networks $\mathcal{N}_1$ and $\mathcal{N}_2$ meet is shown in Figure 2(b).

If peers from the two different networks meet, so that their paths are exactly the same (for example peers $A$ and $U$ from networks $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively in Figure 2(a)), then they will execute an anti-entropy algorithm to reconcile their content and become mutual structural replicas. In fact, such an anti-entropy algorithm will have to be run among all the other structural replicas of that part of the key-space too, and eventually of the other parts as well. However, since the original members of each network still retain the original routing links, routing functionality is not affected - and whichever keys were originally accessible will continue to be accessible. So to say, peer $C$ will always be able to access all the keys/content available at $A$ before the merger process. The keys from the same key-space which were present in the other network would however be available only after the background replication synchronization has completed. That is to say, a resource available originally only in $\mathcal{N}_2$ at $U$ and $Z$ (but with the same prefix 00 as $A$) will be visible to $C$ only when $A$ has synchronized its content with any one of $U$ or $Z$.

Use of structural replication has an additional downside - by not limiting the number of replicas nor having a proper structure among the replicas, it is difficult to have knowledge of the full replica subnetwork at each peer, and hence updates and replica synchronization is typically probabilistic. In contrast, once the ring is reestablished, replica positions are deterministic and hence locating replica is trivial in ring based topologies. Having discovered a replica, the anti-entropy algorithm itself (is an orthogonal issue) and hence the cost of synchronization of a pair of peers will be the same.

When two peers from $\mathcal{N}_1$ and $\mathcal{N}_2$ meet so that one's path is strictly a prefix of the other peer' path, then the peer with shorter path can execute a normal network joining algorithm [3] - extending its path to replicate either the peer it met, or a peer this peer refers it to. For example, $Y$ may extend its path from 1 to 11. In order to do so, $Y$ will need to synchronize its content with one of the peers which originally had the path 11, say $G$. Moreover $Y$ will need to obtain routing reference to a peer responsible for the path 10 (e.g., peer $C$) - information it can obtain from $G$ itself.

Since new peers join as structural replica or existing peers, no other existing peer need necessarily to update their routing table for routing functionality (unlike in a ring based topology). Thus, peer $Q$ referring to $Y$ for prefix 1 continues to refer to it as such, and any query 10 from $Q$ is routed first to $Y$, which then forwards it to - say $C$. Peers may however, over time add more routing entries, for instance, $Q$ adding a reference to $D$ for redundancy in its routing table for the prefix 1. Such changes however is a normal process in the P-Grid network and can be carried on in the background, again without interrupting the functioning of the overlay (and in fact instead making it more resilient to faults and churn).

Consequently, neither joining peers, nor merger of two existing overlay network disrupt the available functionality of the network members.[6]

[6] Note that the above discussion is true only for write once and then onwards read-only data, since for read/write, it will be necessary to maintain the replicas more pro-actively.

If peers with different paths meet each other, they need to do nothing, though they can refer each other to peers which are most likely to have the same path (similar to ring based topologies which can forward the peers closer to their respective key-spaces).

**Managing Keys in the Merged Network.** The amount of data that needs to be transferred from each system to the other is essentially the non-intersecting data. However, there is no need to transfer data from one peer to another merely because the key-space partition a peer is responsible for changes - because with structural replication, new peer joins or network mergers do not in itself automatically change the network's structure.[7]

The important thing to reemphasize is that a peer always finds the keys it could find before the merger process began, irrespective of the state of the merger process. Hence the replica synchronization can be done as a slow background process - hence the performance and network usage is also graceful - that is, merger of two overlays does not suddenly overburden the physical network's resources nor disrupt the functioning of the overlay networks. Such a graceful merger of existing networks also facilitates highly parallelized overlay construction [3] in contrast to the traditional sequential overlay construction approaches.

## 5 Related Work

There is very little specifically looking into network partitioning issues of overlays. The only system which explicitly discusses the network partitioning issue is SkipNet [9]. SkipNet is based on ring topology and imposes a restriction on the peers' identifiers in that nodes from same administrative domain get contiguous exclusive stretch of the overlay. Thus if the domain gets partitioned, all the nodes are contiguous on the identifier space and thus reconstructing the two rings is relatively easy. However such restrictions have serious implications on the general purposes in which the SkipNet can be used. Particularly their approach does not in any way solve the problems faced by most other overlays. Following their principles, if private overlays are first formed, either these overlays will use a minuscule portion of the key-space - which is unrealistic - particularly if its not known if and when new overlays need to be merged, otherwise, each of these private overlays will again overlap on the key-space (not be contiguous), and thus we'd again have the ring reestablishment problem as studied in this paper. Moreover, in order to find a resource, the query needs to know not only the resource name, but also the domain it can be found - which may be fine for specific applications but is very restrictive in general.

Canon [7] proposes organization of isolated overlays in a hierarchical fashion, and requires merger of the rings. However, they have not investigated the

[7] Local view of the structure however changes when a peer with shorter path meets a peer with longer path, and extends its own path according to the network construction algorithm [3], as explained above.

complexity of ring mergers.[8] Moreover, placement of keys in Canon is again either domain specific, so that peers from different domains may/may not find the keys. Such ambiguity (no guarantees on recall!) severely limits applications for which Canon may be used.

## 5.1 Network Dynamics

Merger of two overlays, seen from the perspective of individual peers (which is how these decentralized systems operate) can look very similar to simple membership dynamics, churn!

However, churn is a gradual process, and the system needs to continuously perform repair operations to rectify local view of peers in order to deal with the continuous membership changes. When two isolated networks need to merge, the magnitude of the population change with respect to their original population size is very high. In fact, if we consider merger of two same-sized overlays, it'd essentially mean a vanishing half-life [10], without giving any time to the usual overlay maintenance operations to deal with the changes in the network. This also means that mechanisms other than what are employed to deal with normal churn needs to be developed. Even so, we try to reuse tools and ideas already honed in dealing with churn.

At this juncture, it is also worthwhile to point out that since merger of two networks is indistinguishable from normal churn from the perspective of individual peers, it is not obvious when to use the network merger algorithms.

## 5.2 Portability

Over the last few years, there has been tens (possibly even running close to a hundred) different topologies defined for structured overlays, of which some have seen sustained development and deployment - including Chord [17], Kademlia [12], Pastry/Bamboo [16,15] and P-Grid [3]. Nonetheless, these are diverse systems using different protocols. To that end, there has been effort to identify common APIs [1,6,15] that can make development of applications modular and independent of the underlying overlay infrastructure.

Intercommunication of peers based on different implementations and protocols has been relatively better explored - possibly because of the proliferation of numerous overlay topology proposals as well as different implementation/protocols of theoretically equivalent networks - leading to these rather numerous tentative proposals for universal APIs.

The problem of merger of two distinct overlays (but using the same protocols) is somewhat different from how peers across different overlay networks using different protocols can communicate among themselves, or how applications can be developed and run transparently on any overlay substrate.

[8] The simulations they have seem to have merged the rings automatically. One can only speculate that most likely the new rings were formed using the global knowledge of the simulator in a manner a centralized system would. To that end, our paper exposes the complexity of achieving ring mergers.

## 6 Summary and Conclusion

This paper is a first step in looking at the problem of merging two separate overlay networks. Depending of the peculiarities of a specific overlay - whether it relies on the strong stability of the ring, or whether it uses structural replication, we identified the mechanisms which are necessary to execute the merger process, indicated the minimal effort it will require in order to merge two networks as well as discussed the specific challenges that need to be met for practical implementation of such an approach.

For managing data in any merged networks efficiently, it is necessary to use an efficient anti-entropy algorithm, so that ideally non-intersecting data items are identified efficiently and only those are exchanged among peers.

We also observed that the ring based networks can not function at all until the whole merge process is complete. This may have serious consequences of usability of ring based topologies, particularly if merger of two such overlay networks becomes necessary.

In contrast, use of structural replication instead of the ring as a fault-tolerance mechanism makes overlay mergers graceful. However, location of all structural replicas may be tricky, and hence replica management in such a system is relatively more complex than in ring based systems which has deterministic choice of the location and size of the replica subnetwork.

The actual algorithmic details of merger of ring based overlays is still under study and refinement, particularly looking into the potentially catastrophic combination of faults that may occur during the long latency incurred in merging two rings. Quantitative and comparative evaluation of network merger for ring based networks and structurally replicated networks, along with precise finalized algorithms to merge ring based networks thus is part of our ongoing and future work.

## References

1. K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, M. Hauswirth, and S. Haridi. The essence of P2P: A reference architecture for overlay networks. In *P2P2005, The 5th IEEE International Conference on Peer-to-Peer Computing*, 2005.
2. K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.
3. K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. *31st International Conference on Very Large Databases (VLDB)*, 2005.
4. A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM*, 2004.
5. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One Ring to Rule Them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks. In *ACM SIGOPS European Workshop*, 2002.
6. F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common API for structured peer-to-peer overlays. In *IPTPS 2002*.

7. P. Ganesan, P. K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *ICDCS*, 2004.
8. K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the ACM SIGCOMM*, 2003.
9. N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS 2003*, Seattle, WA, March 2003.
10. D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems, 2002.
11. G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS*, 2003.
12. P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, 2002.
13. C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distribute d Environment. In *SPAA*, 1997.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM*, 2001.
15. S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *SIGCOMM*, 2005.
16. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
17. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM, (Technical report version: http://pdos.csail.mit.edu/chord/papers/)*, 2001.
18. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

# Self-protection in P2P Networks: Choosing the Right Neighbourhood

Ivan Martinovic[1], Christof Leng[2], Frank A. Zdarsky[1], Andreas Mauthe[3], Ralf Steinmetz[4], and Jens B. Schmitt[1]

[1] Distributed Computer Systems Lab, University of Kaiserslautern, Germany
[2] Databases & Distributed Systems, University of Technology Darmstadt, Germany
[3] Infolab 21, University of Lancaster, UK
[4] Multimedia Communications Lab, University of Technology Darmstadt, Germany

**Abstract.** In unstructured peer-to-peer networks, as in real life, a good neighbourhood is not only crucial for a peaceful sleep, but also for an exchange of important gossips and for finding good service.

This work investigates self-protection mechanisms based on reputation in unstructured peer-to-peer networks. We use a simple approach where each peer rates the service provided by others and exchanges the collected knowledge with its direct neighbours. Based on reputation values peers manage their connections to direct neighbours and make service provisioning decisions.

To quantify the impact of our proposed scheme, we implement a simple protocol in a fully unstructured peer-to-peer network. We show that free riding and the impact of malicious peers trying to poison the network with bad files is minimised. Furthermore, we show that a good neighbourhood protects peers from selecting bad files, while free riders suffer in a bad neighbourhood of malicious peers.

**Keywords:** Self-protection, Peer-to-Peer, Trustworthiness, Reputation, Free-Riding, Network Poisoning.

## 1 Motivation

Network poisoning is a problem where malicious peers try to upload invalid files into a peer-to-peer network. Although standard cryptographic solutions like hash functions can help us to check the integrity of files, the impact of network poisoning has increased dramatically in the last few years. The reason is that in a decentralised, large-scale network where content is provided by the users themselves, one cannot easily transfer techniques from centralised content delivery environments. Many peer-to-peer networks have lost their popularity because the content provided within those networks is mostly malicious (infected by virus) or invalid, wasting the users' time and bandwidth. On the other hand, there is an increasing number of decentralized systems using reputation schemes as security measure to protect users and reward cooperative behaviour.

Recently, various research contributions studied the impact and strategies of network poisoning and analyzed its magnitude within P2P networks [14,5]. In this work, we analyse the concept of self-protection against network poisoning and free riding within unstructured peer-to-peer networks by using concepts from soft-security and traditional security. We model our peer-to-peer network as a basic Gnutella network [12] without any structure and extend it only by a simple protocol we call Simple Trust Exchange Protocol (STEP), which enables peers to decide whom to connect to as direct neighbours and to whom to provide service. As the result, a trust-based construction of an overlay network is accomplished by local decisions.

To avoid the extreme case of individual “learning-by-doing” STEP provides a mechanism to exchange knowledge between its direct neighbours, which helps a neighbourhood to avoid selecting bad service or potentially malicious neighbours. Moreover, as the search in unstructured networks is mostly based on different flooding algorithms, having a good neighbourhood becomes essential for a peer’s own search success.

Although scaling problems of the original Gnutella network are well-known, we take advantage of a fully unstructured network to avoid “deus ex machina” intervention and to support the evolutionary growth of topology. Furthermore, the concept introduced in this work does not depend on the flooding approach of Gnutella and could be adapted to different routing mechanisms (e.g. Ultrapeers).

## 2 Simple Trust Exchange Protocol (STEP)

To support service rating and exchange of knowledge among neighbouring peers, we introduce Simple Trust Exchange Protocol (STEP), the pseudo-anonymous token-based protocol. A token is a mutually signed transaction receipt which serves as a proof of service provision between a consumer and a provider (in all our scenarios we consider a service to be a file transfer between a provider and a consumer, but clearly, other scenarios are also conceivable). To participate in the system, every peer needs to create a public-private key-pair as its identity. Because the identity has no ties to the outside world and only becomes meaningful by a token signed with it, there is no need for any public key infrastructure.

To discourage the change of identity, in this work we follow the idea of "no profit to newcomers" [13], where every new peer gets the minimal possible reputation.

**Table 1.** Token Structure

| Name | Description |
|---|---|
| CID | Consumer’s identity |
| PID | Provider’s identity |
| TS | Timestamp of token creation |
| Length | Length/Duration of service (e.g. File size) |
| Rating | Consumer’s rating ({*good*,*bad*}) |
| CSig | Consumer’s Signature |
| PSig | Provider’s Signature |

STEP relies on a peer-to-peer network to discover and deliver services and only extends systems by a token creation mechanism and knowledge exchange between peers. The structure of a token is shown in Table 1.

## 2.1 Token Creation

A token is created upon a request of a service and is supplemented with rating after a transaction has finished. Upon the service request a consumer creates an initial token by filling the data fields except $Rating$ and signs it with his private key. The consumer sends the token and his public key to a provider. If the provider accepts the service request he also signs the initial token and sends the token and his public key back to the consumer. After the transaction the consumer rates the provider (filling the $Rating$ field) and finalizes the token by re-signing it. Finally, he sends a copy of the token to the provider.

The objective behind the consumer's double signing is to mitigate the incentives for not finalizing the token. For example, a free-rider could decide to reject finalization and publication of the token to avoid being detected as a consumer. In this case the initial token containing the free-rider's signature will be published by a provider. As a result, other peers will be able to collect tokens and if the provider is trusted the free-rider can still be detected.

## 2.2 Knowledge Exchange

Every peer can improve its own knowledge over another peer directly by requesting a service and then rating it, or indirectly by collecting other ratings about that peer and then using them to compute its own trust value.

Knowledge exchange between peers is realized through distribution of tokens within an overlay network and by storing the received tokens locally. We use the Gnutella-typical flooding approach for distributing tokens in a way similar to query for services. The $Knowledge$ message is wrapped in Gnutella $Query$ messages to provide maximum compatibility with legacy peers in mixed networks, analogous to the approach described in [8]. To enable the verification of a token, in case the local neighbours do not have the required keys, a peer forwarding the token appends the missing public keys of both, the consumer and provider, to the $Knowledge$ message.

## 2.3 Decision Making

The computation of a peer's reputation is a subjective matter and every peer can choose its own method for interpreting the collected tokens. In this work we focus on the impact of reputation and not on its calculation, which is why we use a simple approach of calculating the number of collected tokens where the provider was rated as good, and subtracting the tokens where the rating was bad. Although this mechanism does not deal with nodes that are actively trying to cheat the reputation system (such as attacks from malicious group of peers which can mutually sign tokens), STEP provides enough room for implementation of more sophisticated algorithms. For example, one peer can choose to follow the PGP's "Web-of-Trust" concept by considering the trustworthiness of both signatures, as well as by utilizing the transitivity of trust relationships (e.g. [4]) or more accurately computing reputation of token-based systems with a more complex attacker model (e.g. [10]). Furthermore, a very restrictive approach would be to base the trust computation only on a closed group od peers (similar concepts already exist as "darknets" which represent a closed group of members within a public peer-to-peer network).

**Search and Service Selection.** After executing a Gnutella query and receiving search results a peer can immediately compute a reputation of available provider peers based on its local knowledge. The consumer peer then selects a file with a probability proportional to the sum of reputations of all providers offering the same file [13].

Due to assigning a reputation with positive minimum to every peer, every provider has the chance of being selected. This method decreases the overloading of a few top providers and increases the chance of selecting a newly joined peer if it provides frequently searched files.

**Service Providing.** After selecting a provider, a consumer sends a service request. If there is enough free bandwidth for an upload connection, the provider grants the service immediately to maximize the utilization of its bandwidth. Otherwise the provider computes a reputation of the new consumer and compares it with the lowest reputation of those consumers already connected. If the reputation of the new consumer is higher than that of any already connected consumers, the connection with the lowest value is cancelled and replaced with a new one. Although it would be better to use a non-preemptive queue to finish the already started file transfer instead of cancelling a transaction in progress, Gnutella does not support such concept.

**Choosing the Neighbourhood.** In Gnutella, every peer has a minimum number of desired neighbours (typically 4 for real-world Gnutella networks). If its current number of neighbours is less then desired the peer actively tries to connect to more neighbours. Instead of choosing the neighbours arbitrarily as in the classic Gnutella a STEP peer tries to connect to neighbours with a reputation at least as high as its current neighbours. If no such peers are available an arbitrary peer is chosen.

Most of the peers, depending on their bandwidth, support more than the minimum number of neighbours and thus are able to accept incoming connection requests by new peers. In the classic Gnutella every peer with available connection slots accepts all incoming connection requests. In STEP only requesting peers with a reputation that is in average not lower than those of the current neighbours, are accepted. If the STEP peer has reached the maximum number of Gnutella connections, then, for every further connection request the reputation of a requesting peer is compared with the local neighbourhood. If a neighbour with a lower reputation is already connected, it will be replaced.

## 3 Experiments

Service mentioned in all of our scenarios is a file transfer and we have modelled 4 different peer profiles:

- *Cooperative Peer*: a peer that offers services to the network by sharing valid files,
- *Freerider:* an opportunistic peer that does not share any files but only requests services,
- *Foul Dealer*: a malicious peer with a high bandwidth and maximum number of files (highest probability to answer with $QueryReply$), which tries to upload invalid files only (network poisoning),
- *Good Dealer*: opposite of the Foul Dealer, an altruistic peer with a high bandwidth and a large number of valid files.

The profile of a *Good Dealer* is only for measurement purposes in order to better compare the impact of good and bad dealers within a STEP-enhanced and legacy Gnutella network.

## 3.1 Configuration

One of the advantages of using Gnutella for our investigation is the availability of rich empirical data. To provide realistic assumptions of the peers' online times and available bandwidths, we have used empirical distributions based on the Gnutella analysis from [16]. The file popularity distribution was also based on empirical data taken from [17] which analysed the Kazaa peer-to-peer network. Other simulation parameters are listed in Table 2.

**Table 2.** Simulation Parameters

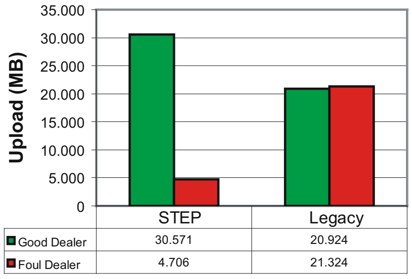
| | |
|---|---|
| Network Size | 2048, 4096 |
| Number of Files | 192.000 |
| Token Validity Time | 10 h |
| Query Interval | 2-4 min |
| Simulation Time | 80 hours |

The number of Free Riders was set to 50% for both network sizes and number of Foul Dealers and Good Dealers was set to 64. The bandwidth was divided into upload- and download streams which can be asymmetric to conform to a real peer-to-peer network with heterogeneous peers (e.g. DSL users with 1024 Kbit/s downstream and 128 Kbit/s upstream). The minimum bandwidth represent modem users with both upstream and downstream of 32 KBit/s and the maximum bandwidth is a broadband user having 10 MBit/s for both directions. Both, Foul Dealer and Good Dealers are assumed to have maximum bandwidth, where other peers follow empirical distributions from the real-world statistics.

As the goal of this work was not to provide highly sophisticated mechanisms to detect malicious behaviour we assume all peers conform to Gnutella and/or the STEP protocol with the exception that every peer rated as bad will not publish tokens. The service is rated as good if the file was sound and bad if not. Every peer makes a wrong rating (unintentionally) with a probability of 5%. In the initialization phase, when a new peer joins the network it randomly tries to connect to peers from a global peer cache. Because of a long transient phase of network initialization, we simulated 80 hours of network activity. This proved to be very important as the steady state was reached after approximately 15 hours. All simulations were conducted on an Athlon XP 2500+ with 1 GB RAM and one simulation run took approximately 24 hours to finish. The memory requirements of the simulation were 889 MB.

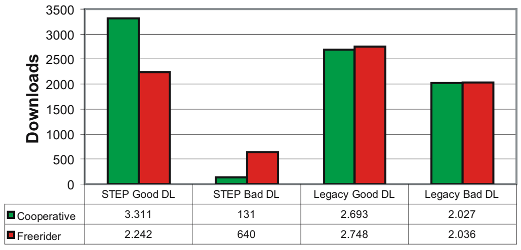
## 3.2 Results

**Network Poisoning.** In Figure 1 we analyse the impact of Foul Dealers and Good Dealers on network poisoning by comparing the number of successful uploads of bad files



| | STEP | Legacy |
|---|---|---|
| 🟩 Good Dealer | 30.571 | 20.924 |
| 🟥 Foul Dealer | 4.706 | 21.324 |

**Fig. 1.** Network Poisoning



| | STEP Good DL | STEP Bad DL | Legacy Good DL | Legacy Bad DL |
|---|---|---|---|---|
| 🟩 Cooperative | 3.311 | 131 | 2.693 | 2.027 |
| 🟥 Freerider | 2.242 | 640 | 2.748 | 2.036 |

**Fig. 2.** Download Scenario

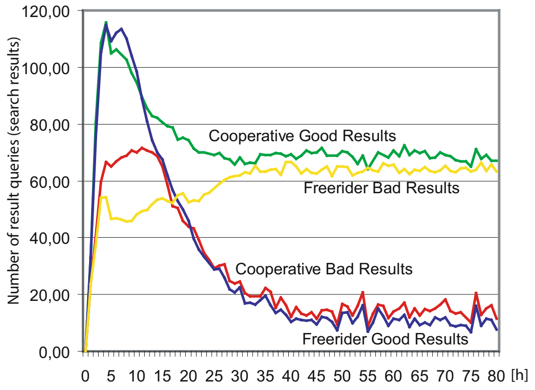


**Fig. 3.** Search Results

within both scenarios. As it can be seen, in the STEP-enhanced scenario the impact of Foul Dealers is decreased by 77% while the impact of Good Dealers is increased by 46%.

Figure 2 analyses the downloads between cooperative peers and free-riders, again for the two scenarios. The total number of good downloads (cooperative and free-riders) remains almost equal in both networks, but the number of good downloads of cooperative peers is almost 20% higher than for free riders. In a legacy network there is no incentive to behave cooperatively since all peers equally download good and bad files. Even more interesting is the fact that there is only 13% bad downloads within the STEP-enhanced network out of which 83% are performed by free-riders.

Furthermore, we measured the number of good and bad query results (a bad result is service offered by a foul dealer) for both cooperative and free-rider peers when they search for a specific file. As previously mentioned, the search in the Gnutella peer-to-peer network is a simple flooding. Due to the limitation of flooding by a query's TTL counter, it is important for a peer to be in a good neighbourhood. As it can be seen in Figure 3, the number of bad results for cooperative peers decreases dramatically, while the free-riders are losing good results and gaining more bad results. The overall number of query results is decreased after 15 hours as a result of neighbourhood clustering.

**Knowledge.** Figure 4 (left) shows local knowledge of a peer over all tokens in the network. As it can be seen, after 15 hours the local knowledge covers over 40% of all tokens existing in the network. The reason for its decrease after the first 15 hours is the clustering of the network, when the good neighbourhood has been established and tokens from the bad neighbourhood cannot reach most of the peers within the good neighbourhood. The Figure 4 (right) shows the accuracy of a peer's local knowledge about every other peer. As a result, although every peer knows only 40% of all tokens in the network it has on average 50% of all tokens associated to the peer when calculating its reputation. This is due to the limited horizon of query and knowledge exchange messages, both affected by the location of a peer in the overlay network in the same way. Thus, the received tokens will be more related to providers' and consumers' reputations actually calculated than tokens beyond the horizon, which never reached the peer.

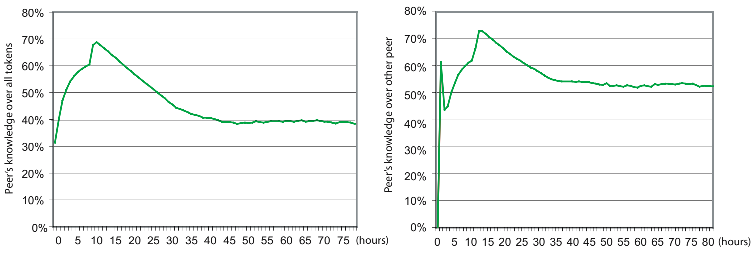


**Fig. 4.** Local Knowledge

**Topology.** Figure 5 shows topologies captured during the simulation times of 10h, 20h, 30h and 40h. After 40h the topology has stabilised and no significant changes occured afterwards. A clear separation of neighbourhoods can be seen after 25 hours of simulation where cooperative peers have distanced themselves from most of the Foul Dealers creating a neighbourhood with the Free Riders.

There are still many connections between the neighbourhoods, but due to the TTL scoping of $Query$ and $Knowledge$ messages most message exchanges remain inside

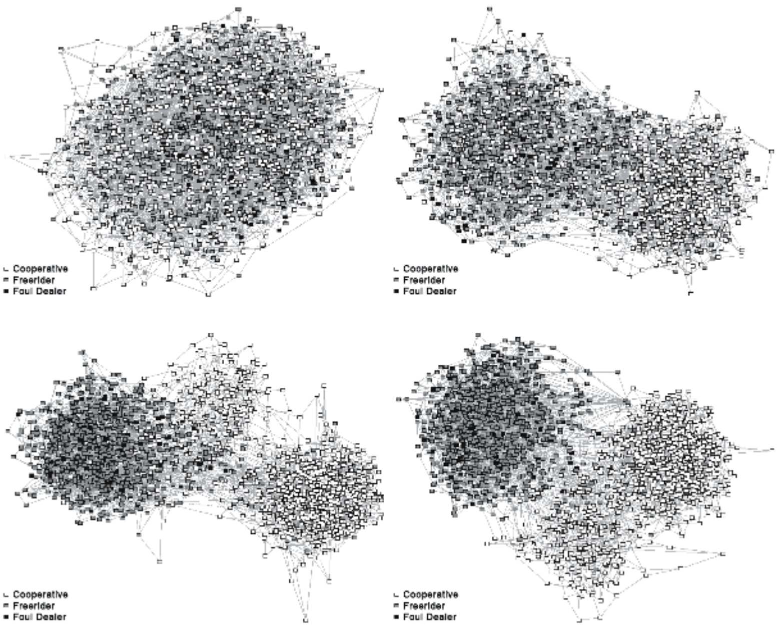


**Fig. 5.** Topologies after 10h, 20h, 30h and 40h

neighbourhoods, resulting in decreased number of query results from remote neighbourhoods. This also results in a decreased number of queries sent from free-riders to good peers (and thus self-protecting) but also from other good neighbourhoods.

**STEP Overhead.** The Overhead of STEP can be mainly attributed (98%) to $Knowledge$ messages as they contain tokens and public keys. In the worst case every token has 2 public keys and if the key length is set to 512 Bit every token costs 128 Bytes. Nevertheless we implemented a simple caching strategy where only new keys are forwarded to direct neighbours decreasing the cost to only 0.6 keys per token and still provide key distribution within fully unstructured peer-to-peer network. Furthermore, the length of an average knowledge message was 750 Bytes and maximally 1500 Bytes.

As the result, we can state that the price to pay for STEP is a 28% increase in bandwidth usage over standard Gnutella control message overhead; yet, its advantage is a dramatical decrease in the number of bad downloads and enhanced incentives for cooperative peers as they are now able to increase the utilization of their bandwidth with valid downloads which are preferred over those of the free-riders.

## 4 Related Work

The problems caused by selfish and malicious participants in peer-to-peer and ad hoc networks have been examined in several studies (e.g. [9,11,3,2]).

Probably the first to propose a practical solution based on traditional cryptography and structured overlay networks were Aberer and Despotovic [1] using the P-Grid system to store and retrieve complaints about misbehaving peers. The principle of "no profit to newcomers" together with other design goals for peer-to-peer networks (self-policing, anonymity, no profit to newcomers, minimal overhead, robust to malicious collectives) has been described by Kamvar et al [13]. They created the EigenTrust peer-to-peer reputation system according to those design goals. EigenTrust uses a distributed way of approximating the global trust value of a peer. Damiani et al [8] proposed a distributed reputation system as an extension to Gnutella called P2PRep, with a seamless migration path for existing Gnutella networks. Those integration concepts can be applied to our system in a similar fashion. Instead of using signed transaction receipts they rely on a live poll of user opinions, which has its own advantages and disadvantages. The usage of reputation as a trust building concept within a decentralised network of WLAN providers was described by Efstathiou and Polyzos in [10]. They apply the concept of a token-based reputation to a system they call a Peer-to-Peer Wireless Network Confederation (P2PWNC). For reputation calculation they use a maximum flow based algorithm called NWAY which is very resilient to cooperating groups of malicious nodes. The tokens in their system are exchanged by interacting parties, instead of the active publication mechanism we use.

Very interesting work on using trust to build an unstructured overlay network (Adaptive P2P Topologies) was introduced by Condie et al. [6]. Instead of a reputation system, they count only the subjective experience of every node itself. This minimizes the overhead of the protocol but also limits the available information on other peers. As the result of their adaptive topologies, the malicious peers and free-riders are pushed to the fringe of the network, which is similar to our work. However, the major difference is the usage of tokens to avoid distribution of only subjective reports (gossips). We considered the token mechanism to provide further advantages such as, choosing indivdual reputation algorithms and support of the Web-of-Trust concept in a peer-to-peer network where every peer can individually decide whose tokens can be more trusted based on the providers and consumers signatures.

Furthermore, tokens constitute the basis for coordination and control mechanisms as well as for pricing in commercial scenarios as described by Liebau et al. [15].

## 5 Conclusion and Future Work

This paper presents our initial work on trust in unstructured peer-to-peer networks. Our objective is to investigate into more depth both, the impact and efficiency of self-protecting mechanisms based on trust computation. We have chosen to use a completely unstructured peer-to-peer network as it allows us to investigate impacts on topology, as well as to support evolutionary growth of the network.

STEP defines a simple way to create transaction receipts and is adaptable to many different methods for knowledge exchange. The algorithm presented here adapts the Gnutella query method for maximum compatibility with legacy networks.

The Gnutella network is clearly not very scalable but many alternative routing mechanisms for message routing in peer-to-peer networks have been proposed in recent years. Alternatively, tokens can be accumulated at each intermediate peer and forwarded in regular intervals to reduce the number of messages needed. Also, as a part of our future work we intend to investigate how trust can be used for a better routing of search messages to mitigated overloading of trusted peers and to create incentives against free-riding. The idea of trust-based routing could help us to scale the efficiency of routing scheme based on trustworthiness of a peer or a neighbourhood.

Furthermore, if the service provided by the peers is the upload of shared content, the semantics of this content may also be used to achieve better routing of tokens, which leads to a more precise reputation calculation and consequently to a better reorganization of the overlay. It has been observed that content is normally clustered into categories with most peers being active only in very few categories [7]. Therefore, the interaction topology of such content distribution networks are typically highly clustered. This circumstance may be used to distribute tokens to peers which can make use of them more efficiently.

The STEP system is not only highly independent of the knowledge exchange method but also of the choice of reputation algorithms such that every peer can choose an algorithm independently. The choice of usable algorithms is very broad as every node can store a relevant proportion of the globally available knowledge about rated peers. To optimize the overall system performance the chosen algorithm should not only be resilient against attacks, but should also provide good results even with little aggregated data. To follow this idea, we consider not only token mechanisms, but also concepts of accumulated gossiping for building trust within peer-to-peer networks.

## References

1. K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *CIKM '01: Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 310–317, 2001.
2. S. Buchegger and J.Y. Le Boudec. Performance analysis of the CONFIDANT protocol. In *MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad hoc Networking & Computing*, pages 226–236, 2002.
3. S. Buchegger and J.Y. Le Boudec. A Robust Reputation System for Mobile Ad-hoc. In *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.
4. S. Capkun, L. Buttyán, and J-P. Hubaux. Self-Organized Public-Key Management for Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 2(1):52–64, 2003.
5. N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-peer Networks. In *EC '05: Proceedings of the 6th ACM Conference on Electronic commerce*, pages 68–77, 2005.
6. T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive Peer-to-Peer Topologies. *IEEE Transactions On Systems, Man and Cybernetics, Part A*, 35(3):385–395, May 2005.
7. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, October 2002.
8. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Managing and Sharing Servents' Reputations in P2P Systems. *IEEE Transactions on Data and Knowledge Engineering*, 15(4):840–854, July 2003.
9. P. Dewan and P. Dasgupta. Pride: Peer-to-peer Reputation Infrastructure for Decentralized Environments. In *WWW Alt. '04: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pages 480–481, 2004.

10. E. C. Efstathiou, P. A. Frangoudis, and G. C. Polyzos. Stimulating Participation in Wireless Community Networks. Technical report, June 2005.
11. M. Feldman and J. Chuang. Overcoming Free-riding Behavior in Peer-to-Peer Systems. *SIGcom Exch.*, 5(4):41–50, 2005.
12. Gnutella Developer Forum. The Annotated Gnutella Protocol Specification v0.4. http://rfc-gnutella.sourceforge.net/developer, 2006.
13. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 640–651, May 2003.
14. J. Liang, R. Kumar, Y. Xi, and K. Ross. Pollution in P2P File Sharing Systems. In *IEEE INFOCOM, Miami, FL, USA*, March 2005.
15. N. Liebau, V. Darlagiannis, A. Mauthe, and R. Steinmetz. Token-based Accounting for P2P-Systems. In *Proceeding of Kommunikation in Verteilten Systemen KiVS 2005*, pages 16–28, February 2005.
16. H. Luemkemann. Leistungsfaehige Verteilte Suche in Peer-to-Peer File-Sharing-Systemen. Master Thesis, CS Department, University of Dortmund, 2002.
17. S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, January 2002.

# Modelling the Population Dynamics and the File Availability in a BitTorrent-Like P2P System with Decreasing Peer Arrival Rate

Riikka Susitaival and Samuli Aalto

Helsinki University of Technology
P.O. Box 3000, FIN-02015 TKK, Finland
{riikka.susitaival, samuli.aalto}@tkk.fi

**Abstract.** Many measurement studies of P2P file sharing systems suggest that the request rate for a file changes over time and the system is thus non-stationary. For this reason we study the population dynamics and the availability of a file in a BitTorrent-like file sharing system, when the arrival rate for file requests decreases exponentially. We study the system first by a deterministic fluid model and then by a more detailed Markov chain analysis that allows estimating the life time of a single chunk exactly. Simple approximation for the life time is also derived. In addition, we simulate the life time of a file consisting multiple chunks in order to verify the analytical results to be applicable also to a more complex system.

## 1 Introduction

Peer-to-peer (P2P) applications, such as file sharing, have become a significant area of Internet communication in recent years. Older examples of these applications are Gnutella, Napster and Kazaa, whereas BitTorrent is currently the most popular file sharing system. It has been widely reported that P2P related traffic forms a major part of the total traffic in the Internet and the share is even increasing [1], [2].

The idea of BitTorrent is to divide the file into parts, named *chunks*, so that different parts can be downloaded from several peers simultaneously, where the size of the chunk is typically 256 KB. The technical details of BitTorrent are skipped in this paper but can be found in [3]. According to performance studies [4], BitTorrent is an effective P2P protocol and scales well even when the number of participating peers is very high. In this paper we concentrate on BitTorrent-like P2P protocol because of its popularity but the results are applicable to other similar protocols as well.

Measurement studies [5], [6], [7], have shown that the evolution of a single file in the system can be divided into three phases. In the first *flash crowd* phase the demand for the newly released file is high. The flash crowd phase is followed by a *steady state* in which the demand for the file and the service capacity of the system are in balance. Due to the decentralized manner of BitTorrent, there are not any guarantees that all chunks of the file are present in the system over time. If some chunk of the file is missing, the file is not complete anymore. Depending on application, this might or might not be crucial. In this paper we assume the former. This third phase of the system life time is called as *end phase*.

A few papers have analyzed P2P file sharing systems by stochastic models so far. Yang et al. divide the analysis of BitTorrent-like system into transient and steady state regimes [8]. The service capacity of the transient regime is studied by a branching process and the steady state by a Markov model. Paper [4] studies the performance of the system by a deterministic fluid model, whereas in paper [9] the network level latencies are modeled by the delay of a single class open queueing network and peer level latencies by the delay of M/G/1/K processor sharing queues. However, these early studies do not capture all aforementioned phases of the sharing process. The arrival rate is assumed to be constant, and especially, the file is assumed to be complete forever.

More detailed models for BitTorrent-like P2P systems are provided in [10] and [11]. Tian et al. study the integrity of the file in [10]. First, using Markov chain modelling, the steady state solution for the number of downloaders holding a certain number of chunks is derived and then the file availability is estimated by a simple probabilistic model. Drawback of the paper is that the steady state solution of the system is assumed to exist and the availability model is only indicative. Paper [11] first proposes that arrival rate decreases exponentially and studies then the system by a deterministic fluid model. The life time of the system, i.e., the mean time from the appearance of the file until its disappearance is derived by a simple deduction. However, the model is very coarse, and does not take the dowload/upload time of the file into account, for example. As a conclusion, the correct analytical models for file availability are missing and should be studied in more detail.

In this paper we analyze the population dynamics of the P2P file sharing system, when the demand for the file decreases exponentially. We use three approaches to study the scenario: deterministic fluid modelling, time-dependent Markov chain modelling and simulations. First we construct a deterministic fluid model for sharing a single piece of the file and study the dynamics of the average number of downloaders and seeds over time by solving the differential equations of the model.

The deterministic fluid model is, however, unable to capture all the details of the chunk sharing process such as possible unstability and extinction of the system. For this reason we construct a complete non-homogenous Markov chain model to obtain more information of the life cycle of chunk sharing process. From the Markov chain model we are able to calculate the mean absorbtion time and thus evaluate the life time of the chunk sharing system more precisely than in papers [10], [11]. We also derive approximations of the life time in some limiting scenarios and evaluate them by simulations using different parameter combinations.

As both the fluid model and the Markov chain model concentrate on the sharing single chunks independently of the other chunks, we study by simulations the

P2P file sharing system with multiple chunks. The obtained analytical results for the life time of a single chunk are compared to simulations of the corresponding system with multiple chunks.

We have studied the population dynamics of the P2P file sharing system already in paper [12]. In the previous paper we assumed that peer requests for the chunk arrive with a constant rate, whereas in this paper we let the peer arrival rate to change over time. Due to this fundamental difference in the assumptions concerning the arrival process, also the deterministic fluid models, Markov models and their solutions characterizing the population dynamics of the system are completely different in the papers.

As a summary, our main contributions in this paper are:

– We construct a deterministic fluid model for chunk sharing and analyze the number of the dowloaders and seeds over time.
– We develop a time-dependent Markov chain model for the number of seeds and downloaders and calculate the mean life time of the chunk sharing process. Also approximations are given.
– We verify the applicability of a single-chunk model to estimate the mean life of file sharing by simulating the system with multiple chunks.

The paper is organized as follows: In Section 2 we introduce a model for sharing a chunk with exponentially decreasing arrival rate. Section 3 studies the population dynamics of sharing of a chunk by a deterministic fluid model. In Section 4 we present an accurate result for the mean life time and in Section 5 we approximate it. Section 6 simulates the system with multiple chunks and finally, section 7 makes a short conclusion.

## 2 A Model for Sharing a Single Chunk

In the sections from 2 to 5 the dynamics of the file sharing system is analyzed by concentrating on a single chunk of the file. We study how the number of downloaders and seeds evolves over time from the emergence of the chunk to the disappearance of it. The disappearance of a single chunk means the death of the whole file sharing process since the file is not entire anymore. The work is motivated by the models of [8], [4], [11], but has some differences. In papers [4], [11] the problem of sharing of several chunks concurrently is solved by assuming that peers can forward the chunks with a constant rate. However, we find the assumption unrealistic and the model probably hides some details of the population dynamics. For this reason we consider the sharing of a single chunk at a time. In addition, papers [8] and [4] assume that at least one seed stays in the system keeping the chunks available. However, due to lack of centralized management of BitTorrent, we believe that this assumption is not always true.

In many studies ([8], [4]) the inter-arrival time of new peers is assumed to be exponentially distributed with a constant arrival rate. However, some measurement studies, such as [6] [11], show that the peer arrival rate decreases over time. So in this paper we study the population dynamics of the P2P system with time-dependent arrival rate $\lambda(t)$ for a request of the chunk. Paper [11] proposes a traffic model for P2P networks, where the arrival rate decreases exponentially from the release of the file:

$$\lambda(t) = \lambda_0 e^{\frac{-t}{\tau}}, \tag{1}$$

where parameter $\tau$ describes the attenuation of the demand over time. The average total number of requests arriving in the system, denoted by $N$, is then:

$$N = \int_0^\infty \lambda_0 e^{-t/\tau} dt = \lambda_0 \tau. \tag{2}$$

According to [11], the exponentially decreasing arrival rate fits well to the corresponding measured peer request rate. For this reason we assume (1) to be a reasonable model to capture the inherent flash crowd effect.

The downloader can download the chunk with rate $\mu_d$. On the other hand, the maximum upload rate of a peer for the chunk is assumed to be $\mu_s$. After the download, the status of the downloading peer changes from a *downloader* to a *seed*. Note that in this context, a peer is referred to as the seed if it has the chunk in question, but not necessarily all chunks of the file. Seeds stay in the system for downloading other chunks of the file as well as unselfishly serving other peers. All together, the time of staying in the system after downloading the given chunk is assumed to be exponentially distributed with mean $1/\gamma$.

Let $x(t)$ be the number of downloaders and $y(t)$ be the number of seeds at time $t$. In the next sections we study the evolution of the pair $(x, y)$ both by a deterministic fluid model and by a Markov model.

## 3 Deterministic Fluid Model

As described in the previous section, we consider a system where a peer starts to spread a single chunk to other peers that are willing to download it. In the system, if the total download capacity of the downloaders, $\mu_d x(t)$, is smaller than the total upload capacity of the seeds, $\mu_s y(t)$, the downloaders can not use all service capacity provided by the peers. On the other hand, when $\mu_d x(t) > \mu_s y(t)$ the upload capacity of seeds limits the download process. Thus the total service rate of the system is $\min\{\mu_d x(t), \mu_s y(t)\}$. In Figure 1 we have illustrated the model by a flow diagram. The evolution of the number of downloaders and seeds, pair $(x, y)$, can be described by a deterministic fluid model. The differential equations of the model are:

$$\begin{aligned} \frac{dx(t)}{dt} &= \lambda_0 e^{\frac{-t}{\tau}} - \min\{\mu_d x(t), \mu_s y(t)\}, \\ \frac{dy(t)}{dt} &= \min\{\mu_d x(t), \mu_s y(t)\} - \gamma y(t), \end{aligned} \tag{3}$$

where $y(0) = 1$ and $x(0) = 0$ meaning that at the beginning, $t = 0$, there is one seed and none downloaders. Let $\bar{x}$ and $\bar{y}$ be possible equilibrium values of $x(t)$ and $y(t)$. The differential equations (3) have different solutions depending on the parameters $\mu_s$, $\mu_d$ and $\gamma$. Assuming that the mean download and upload times

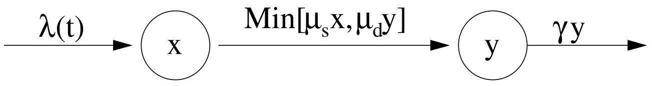


**Fig. 1.** A model for sharing a chunk

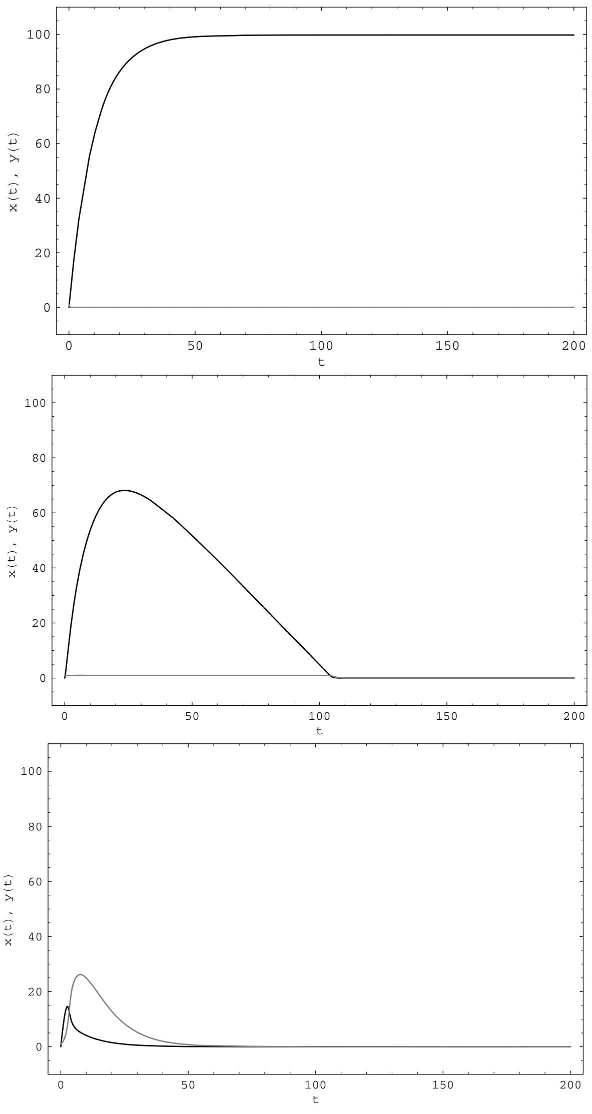


**Fig. 2.** The number of downloaders and seeds as a function of time, when $\lambda_0 = 10$, $\tau = 10$, $\mu = 1$ and $\gamma = 5$ (top), $\gamma = 1$ (middle) and $\gamma = 1/5$ (bottom). Black lines: downloaders, gray lines: seeds.

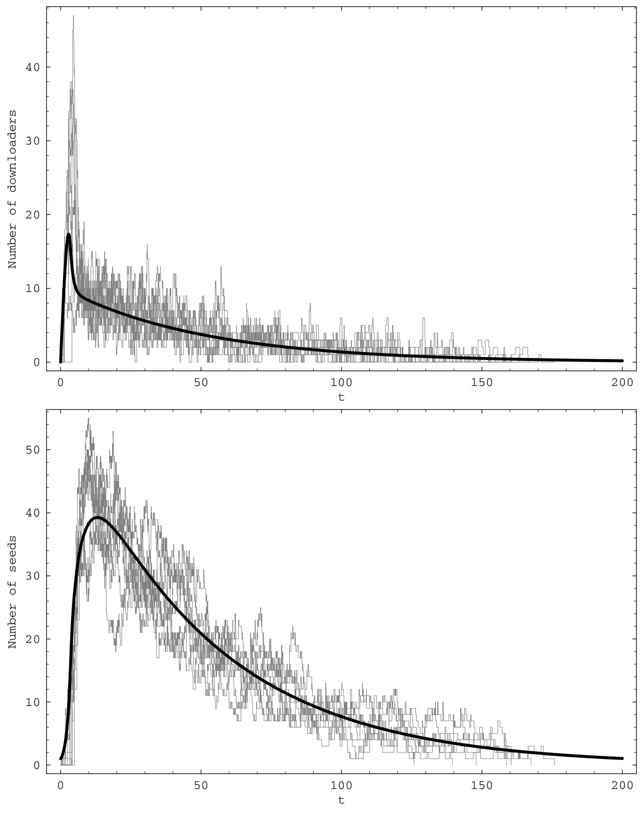


**Fig. 3.** The number of downloaders and seeds as a function of time, when $\lambda_0 = 10$, $\tau = 50$, $\mu = 1$ and $\gamma = 1/5$. Solid line: fluid model, Gray lines: simulation.

are same, $\mu_s = \mu_d := \mu$, there are three characteristic solutions for the system, which are depicted by a numerical example of Figure 2:

1. $\mu < \gamma$. The existing seeds leave the system faster than new seeds arise. For that reason the number of seeds goes to zero and the number of downloaders increases until all $N$ requests have arrived (see the top side of Figure 2). The steady-state solution is thus $\bar{x} \approx N$ and $\bar{y} = 0$. Note that these $N$ downloaders do not receive the chunk at all.
2. $\mu = \gamma$. After a very short initial period, the number of seeds stays constant, while the number of downloaders increases beyond that (see the middle of Figure 2). Finally, due to the attenuation of the arrival rate, the number of downloaders returns back to the level of the number of seeds, after which they both go to zero. The steady-state solution is thus $\bar{x} = 0$ and $\bar{y} = 0$.
3. $\mu > \gamma$. Both number of the downloaders and seeds increase until all downloaders are served (see the bottom side of Figure 2). Also in this case $\bar{x} = 0$ and $\bar{y} = 0$.

Next we compare the deterministic fluid model to simulations of the system. The number of downloaders and seeds as a function of time is shown in the top and bottom side of Figure 3, respectively. We have fixed $\lambda_0$ to a moderately small value in order to better demonstrate the dynamics of the system. The black lines correspond to the solutions of equations (3) and gray lines to 10 different simulations of the model described by Figure 1. Simulations have been done by a simple event-based simulator. In the figures we can see that the number of downloaders increases suddenly and the number of seeds increases also a little bit later. As $t \approx 200$, both $x(t)$ and $y(t)$ are decreased practically to zero. Also all simulation processes have ended due to all seeds had left the system and the chunk is not available anymore.

## 4 Time-Dependent Markov Chain Model for Chunk Sharing

The deterministic fluid model of the previous subsection describes the average behavior of the sharing of a chunk. However, from the simulation results we saw two effects in the population dynamics that were not captured by the fluid model. First, when the chunk became available the earliest seeds could not serve all the downloaders. This capacity shortage was seen as a peak of downloaders during the first 20 time units. Second, if the original seed can leave the system, sooner or later the number of seeds goes to zero. Thus the end of the whole file sharing process is irrevocable. In the simulations all processes were died before $t = 200$. The limited life span of the file sharing process has an influence on the performance of the total P2P system and has to be analyzed by some other models than the deterministic fluid model. To this end, in the next subsections we study the evolution of the process $(x, y)$ in more detail by a non-homogeneous Markov chain model with absorbtion.

In paper [12] we have derived an analytical model for the life time of the file sharing assuming that the arrival rate of new requests is constant. The result of the paper was that the mean life time of the system increases exponentially as the expected number of the seeds in the system increases. In addition, when the mean downloading time of the chunk was very short, the system can be modelled as an M/M/$\infty$-queue with arrival rate $\lambda$ and departure rate $\gamma$. The mean life time equals then to the average length of the busy period $E[B]$ of M/M/$\infty$-queue:

$$E[B] = \frac{1}{\lambda}(e^{\lambda/\gamma} - 1). \tag{4}$$

Contrary to [12], in this paper we assume that peers arrive according to Poisson process with time-dependent rate $\lambda(t)$. The mean departure time of a seed, $1/\gamma$, the mean download time $1/\mu_d$ and the mean upload time $1/\mu_s$ remain constant. Let $\pi_{(x,y)}(t)$ denote the probability of state $(x, y)$ at time $t$. Next we form the time-dependent Kolmogorov's forward equations for transition rates between the states:

$$\begin{aligned}
\tfrac{d}{dt}\pi_{(x,y)}(t) &= \lambda(t)\pi_{(x-1,y)}(t) + \min\{\mu_d x + 1, \mu_s y - 1\}\pi_{(x+1,y-1)}(t) \\
&\quad +(y+1)\gamma\pi_{(x,y+1)}(t) - (\lambda(t) + \min\{\mu_d x, \mu_s y\} + \gamma y)\pi_{(x,y)}(t), \\
&\quad \text{for all } x \geq 1, y \geq 1, \\
\tfrac{d}{dt}\pi_{(x,0)}(t) &= \lambda(t)\pi_{(x-1,0)}(t) + \gamma\pi_{(x,1)}(t) - \lambda(t)\pi_{(x,0)}, \text{ for all } x \geq 1, y = 0, \quad (5) \\
\tfrac{d}{dt}\pi_{(0,y)}(t) &= \pi_{(\mu_d 1, \mu_s y-1)}(t) + (y+1)\gamma\pi_{(0,y+1)}(t) \\
&\quad -(\lambda(t) + \gamma y)\pi_{(0,y)}(t), \text{ for all } x = 0, y \geq 1, \\
\tfrac{d}{dt}\pi_{(0,0)}(t) &= \gamma\pi_{(0,1)}(t) - \lambda(t)\pi_{(0,0)}(t), \text{ when } x = 0, y = 0,
\end{aligned}$$

with initial condition $\pi_{(x,y)}(0) = 1$, when $x = 0$ and $y = 1$ and $\pi_{(x,y)}(0) = 0$ otherwise. The explicit solution of the state probabilities is very complicated but the problem can be solved numerically using a differential equation solver.

In this system all states $(x, y)$ with $y = 0$ are absorbing. Let $z_{(x,y)}$ denote the mean time spent in state $(x, y)$ from the beginning $(t = 0)$ to the absorbtion of the system:

$$z_{(x,y)} = \int_0^\infty \pi_{(x,y)}(t)dt. \tag{6}$$

The mean time to absorbtion, i.e. the life time of the system, is thus the sum of the time spent in the non-absorbing states:

$$T_{life} = \sum_{(x,y):y>0} z_{(x,y)}. \tag{7}$$

By the proposed model above we first study how the life time of the file sharing process depends on the attenuation of the demand. As we increase parameter $\tau$, the arrival rate of the requests decreases slower and we have more downloaders in the system. That indicates also more seeds, and a longer life time of the system. This can be seen in Figure 4, which depicts the almost linear growth of the mean life time as a function of $\tau$ for different download times. For simplicity we assume that the mean upload and download times are the same, denoted by $1/\mu$. Also the shorter mean download time increases the life time of the system. The results are obtained from the numerical solution of the mean absorbtion time (7) when the state space of the Markov process is truncated to $20 \times 20$ states.

By the model we can also study how the patience of the seeds to stay in the system as servers after own download affects on the availability of the file. In the top of Figure 5 the mean life time of the chunk sharing process is shown as a function of $1/\gamma$ first for the relatively small values of $1/\gamma$. In the figure $\lambda_0 = 10$ and $\mu_d = \mu_s = 1$ and $\tau$ varies from 10 to $\infty$. We can see that the mean life time of the system increases exponentially as a function of $1/\gamma$. For example, when $\tau = 1000$, the mean life time of the system is close to the system with constant arrival rate (case $\tau = \infty$). For smaller $\tau$, such as $\tau = 10$, the increase is more moderate. The bottom of Figure 5 presents the results for same parameter

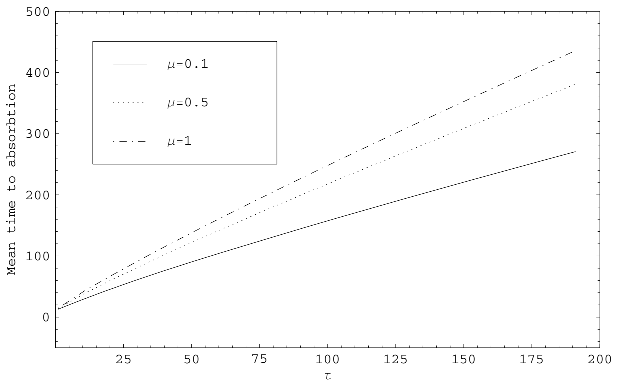


**Fig. 4.** The mean life time of the process a function of $\tau$. Parameters $\lambda_0 = 10$, $1/\gamma = 5$, $\mu_s = \mu_d = \mu$ varies from 0.1 to 1.

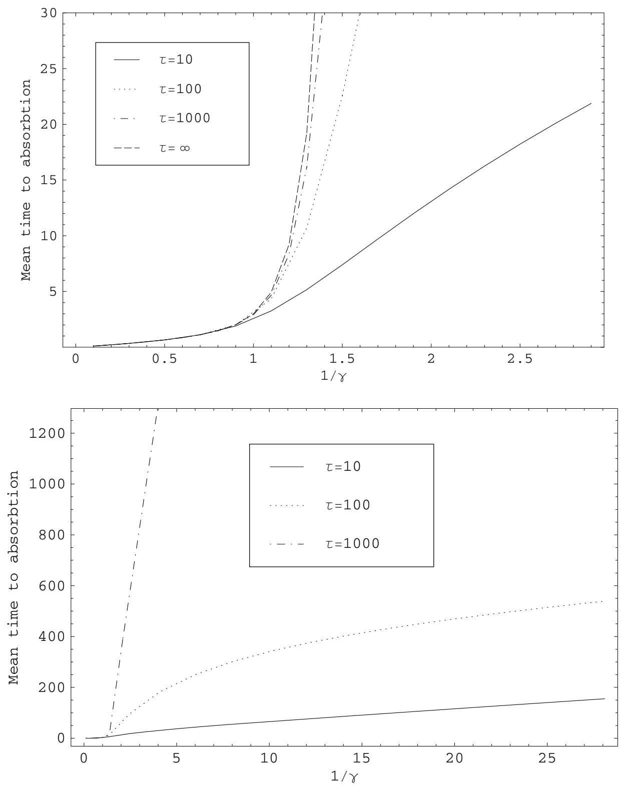


**Fig. 5.** The mean life time of the chunk sharing process when $1/\gamma$ varies from 0 to 3 (top) and from 0 to 30 (bottom). Attenuation parameter $\tau$ varies from $\tau = 10$ to $\tau \to \infty$. Parameters $\lambda_0 = 10$ and $\mu_s = \mu_d = 1$.

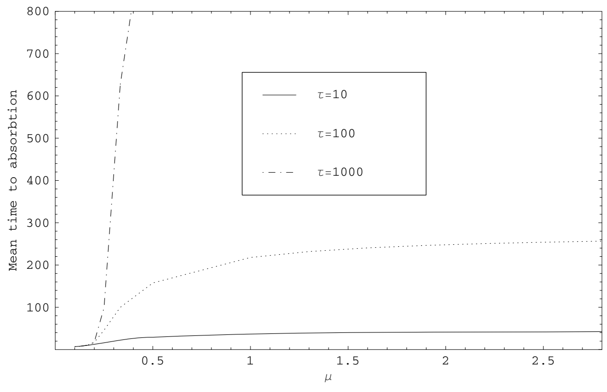


**Fig. 6.** The mean life time of the process as a function of $\mu$. Parameters $\lambda_0 = 10$, $1/\gamma = 5$ and $\tau$ varies from 10 to 1000.

combination of $\lambda_0$, $\mu_d$, $\mu_s$ and $\tau$, but the mean departure time $1/\gamma$ is longer. From the results we can see that the ultimate increase of the mean life time as a function of $1/\gamma$ is only linear when $\tau < \infty$.

Last we study the mean life time as a function of the download time of a chunk. Figure 6 depicts the mean life time of the chunk sharing as a function $\mu$. When $\mu \approx 0$, the mean download time is very long and none of the downloaders has got the file before the original seed has left the system. Thus the mean life time of the system is $1/\gamma$. For bigger $\mu$ the mean life remains almost constant since the system is characterized more by attenuation parameter $\tau$ and departure rate $\gamma$.

## 5 Approximation of the Life Time

In the previous section we presented an analytical result for the mean life time of sharing a single chunk in the P2P file sharing system. Next we will approximate the life time in a limiting case.

Also paper [11] approximates the mean time to extinction. The simple assumption is that the system dies when $\lambda(t) < \gamma$. The mean life time of the system $T_{life}$ can be then approximated by solving $t$ from equation $\lambda(t) = \gamma$:

$$T_{life} = \tau \log\left(\frac{\lambda_0}{\gamma}\right). \tag{8}$$

However, we will see that this is a very rough approximation for the life span of the bitTorrent-like P2P system and much better bounds for the mean time to absorption can be given.

We consider a case where the mean time that seeds spend in the system is long as compared to the length of the burst of arrivals (meaning the time period

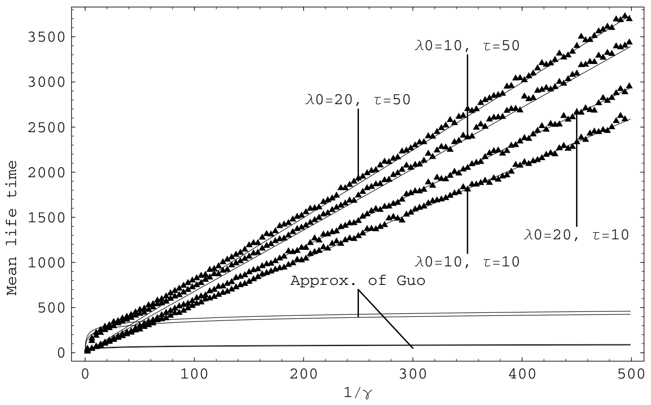


**Fig. 7.** The mean life time of the process as a function of $1/\gamma$. First four lines (from bottom): Approximation of paper [11]. Next four lines: our approximation (line) together with corresponding simulations (dots), when a) $\lambda_0 = 10$ and $\tau = 10$, b) $\lambda_0 = 20$ and $\tau = 10$, c) $\lambda_0 = 10$ and $\tau = 50$, d) and $\lambda_0 = 20$ and $\tau = 50$. Parameter $\mu_d = \mu_c = 1$.

from the beginning to the moment when $\lambda(t) \approx 0$). If we assume that any seed has not left the system before the end of arrival burst, the life time of the chunk sharing system can be approximated by the two consecutive time periods, $T_1$ and $T_2$, where $T_1$ is the length of the arrival burst and $T_2$ is the length of the period from the first to last departure of the seeds:

$$T_{life} \approx T_1 + T_2. \tag{9}$$

First, the length of period $T_1$ depends on the attenuation of the demand, which is described by parameter $\tau$. If we wait fraction $p$ of total $N$ arrivals, the length of the burst period is:

$$T_1 = \tau \log(\frac{1}{1-p}). \tag{10}$$

After the arrival burst all downloaders gradually change their status from the downloader to the seed. We have $N$ seeds in the system and the mean departure time of all $N$ seeds is

$$T_2 = \frac{1}{N\gamma} + \frac{1}{(N-1)\gamma} + \frac{1}{(N-2)\gamma} ... + \frac{1}{\gamma} = \sum_{i=0}^{\lfloor N \rfloor} \frac{1}{i\gamma}. \tag{11}$$

Next we assume that the mean departure time is long as compared to the attenuation of the demand, $1/\gamma \gg \tau$. In this case also $T_2 \gg T_1$ and we can approximate $T_{life} \approx T_2$. In Figure 7 we compare the proposed approximation to the simulation of the corresponding system with different combinations of

$\lambda_0$ and $\tau$. If $1/\gamma$ is long, there are lot of seeds in the system and we use simulation instead of analytical model due to limited state space of the Markov model (simulation results are averaged over 100 simulation replications). Also the approximation of paper [11] is presented in Figure 7. For small values of $1/\gamma$ neither of the approximations are good. However, as $1/\gamma > \tau$ we can see that our approximation is very accurate for many parameter combinations as compared to the life time proposed by paper [11].

## 6 Sharing of Multiple Chunks

In the previous sections we considered sharing of a single chunk as an independent process of other chunks and estimated the life time of the file sharing system by the disappearance of a single chunk. However, in reality the dynamics of a single chunk depends on the other chunks as well. So in this section we consider population dynamics of a more complex system with multiple chunks.

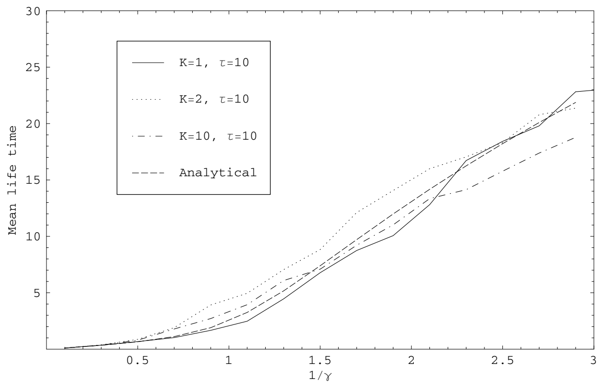
In the system we have three types of peers, *downloaders*, *leechers*, and *seeds*. Downloaders do not have any chunk yet and try to only download a first one. Leechers have already downloaded a part of the chunks and can upload those chunks to other peers in the system. While uploading the chunks, they try to find the rest of the chunks. Finally, peers that have all chunks are called seeds. These seeds stay for a while in the system uploading the chunks to the downloaders.

Let $L$ be the size of the file under consideration in bytes. The file is divided into blocks, named chunks, which are assumed to be downloaded one after another separately. If we fix the size of chunk to $l$ bytes, the total number of chunks is naturally $L/l$, denoted by $K$. New peers requesting the given file are assumed to arrive at the system with time-dependent rate $\lambda(t)$. Also in this section we assume that request rate decreases exponentially according to formula (1).

When new downloader $i$ has arrived, it seeks random peer $j$ among all available peers including leechers and seeds in the system and compare the own set of chunks and peer $j$'s set of chunks. If downloader $i$ finds a chunk that is not in its own collection, it starts to download it. We fix the mean download time of a chunk to be proportional to the chunk size, that is, $\frac{1}{K\mu_d}$. Also the maximum upload rate of a peer is proportional to the chunk size, $K\mu_c$. If there are many downloaders per one leecher or seed, the capacity of the peer is divided among downloaders. We assume that time required to find the peer for exchanging chunks is negligible as compared to the download time.

After the peer has downloaded the first chunk, it seeks a next random peer and a next random chunk. Also if selected peer $j$ has not any new chunk that the downloader $i$ does not already have, downloader seeks a new peer. When a peer has some, it can as a leecher upload the chunk to other peers, if required. When the leecher has collected all $K$ chunks, the status of the peer changes from a leecher to a seed. We assume that the seeds stay in the system for a random period. Let $\gamma$ denote the departure rate of a seed.

However, we do not assume that there is always a seed keeping the file entire in the system. If some of the chunks is missing, i.e. there are no seeds in the system



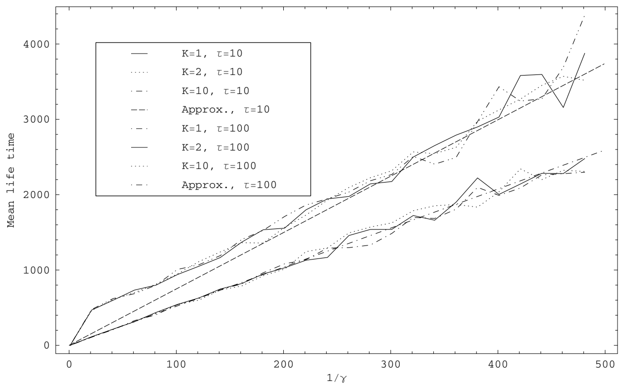


**Fig. 8.** The mean life time of the process as a function of $1/\gamma$, when the file is divided into multiple chunks. Number of chunks varies from $K = 1$ to $K = 10$ and $\tau$ varies from $\tau = 10$ to $\tau = 100$. Parameters $\lambda_0 = 10$, $\mu_d = \mu_c = 1$.

and the chunk is not included in any chunk collections held by the leechers, the file is not complete anymore. In this case the sharing process of the given file dies.

First in the top of Figure 8 we have simulated the file sharing process for small values of $1/\gamma$, when the number of chunks varies from $K = 1$ to $K = 10$ and $\tau = 10$. The number of the simulation replications for a given value of $1/\gamma$ is 500. From the figure we can see that our analytical model for sharing of a single chunk corresponds well to the system with multiple chunks.

In the bottom of Figure 8 we have simulated the same file sharing process for big values of $1/\gamma$. Lower three lines correspond to $\tau = 10$, and upper three lines to $\tau = 100$. Also the approximations for one chunk of the previous section are presented. The result is that also in these cases the mean life time of the system increases linearly as a function of $1/\gamma$ and fits well to approximation of the chunk model presented in Section 5. For these parameter combinations, the selected coarseness of the division of the file into chunks does not influence substantially the mean life time of the file sharing process.

As a result of this section we can say that the mean life time calculated from the one chunk model correspond well to the life time of the more realistic system with multiple chunks. This information helps in performance evaluation and optimization of the file sharing systems with varying system parameters.

## 7 Conclusion

In this paper we have studied the population dynamics and the file availability of the BitTorrent-like P2P system. Our approach was to calculate the mean life time of a single chunk independently of the other chunks. We considered both a deterministic fluid model and a non-homogenous Markov chain model. By the latter model we were able to study how the different parameters, such as the attenuation of the demand, the mean download time and the mean departure time affected the mean life time of the sharing process. We provided also a simple approximation for the life time, which gave better results than an earlier proposal. The applicability of the one chunk model was verified by the simulations of sharing of the file in multiple chunks. Our approach in this paper was rather in modelling the existing systems than in optimization of the P2P file sharing algorithms.

In future we will study more the analytical models and especially approximations of the life time also in other scenarios than we have done so far. For example, the behavior of the system, when seeds can return back to the system, is interesting. In addition, we will deepen the study of the scenario with multiple chunks by both analytical model and more complex simulations.

## References

1. T. Karagiannis, A. Broido, N. Brownlee, kc claffy, M. Faloutsos, Is P2P dying or just hiding?, in Globecom, 2004.
2. `http://www.cachelogic.com/p2p/p2ptraffic.php`
3. B. Cohen, Incentives Build Robustness in BitTorrent, 2003, in Proc. of First Workshop on Economics of Peer-to-Peer Systems, June 2003, `http://www.bittorrent.com/bittorrentecon.pdf`.
4. D. Qiu, R. Srikant, Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks, in Proc. of SIGCOMM 2004.
5. M. Izal, G. Uvroy-Keller, E.W. Biersack, P.A. Felber, A.Al Hamra, and L. Garcés-Erice, Dissecting BitTorrent: Five Months in a Torrent's Lifetime, in Proc. of PAM, 2004.
6. J.A. Pouwelsem P. Garbacki, D.H.J. Epema, H.J. Sips, The BitTorrent P2P File-sharing system: Measurements and analysis, in Proc. of IPTPS, 2005.
7. L. Massoulié and M. Vojnović, Coupon replication Systems, in Proc. of SIGMETRICS, 2005.
8. X. Yang, G. de Veciana, Service Capacity of Peer to Peer Networks, in Proc. of INFOCOM 2004.
9. K.K. Ramachandran, B. Sikdar, An Analytic Framework for Modeling Peer to Peer Networks, in Proc. of INFOCOM 2005.

10. Y. Tian, D. Wu, K. Wing Ng, Modeling, Analysis and Improvement for BitTorrent-Like File Sharing Networks, in Proc. of INFOCOM 2006.
11. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, Measurement, Analysis, and Modeling of bitTorrent-like Systems, in Proc of Internet Measurement Conference of USENIX Association, 2005.
12. R. Susitaival, S. Aalto, J. Virtamo, Analyzing the dynamics and resource usage of P2P file sharing by a spatio-temporal model, in Proc. of P2P-HPCS'06 in conjuction with ICCS'06, pp. 420–427, 2006.

# Combining Virtual and Physical Structures for Self-organized Routing

Thomas Fuhrmann[*]

System Architecture Group, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
Tel.: +49 721 6086730; Fax: +49 721 6087664
fuhrmann@ira.uka.de

**Abstract.** Our recently proposed scalable source routing (SSR) protocol combines source routing in the physical network with Chord-like routing in the virtual ring that is formed by the address space. Thereby, SSR provides self-organized routing in large unstructured networks of resource-limited devices. Its ability to quickly adapt to changes in the network topology makes it suitable not only for sensor-actuator networks but also for mobile ad-hoc networks. Moreover, SSR directly provides the key-based routing semantics, thereby making it an efficient basis for the scalable implementation of self-organizing, fully decentralized applications.

In this paper we review SSR's self-organizing features and demonstrate how the combination of virtual and physical structures leads to emergence of stability and efficiency. In particular, we focus on SSR's resistance against node churn. Following the principle of combining virtual and physical structures, we propose an extension that stabilizes SSR in face of heavy node churn. Simulations demonstrate the effectiveness of this extension.

## 1 Introduction

Many researchers consider routing to be a prototypical example for self-organizing distributed systems. In practice, however, large-scale routing is often not as self-organizing as one might think. For example, the Internet's scalability relies on a suitable assignment of addresses that allows address aggregation in the routers. Self-organizing network systems such as mobile ad-hoc or sensor network routing protocols, on the other hand, mostly limit the network size to a few hundred nodes or limit the routing purpose to, e. g. , data aggregation.

We aim at a related but different scenario: large unstructured networks where the nodes communicate using the key-based routing semantics of structured routing overlays. Our guiding example is a community of digital homes where inexperienced users deploy an organically growing number and variety of networked devices. These devices comprise tiny processing and communication units that interface with sensors or actuators or both. Depending of the respective application these devices shall be able to communicate across the entire network.

[*] This work was supported by Deutsche Forschungsgemeinschaft under grant FU448-1.

The following examples shall motivate why we believe that such a scenario will become more and more important in the near future.

Example 1: With SSR, the association of a wall-mounted switch (=sensor) with a light bulb or marquee motor (=actuator) is no longer fixed by physical cabling, but can be (re-)programmed according to the current users' preferences. Since the system is self-organizing, users can arbitrarily add and remove devices without breaking the system. This is more robust than today's centralized building automation systems.

Example 2: Blinds, marquees, etc. must be retracted when heavy rain or a storm approach. Today, each house has its own sensors. With SSR, all the sensor readings from the neighborhood can be combined to yield more reliability for the entire community. This is achieved by exploiting the aggregation tree implicitly formed by proximity aware key-based routing (see below).

Example 3: Sensors in the greenhouse at the other end of the garden need not directly report to the associated display in the living room, kitchen, etc. Any infrastructure (both wired and wireless) can relay the data. If required, the data can be redirected to other locations on demand, e.g. to the person who takes care of the greenhouse when the owners are on vacation.

In the sketched scenarios, typically, wireless links are mixed with wired links. Individual sensors might use (different) radio communication technologies, but there will also be e. g. power line cabling. Using these wires to bridge larger distances reduces both, the energy required by the sensors and the electromagnetic interference problems. Thereby, communication relations can efficiently be stretched across the whole network. Unlike today's building automation networks, the resulting network will lack any simple structure: it will neither be a tree that allows an address structure that can be aggregated, nor will it be a simple 2-dimensional layout, as it is, for example, required to use greedy perimeter geographical routing.

Furthermore, these examples also demonstrate that self-organization is essential for future networks: Users will add and remove devices without taking care of the network structure. These devices may have only limited memory and computation resources. New devices need to be quickly integrated; and devices leaving suddenly and ungracefully should not harm the entire network's functionality. Small, isolated networks consisting of only a few devices may grow together to form a large network with thousands of devices covering a whole neighborhood of digital homes. Scalable source routing (SSR) has been specifically designed for such scenarios.

The SSR protocol has been described elsewhere in detail [6]; and we kindly ask you, the reader, to revert to that publication for any questions regarding the details of the SSR protocol. Here we primarily discuss SSR's self-organizing properties. By that we mean SSR's ability to provide structured routing in large, unstructured networks without central components and without the need for configuration or human interaction. Unlike many state-of-the-art routing protocols for large networks, SSR operates in arbitrary network topologies, i. e. SSR is self-organizing not only with respect to its operation but also with respect to the physical system setup.

Especially, in this paper, we describe how a simple set of rules creates a globally consistent structure from limited local information. Furthermore, we describe how these simple rules achieve robustness in face of node churn, i. e. when nodes ungracefully drop out of the network. We hope to thereby promote the general understanding for the mechanisms of self-organizing systems, and to foster the application of similar principles in other problems in networking and distributed systems.

This paper is structured as follows. Section 2 briefly describes SSR's core rules. Section 3 discusses SSR's mode of operation in the light of self-organization. Based on this insight, section 4 describes a novel extension to SSR that improves SSR's stability in face of node churn. Finally, section 5 concludes with an outlook to future work.

## 2 Scalable Source Routing

Scalable source routing (SSR) is a self-organizing routing protocol that provides key-based routing in large, unstructured networks where nodes have limited resources. Its design was guided by the following assumptions that reflect the constraints and conditions of the scenario described above:

1. Nodes can afford only little memory to hold routing state. The memory is assigned statically, i. e. there is no way to acquire additional memory. (Our implementation, for example, requires only 4kB state per node.)
2. The network has no simple topology, i. e. it is neither hierarchical nor necessarily planar. (This is caused by mixing wireline and wireless links.)
3. Nodes have location independent addresses, e. g. assigned by the manufacturer.
4. Nodes may communicate with many different destinations when using the key-based routing semantic.

SSR's key engineering trick is the combination of a virtual structure, the address space, with the actual physical structure (=topology) of the network. Both structures are equipped with a metric. The virtual metric is defined as the absolute value of the numerical address difference, taking the address wrap into account. The physical metric is defined as the length of a source route (i.e. hop count) that connects the respective nodes.

As a result, SSR combines the beneficial properties of other routing approaches:

Like typical on-demand protocols such as AODV [13] or DSR [10], SSR does not need to maintain routing state for all the nodes in the network. This keeps the required routing state small and does not burden an otherwise (nearly) idle network with unproportionally high control traffic.

Like typical link state or distance vector protocols such as OSPF or RIP, once consistency is achieved, SSR can route any message without the need to acquire or build up routing state for a previously unknown destination. This keeps the routing delay low and avoids buffering the messages.

Moreover, SSR does *not need to flood* the network with route request messages (contrary to e. g. AODV and DSR); and it does *not need to assign special node identifiers* according to the nodes' position in the network (contrary to e. g. LANMAR [12], Safari [2], and Beacon Vector Routing [4]). Thereby SSR avoids the use of additional look-up protocols which, again, reduces the control traffic associated with such look-up mechanisms. Moreover, SSR does not make any assumption about the network topology. Thus SSR can efficiently combine wireless and wireline links. This is unlike e. g. *Greedy Perimeter Stateless Routing* [11] where the network graph is assumed to be planar.

SSR shares some ideas with previous work by various authors: DPSR [9] combines the Pastry overlay with DSR. Ekta [14] enhances DPSR with indirect routing. But unlike SSR both protocols need to flood the network to discover source routes. MADPastry [17] combines Pastry with AODV to achieve indirect routing. Unlike SSR, MADPastry needs to maintain a key-to-node mapping which causes significant control traffic when nodes are mobile. Several other authors have proposed similar uses of overlay structures for the maintenance of location directories [3,5,16].

One of SSR's key advantages is that the protocol is so simple that it can be compactly implemented on resource-limited embedded controllers. Its core rule is based on the Chord [15] idea: Nodes have location independent addresses that are viewed as virtual ring.

> A message is forwarded so that its virtual distance to the destination decreases monotonically.

Clearly, this is not always possible with a node's direct physical neighbors alone. Hence, a node maintains a cache with source routes which correspond to the 'fingers' in the Chord overlay. (See below which source routes are entered into the cache.) >From that cache the node picks a so-called next *intermediate node* to which the message is then forwarded using the respective source route.

> When choosing the next intermediate node, physically close nodes are preferred over virtually far nodes.

As a result, SSR builds up a source route in the message header that spans from the message's source to its destination including the intermediate nodes. Note that, when appending a source route, loops are cut out of the source route so that a minimal spanning tree remains.

> Intermediate nodes enter the source routes that are conveyed in the messages into their cache. This cache is operated in a least-recently-used manner.

Those familiar with the Chord overlay routing protocol and its proximity route selection extension [8] will recognize that the just described procedure allows efficient *key-based routing* (KBR). KBR is a generalization of unicast routing, i. e. SSR is capable of both, KBR and unicast routing. With KBR messages are delivered to that node whose address is numerically closest to the message's destination. KBR can serve as foundation for many distributed applications, like for example distributed hash tables (DHT).

Nevertheless, as known from Chord, correct unicast routing and consistent key-based routing is only guaranteed if each node stores correct fingers (here: source routes) to its virtual neighbors, i. e. its successor and predecessor in the address space ring.

Before we describe why SSR is in fact able to self-organizingly obtain and maintain these source routes, we briefly discuss figure 1 which illustrates SSR's mode of operation. In practice, in such a small network all nodes would quickly acquire the full topological knowledge of the entire network. For our example here, we assume that nodes have only very limited information, i. e. each node knows only its direct physical and virtual neighbors.

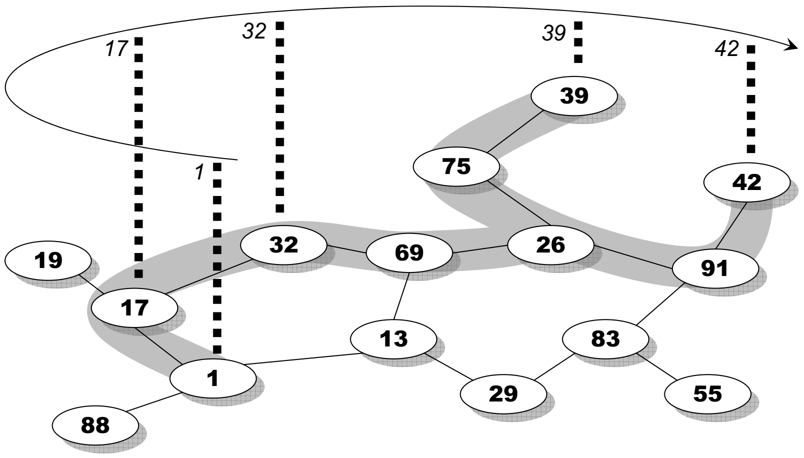


**Fig. 1.** Illustration of Scalable Source Routing

Assume further that node 1 wants to send a message to node 42. The message is first sent from node 1 to node 17, because among the physical neighbors of node 1 node 17 is virtually closest to the destination. (Note that $42 - 17 = 25$ is smaller than $42 - 13 = 29$ and $88 - 42 = 46$.) For the same reason, node 17 forwards the message to node 32.

So far, the routing process has only involved direct physical neighbors. At node 32, none of the physical neighbors is virtually closer to the destination than node 32, the current intermediate node. Thus, node 32 must append a non-trivial source route. According to our assumption, node 32 caches a source route to its virtual neighbor, node 39, so that the message can be accordingly forwarded to node 39. Node 39, finally, can append a source route to node 42.

As can be seen from figure 1 the result of this routing process is a spanning tree that connects the intermediate nodes 1, 17, 32, 39, and 42. (Note that for simplicity of terminology, we include the source and destination node in the set of intermediate nodes.) This tree is conveyed in the message header, so that the intermediate nodes, especially the destination node, can enter the source routes to the previous intermediate nodes into their caches. Thereby, the nodes acquire more knowledge about the network.

In our example, node 42 has acquired a source route to node 32. When node 42 sends a reply via node 32 to node 1, node 32 will acquire a source route to node 42. Thus subsequent messages from node 1 to node 42 will not have be routed via node 39 any more. The nodes in the network are 'learning' the relevant source routes.

Note that in order to reduce the overhead in message headers, the source route tree can be pruned, so that it contains only the most recent intermediate nodes. A detailed analysis shows that these are the most relevant nodes of this self-organized cache improvement process.

## 3 SSR's Self-organizing Properties

The discussion of the example from figure 1 has already shown that once the nodes have source routes to their direct virtual neighbors, SSR's caching rule has the effect that the nodes also accumulate source routes to (some of) their indirect virtual neighbors. In presence of payload traffic, these source routes are frequently used and therefore remain in the caches. Nevertheless all state is soft state and can thus adapt to changes. (See below for a discussion of cache maintenance in presence of node churn.) Moreover, unlike e. g. Chord, SSR does not employ an explicit stabilization method, but self-organizingly builds the respectively required knowledge. As a consequence, SSR does not necessarily produce control messages in an idle network. This is an important feature in our target scenario.

As mentioned above, SSR only works consistently if all nodes have source routes to their respective direct virtual neighbors. In the remainder of this section, we discuss why this is in fact likely to be the case in practice. To this end, we have to review three emergent properties of SSR.

1. First, we note that due to SSR's preference for physically close nodes, each of a node's direct physical neighbors will be chosen as a first intermediate node for a particular address space interval, namely the interval covering all positions in the address space for which the respective node is virtually closest. Fig. 2 illustrates the choice for node 1 in our example of fig. 1: Node 1 sends traffic for the interval $[7, 14]$ to node 13, traffic for the interval $[15, 52]$ to node 17, and traffic for the interval $[53, 94]$ to node 88. Traffic for the interval $[95, 6]$ stays at node 1. (We assume nodes to have addresses in $[0, 99]$.)

Correspondingly, node 32 sends traffic for the interval $[51, 92]$ to node 69, and traffic for the interval $[93, 24]$ to node 17. Thus, nodes specialize for the addresses around their own address and attract the respective traffic from their physical vicinity.

This is demonstrated by the simulation results in figure 3. The plot shows nodes (black disks) in a unit disk random graph that all try to route messages

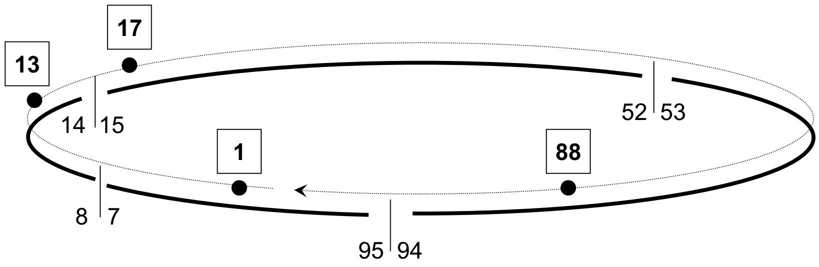


**Fig. 2.** Illustration of address range specialization

to the same destination. For the plot, this destination was randomly picked. The node density in the graph is chosen to be near the critical density to obtain a connected graph (approx. 10 nodes per radio range disk). The arrows indicate to which of their direct physical neighbors the nodes forward the messages.

Obviously, clusters emerge in the graph, where each cluster forwards the message to one node in the cluster (= cluster head), namely the node that has specialized in the respective destination address. Note that unlike many MANET protocols, SSR does not explicitly determine the clusters and the cluster head. The clusters rather are an emergent property of SSR that is caused by the basic routing rule. Consequently, the cluster formation is robust against node churn. Note further that the clustering will be different for different destination addresses.

Clearly, if SSR had only this first emergent property described so far, the system would be inconsistent in the sense that messages destined to the same address but originating in different clusters would not end up at the same node. Two further emergent properties resolve this inconsistency. By virtue of theses properties SSR can connect (some of ) the nodes within the clusters; and it can connect nodes in different clusters via the cluster heads.

2. Upon bootstrapping, a node only knows its direct physical neighbors, and thus has to choose its virtual neighbors from this limited set of nodes. With high probability this leads to inconsistent assumptions among the nodes in the clusters. In the example of figure 1 both, node 17 and node 13, will assume node 1 to be their predecessor. If node 1 is aware of this inconsistency, it can easily resolve it. Thus, SSR requires the nodes to express their assumptions:

> When routing to a next intermediate node that is assumed to be a virtual neighbor of the previous intermediate node, indicate that assumption in the message header.

In the example, upon reception of such a message from node 17, node 1 can inform node 17 of the existence of node 13 by sending a message containing a source route that connects node 1 and node 13. In general, the cluster heads will serve as catalysts for their physical neighbors to find source routes to better virtual neighbors within the cluster. However, this process will only rarely

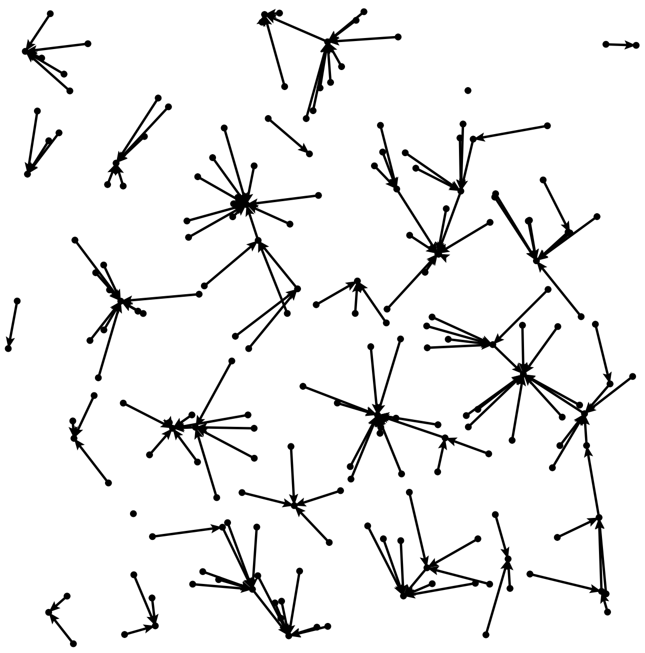


**Fig. 3.** Example for SSR's self-organized clusters

provide nodes with source routes to their direct virtual neighbors, because these are likely to lie in other clusters.

The observation of a third emergent property explains why this is only an apparent problem:

3. The clusters are slightly different for slightly different destination addresses. Thus, information about the virtual neighbors can be conveyed between the different clusters. In the example of figure 1 node 1 mediates the information that nodes 13 and 17 are virtual neighbors. Before that mediation, messages from node 29 or 69 to node 17 would have ended at node 13, whereas the same message originating at nodes 1 or 32 would have been correctly routed to node 17. In other words, nodes 29 and 69 belong to another cluster than 1 and 32 with respect to destination node 17. After that mediation, all the messages will be correctly routed to node 17, i. e. the clusters (for destination node 17) have become connected by the source route connecting nodes 13 and 17. Furthermore, node 17 can now mediate a source routing connecting the direct virutal neighbors 29 and 32.

Simulations show that even large networks converge quite quickly [6]. Even though a detailed analysis [1] shows that this bootstrapping mechanism cannot always guarantee global consistency, extensive simulations including node mobility and churn [7] indicate that an such inconsistency is unlikely to appear in practice. In fact, any inconsistency in the network is much more likely to be caused by nodes moving or leaving the network and thereby breaking the source routes that connect the virtual neighbors. We will discuss this problem in the following section.

## 4 Stability in Face of Churn

To cope with node and link churn, overlay peer-to-peer protocols such as Chord maintain finger tables that contain not only the direct virtual neighbors, but also several indirect virtual neighbors. But with SSR the source routes to the virtual neighbors have a high probability to overlap. As explained above, in the example of figure 1, node 1 mediates the route 13-1-17 from which node 17 then mediates 29-13-1-17-32. Node 1 is thus crucial for the virtual neighborhood of nodes 13, 17, 29, and 32. In general, with SSR often a single node suffices to break the source routes to all direct and several indirect virtual neighbors of a node.

Luckily, a simple trick resolves that problem: Nodes cache not only the source routes to their virtual neighbors, but also store some of those virtual neighbors' physical neighbors. When a route is found to be broken and cannot be salvaged, it is back-propagated to the intermediate node that appended the broken path. There, the outdated source route is deleted from the cache and the message is then routed to one of the physical neighbors of the now unreachable virtual neighbor. Since with high probability that new destination induces an entirely different cluster pattern, the message is likely to follow a completely independent path. Once that new destination has been reached, the message can be trivially forwarded to the original destination because it is by definition a physical neighbor. In order to do so, the original destination has always to be kept in the message header.

We explain this again with the example of figure 1: Assume the link between 26 and 91 is broken, e. g. because node 91 has moved out of node 29's radio range. When the message in the example above cannot be forwarded to node 91, it is back-propagated to node 39, the previous intermediate node. Along the way, the nodes delete the respective link from their caches. If any of the nodes on the path back to node 39 can salvage the source route, i. e. if it knows a source route to a node in the message's source route that lies beyond the broken link, the message's source route is accordingly modified and the message is forwarded along the new source route. In order to update the caches SSR always back-propagates at least a respective control message indicating the broken link together with an alternative route. This is especially important for node 39 whose cache was the cause that the message contained an outdated link. This back-propagation mechanism ensures that the caches are quickly updated when outdated information is used for the routing process.

If the message has been back-propagated to node 39 without having been salvaged, node 39 substitutes the destination 42 with one of node 42's physical neighbors, here with node 91. According to SSR's routing rules, the message is now routed along the intermediate nodes -39-75-83-91. At node 91, the message can then be forwarded to the original destination, node 42.

As a result, SSR achieves a high delivery ratio even in face of heavy node churn. This may however be at the expense of a message being in flight for a

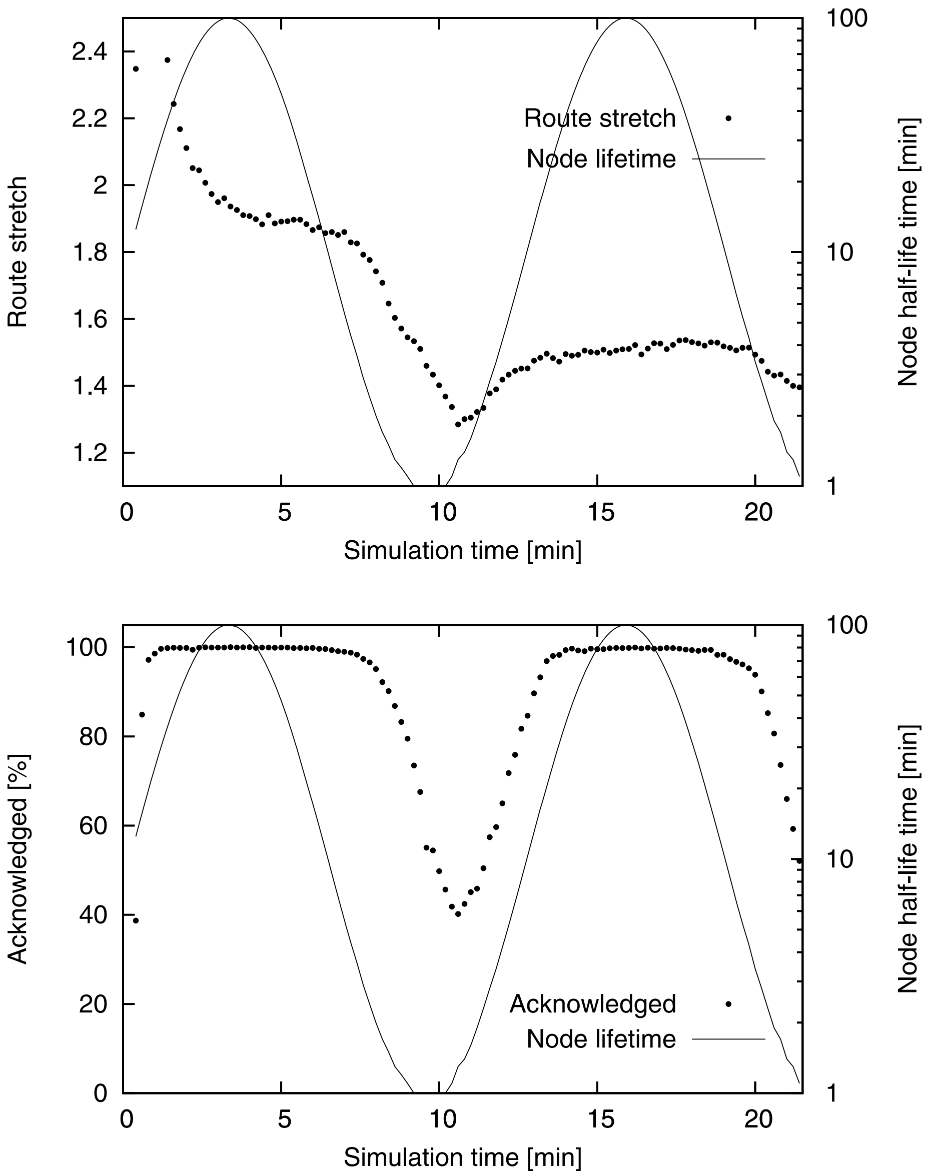


**Fig. 4.** SSR's performance in face of heavy node churn

potentially long time. Nevertheless, extensive simulations show that SSR is still more efficient than state-of-the-art protocols such as AODV that has to flood the network to discover routes. The comparison of SSR to MANET routing protocols such as AODV is extensively discussed in [7]. Here, we focus on the effect of node churn.

Figure 4 shows results for an 8000 node small-world network with 8.3 kBit/s links. We have chosen this topology here, because it is more affected by node churn than unit disk graph networks, which are typical for MANETS. Moreover, we expect the networks in our target scenario to be exhibit more small-world properties than typical MANETs. The low link rate reflects the capabilities of typical low-cost, low-resource embedded devices.

In our simulation, each node every 12 seconds sends a message to a randomly chosen node (equal distribution). This simulates frequent look-ups in e. g. a DHT built on top of SSR. Upon reception of such a message, the receiving node acknowledges the message by sending back a response to the message's original source. Throughout the simulation nodes are killed according to a Poisson process with a half-life time that oscillates between 60 and 6000 seconds (plotted line). Whenever a node is killed, a new node joins the network (at a random position), thereby keeping the entire number of nodes constant. Note that the oscillating node life time allows us to study the hysteresis in the routing system, an effect that is typical for many self-organizing dynamic systems.

Figure 4 shows the acknowledgment rate and the route stretch (i.e. the ratio to the shortest path determined using the Dijkstra algorithm and global knowledge of the network) for the first two oscillation periods of a typical simulation run. As expected from [6], during bootstrapping, the route stretch peaks at about 2.4 and slowly declines afterwards. The stretch is slightly smaller during heavy node churn periods (i. e. periods with small half-life time) because then longer source routes have a significant probability to break and thus do not contribute to the stretch calculation.

More importantly, the simulation shows that with this extension SSR is capable of maintaining almost 100% correct delivery as long as the nodes' half-life time is above about 10 min. Even if the life time falls below 1 min, SSR is able to correctly acknowledge almost 50% of the messages.

Note that for determining these values we have to take the hysteresis into account. To this end, we compare the rate at a given life-time value when the life time is reducing, with that value when it is increasing. Thereby, we eliminate potential short-term effects and get a good approximation for the long-term behavior for the respective life-time value.

## 5 Conclusion

In this paper, we have reviewed our recently proposed scalable source routing (SSR) protocol. SSR combines source routing in the physical network with Chord-like routing in the virtual ring that is formed by the address space. In particular, we have explained how SSR's combination of the virtual ring structure with the physical network topology creates a self-organizing routing protocol that is able to efficiently operate in large scale networks. We have pointed out that SSR is especially suited for organically growing networks of small embedded devices that have limited memory and computation resources.

Based on our discussion of SSR's self-organizing features we have proposed an extension to SSR, that stabilizes SSR under heavy node churn. This extension was motivated by the insight, that a single churn event can affect the source routes to several of a node's virtual neighbors, but that it is unlikely to affect the source routes to the virtual neighbors' physical neighbors. In order to confirm the effectiveness of that extension, we have briefly discussed an 8000 node simulation of SSR.

Currently, we are applying SSR's basic principle, namely the combination of virtual and physical structures to other problems in the design of self-organizing systems. These problems include various aspects of distributed systems such as distributed scheduling and replica management. We hope to be able to report on first results soon.

## References

1. Curt Cramer and Thomas Fuhrmann. Self-Stabilizing Ring Networks on Connected Graphs. Technical Report 2005-05, University of Karlsruhe (TH), Fakultaet fuer Informatik, March 2005.
2. Shu Du, Ahamed Khan, Santashil PalChaudhuri, Ansley Post, Amit Kumar Saha, Peter Druschel, David B. Johnson, and Rudolf Riedi. Self-Organizing Hierarchical Routing for Scalable Ad Hoc Networking. Technical Report TR04-433, Department of Computer Science, Rice University, Houston, TX, USA, 2004.
3. Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurty. PeerNet: Pushing Peer-to-Peer Down the Stack. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Claremont Hotel, Berkeley, CA, USA, February 2001. Springer Verlag.
4. Rodrigo Fonseca, Sylvia Ratnasamy, Jerry Zhao, Cheng Tien Ee, David Culler, Scott Shenker, and Ion Stoica. Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation*, Boston, MA, U.S., May 2005.
5. Bryan Ford. Unmanaged Internet Protocol. *ACM SIGCOMM Computer Communications Review*, 34(1):93–98, January 2004.
6. Thomas Fuhrmann. Scalable routing for networked sensors and actuators. In *Proceedings of the Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, September 2005.
7. Thomas Fuhrmann, Kendy Kutzner, Pengfei Di, and Curt Cramer. Scalable Routing for Hybrid MANETs. submitted.
8. K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the SIGCOMM 2003 conference*, pages 381–394. ACM Press, 2003.
9. Y. Charlie Hu, Saumitra M. Das, and Himabindu Pucha. Exploiting the synergy between peer-to-peer and mobile ad hoc networks. In *Proceedings of HotOS-IX: Ninth Workshop on Hot Topics in Operating Systems*, Lihue, Kauai, Hawaii, May 2003.
10. David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, 353:153–181, February 1996.
11. Brad Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, pages 243–254, Boston, MA, August 2000.

12. Guangyu Pei, Mario Gerla, and Xiaoyan Hong. Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility. In *Proceedings of IEEE/ACM MobiHOC 2000*, pages 11–18, Boston, MA, U.S., August 2000.
13. Charles E. Perkins and Elizabeth M. Royer. Ad hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, USA, February 1999.
14. Himabindu Pucha, Sumitra M. Das, and Y. Charlie Hu. Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004)*, English Lake District, UK, December 2004.
15. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the SIGCOMM 2001 conference*, pages 149–160. ACM Press, 2001.
16. Aline Carneiro Viana, Marcelo Dias de Amorim, and Serge Fdida. An Underlay Strategy for Indirect Routing. *Wireless Networks*, 10:747–758, 2004.
17. Thomas Zahn and Jochen Schiller. MADPastry: A DHT Substrate for Practicably Sized MANETs. In *5th Workshop on Applications and Services in Wireless Networks (ASWN 2005)*, Paris, France, June 2005.

# Optimizing Locality for Self-organizing Context-Based Systems[*]

Mirko Knoll and Torben Weis

Context-based Systems Group, Universität Stuttgart
{knoll, weis}@ipvs.uni-stuttgart.de

**Abstract.** Running context-based systems with a fixed infrastructure involves substantial investments. There have been efforts to replace those systems with self-organizing ones. Therefore, recent systems use peer-to-peer (P2P) technology as a basis. Context-information is bound to a specific location and thus should be stored on a nearby node. Common P2P algorithms use one-dimensional ID spaces. However, locations have at least two coordinates, namely x and y. We use space-filling curves to map the two-dimensional area onto the one-dimensional ID space. In this paper we discuss the suitability of different space-filling curves for the average case and for stochastic scenarios.

## 1 Introduction

Context-based systems have become more and more popular recently as they can be used for many different scenarios. Running such a system includes dealing with huge amounts of data and requires an appropriate infrastructure for hosting this data. Providing this infrastructure is coupled with investments in hardware, bandwidth, and manpower for management tasks. Therefore, newly more and more projects try to reduce their costs by realizing context-based systems using peer-to-peer (P2P) technology. These systems are self-organizing, hence liberating the operators from network managing. They are decentralized, which makes them independent of any given infrastructure. Furthermore, as each participant contributes own resources (CPU power, disk space, bandwidth, ...) the costs are spread over all peers, making the system fair. However, there are still some challenges when using peer-to-peer systems as an overlay network for context-based applications. Context is related to a specific object (person, building, ...), which is located somewhere on this planet. Therefore, the appropriate coordinates are 3-dimensional (latitude, longitude and altitude). Modern P2P systems however use a ring structure to organize themselves, a node solely knows its neighbors in the circular ID space (1-dimensional). To counter this problem, we need a dimension reduction to map the multi-dimensional data onto the ring. In addition, we want to optimize our P2P system towards locality. Due to latency and security issues it would be best to store information concerning a certain location

[*] The presented work has been funded by DFG Excellence Center 627 "Nexus" and DFG SPP "Organic Computing".

on a computer close to it. The peers in our system therefore have to identify their own location and the location to which the context-information is relevant. Thus, we can ensure that the opening hours of the Berlin Pergamon museum are stored on a peer in Berlin and not in Tokyo. To solve these two major problems we present an algorithm, which bases on Pastry [1]. It optimizes the data distribution for a close relation between the location the information is about and the location the data is actually stored through the use of space-filling curves (SFC). With our approach we want to extend the functionality of the Nexus project [2][3][4] through the use of P2P technology. The Nexus system provides for answering three types of queries: range queries, nearest neighbor queries and position queries. In this paper we present a method of answering nearest neighbor queries in a geographically optimized P2P network.

## 2 World Partitioning

Our algorithm bases on Pastry, which is organized in a ring structure. In plain Pastry each peer is assigned a unique ID through the use of a hash-function over the IP address. To answer a query the according key ID is calculated and sent to the node with the ID "closest" to it (prefix routing). However, the method for assigning IDs to nodes as used in Pastry is sub-optimal for the use in our scenario. It uses a hash function to calculate the node ID, which results in the fact that nodes with adjacent node IDs may be diverse in geography. This is highly undesirable, as queries might have to travel long distances. In contrast, our algorithm optimizes the location where data is stored.

To achieve our goal of assigning an ID corresponding to the location of the node, we divide the world into equally-sized zones [5]. For the remainder of the paper we regard the world as a 2-dimensional quadratic map due to simplicity (see Figure 1). Still, it poses no problem to enhance our model to support a third dimension (additional bits are concatenated with the IDs). Each zone then represents the smallest area a single node can be responsible for. The more zones, the more nodes can be supported. We believe using 56 bit for encoding the node ID will be sufficient for most applications as a single zone is then below $0,007m^2$. Upon booting a node determines its geographical position and calculates its node ID to participate in the P2P ring.

More precisely, starting from a unit square representing the whole world, the square is subdivided into four mini-squares in each further step. We call the squares that emerge after the first division *order-1* squares, the squares after the second division *order-2* squares and so on. After the partitioning is finished, each zone includes at most one node. In reference to the unit square, each zone has a determined set of coordinates. These will then be used as input for an ID assignment algorithm to calculate the ID the node will use to enter the overlay network. Due to the fact, that there is at most one node per zone, no node ID will occur twice. For the time, the positioning mechanisms are not accurate enough to match the resolution of the grid, the node ID is expanded with an additional set of random digits. In case a node should anyhow choose the same

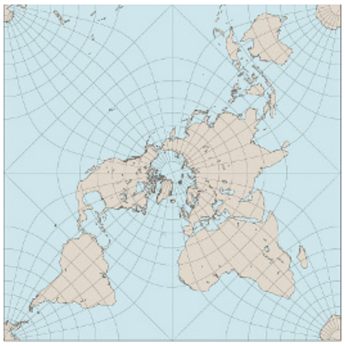


**Fig. 1.** 2D World Model (Map projection after Peirce [6])

ID as another one, this can be easily detected and corrected by choosing a new one.

The ultimate goal is to have a minimum difference between actual geographic distance and the distance in ID space. More precisely, if the geographic distance between two random nodes is small, we expect their nodes IDs to be numerically close, too. Thereby, we limit the distance messages have to travel. However, the problem is, that finding the optimal ID assignment scheme is impossible.

This opens up several methods on how to calculate the node ID. Space-filling curves have proven to be well suited for dimension-reduction, which is why we use space-filling curves for assigning the node IDs. Yet, [7] proved that it is impossible to create a SFC, such that all points close in the multi-dimensional space are close on the curve, too. Therefore, we compare different curves by identifying their characteristics and conclude their usefulness in our scenario. In contrast to other works [8][9] we investigate locality in small-world population scenarios, too.

## 3 Space-Filling Curves

The simplest way to map an index curve onto an area is to superimpose the curve in an s-shaped way. This approach is not very promising, as the expected error is elevated. This is due to the fact that geographically close nodes may have a large discrepancy in their node IDs. For example, the first element of the first two rows are geographically close, but their node IDs differ by two times the edge length of the area as can be seen in Figure 2.

The popular space-filling curves are based on a different approach. G. Cantor proved that it is possible to bijectively map the interval $I = [0, 1]$ onto the space $\Omega = [0, 1]^d ; d \in N$ [10]. G. Peano then defined, that a space-filling curve $f : I \longmapsto \Omega \subset R^d$ is surjective and $\Omega$ is positive in $R^d$. This definition led to the development of several curves, which can be distinguished by their level of locality. Locality indicates the relationship of the distance of two

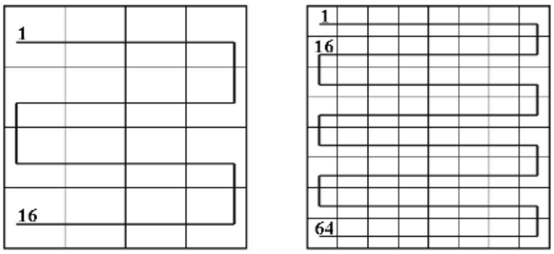


**Fig. 2.** Trivial S-shaped Curve

points $p1, p2 \in I$ on the SFC and its image $f(p1), f(p2) \in \Omega$ in the multi-dimensional space. We search a curve with good locality properties or more precisely: $p1 \approx p2 \Leftrightarrow f(p1) \approx f(p2)$. For the remainder of the paper, we state, that we solely regard approximations of the curves, as we use them for assigning IDs to nodes only. It is difficult to estimate the locality exclusively by analyzing the geometric representation of the curve. However, it is obvious that long edges worsen locality, as there is larger difference in geography and only a minimal difference in ID space, which for example is the problem of the trivial S-shaped curve. In the following we compare a selection of space-filling curves, that promise good locality. Several sources present further SFCs [10][11], however, most curves show a certain level of similarity or are not suited for our purpose.

### 3.1 Peano

Peano presented the first curve fulfilling the definition above. The partitioning of that curve differs from all other well-known SFC. On each partitioning step, each zone is split into three slices in each dimension. Therefore, the unit square is divided into nine zones in the first step. The curve then follows the initial mapping as in Figure 3 (left image). The distances between two adjacent nodes on the curve is homogeneous, thus the curve is often applied in other research areas, such as cache-optimizations [12].

### 3.2 Lebesgue

One of the simplest space-filling curves is described by the standard *Lebesgue Curve* [10] (also known as Z-order curve). It allows for an easy conversion from

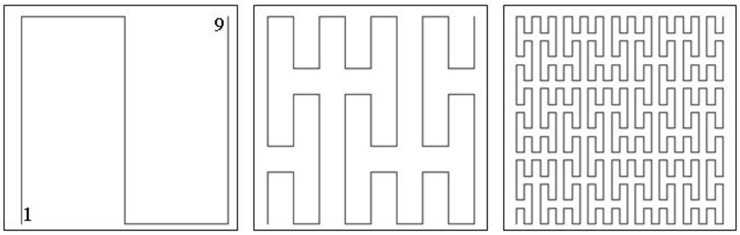
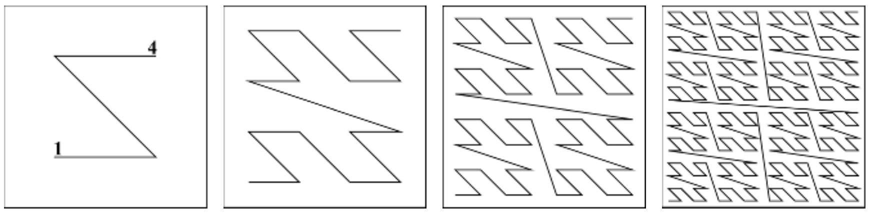


**Fig. 3.** Peano Curve (Order 1-3)

the indices in the 2-dimensional matrix to the 1-dimensional index along the curve, which makes it extremely attractive for the use in our scenario. The index of a point p is calculated as follows:
$x = (x_0x_1...x_n), y = (y_0y_1...y_n) \rightarrow P_{index} = y_0x_0y_1x_1...y_nx_n$ The resulting curve is self-avoiding, but contains some long edges, which is not optimal for our purpose.



(a) (b) (c) (d)

**Fig. 4.** Lebesgue Curve (Order 1-4)

## 3.3 Hilbert

Shortly after Peano presented his SFC, David Hilbert proposed another important curve. In the area of mesh-indexing the authors of [8] could prove that in the worst case the Hilbert curve provided best geometric locality properties. Its geometric construction starts with the basic "u"-form (left image of Figure 5). The order-2 curve is then generated by shrinking its size such that four copies can be placed on the grid. While the position of the upper two curves matches their final orientation, the lower curves have to be rotated according to their position on the unit square (see middle image of Figure 5). Lastly, the ends facing each other have to be connected, forming the continuous curve. For further orders this procedure is applied recursively to all partial squares.

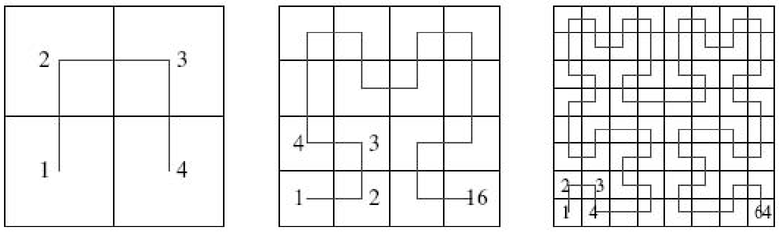


**Fig. 5.** Hilbert Curve (Order 1-3)

## 3.4 Fass II

Fass is an acronym for space-filling, self-avoiding, simple and self-similar curves. Besides the Hilbert and Peano curve we analyze another interesting Fass curve [11], basing on the following Lindenmayer [13][14] parameters: $Angle = 90°$, $Axiom = -L$, $Rules = \{L \rightarrow LFLF + RFR + FLFL - FRF - LFL - FR + F + RF - LFL - FRFRFR+$, $R \rightarrow -LFLFLF + RFR + FL - F - LF + RFR + FLF + RFRF - LFL - FRFR\}$. Similar rules are used in Turtle Graphics [15], where a "minus" indicates a turn by 90 degrees to the left (plus to the right) and an "F" marks the next point or zone in our scenario. Starting with the axiom (initial value), the next order curve is generated by replacing the "L" and "R" according to the rules. The resulting curve resembles the digit four, as can be seen in Figure 6.

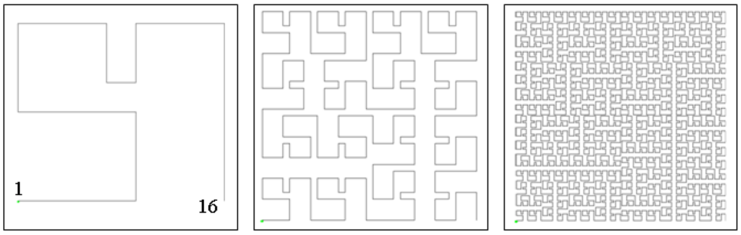


**Fig. 6.** Fass II Curve (Order 1-3)

## 4 Evaluation

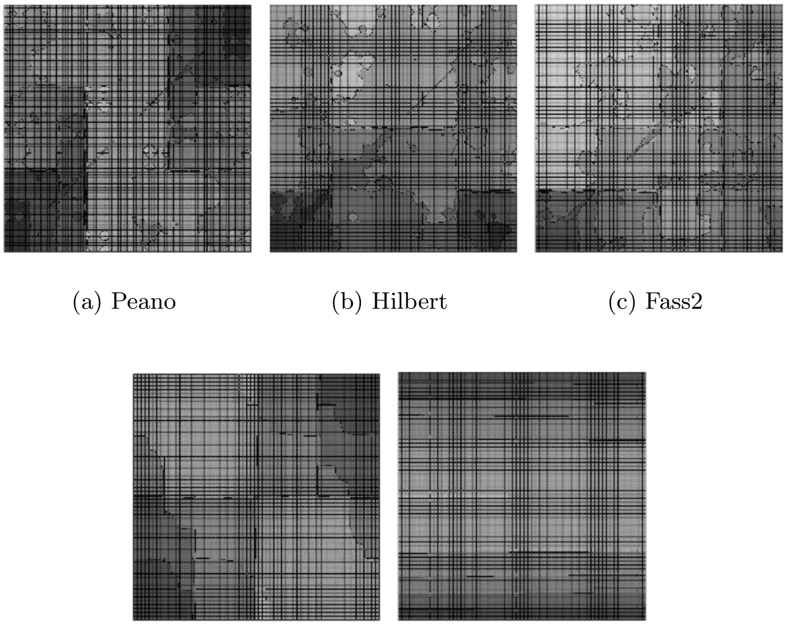
In this section we identify the suitability of the curves for the use in our scenario. Therefore we conducted several tests, measuring the average error distribution, analyzing the locality in dense and sparse populated worlds and testing the behavior in small-world networks.

### 4.1 Mean Error Rate

The mean error rate (MER) reflects the average deviation of the geographic coordinates and the assigned ids. For setup, we assume to have exactly one node in each zone, being assigned an ID as the curve passes by. We then regard each node with every other node and calculate the difference in geometric and ID space. As the curves use different partitioning processes (Peano generates 9 squares whereas some others use 4) we cannot calculate the sum of all these differences for comparison. Therefore, we devise the MER independent of the zone count as we normalize the ID difference and the euclidean distance ($\Delta geo$). We then sum up all these values and normalize this result to devise the MER for every curve as in (1).

$$MER_{curve} = \sum_{i=1}^{n} \frac{\sum_{j=1, j \neq i}^{n} \left| \frac{|i-j|}{n} - \frac{|\Delta geo(i,j)|}{\sqrt{2}n} \right|}{n} \tag{1}$$

Figure 7 illustrates the error distribution for the complete unit square. The darker the zone, the higher its mean error.



(a) Peano (b) Hilbert (c) Fass2

(d) Lebesgue (e) S-shaped

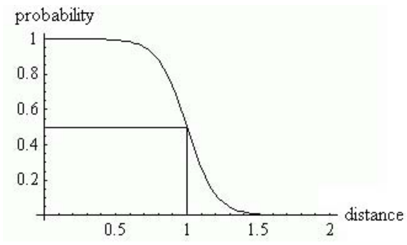
**Fig. 7.** Mean Error Distribution of Space-Filling Curves

The following table shows the different MERs per curve. It shows that the more complex space-filling curves are more efficient than the S-shaped or Lebesgue curves in our scenario.

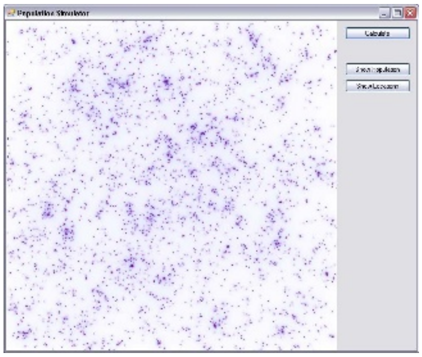
| | S-shaped | Lebesgue | Hilbert | Peano | Fass2 |
|---|---|---|---|---|---|
| Error | 0,33 | 0,33 | 0,26 | 0,26 | 0,27 |

## 4.2 Small-World Populations

In this test we evaluated the curves in a more realistic scenario. Whereas in the latter test we calculated an average over all nodes, we now compare only a selection of nodes. This resembles the fact, that most likely not every zone will contain a node. There will be a some more interesting locations (Hotspots) and therefore, more nodes (Residents) will likely be around. To reflect this natural behavior we used the Fermi-Dirac statistics [16] [17], assigning Hotspots and Residents to zones in the unit square, such that a small-world-like [18] population is mirrored. Small-world models can be used to emulate real-life population scenarios [19]. Nodes (equivalent for people) within a certain proximity of each other form a cluster (e.g. a city). All clusters are randomly distributed and interconnected, thus resembling differently sized population areas. For our simulation, we first assigned one Resident randomly over the area. Further Residents will more likely populate a zone close to a zone, which is already populated, since this zone is more attractive. Afterwards, the Hotspots are placed in the near surroundings of larger Resident gatherings, since it is more likely that there is information to be stored than elsewhere. The curve in Figure 8(a) mirrors the level of attractivity. The very near surrounding zones maintain a high level of attractivity, which is dropping fast with an increasing distance. The unit square will thus contain some metropolitan areas and few scattered nodes, resembling heavy populated cities and less populated back-country. Thus populations as in Figure 8(b) are generated.



(a) Fermi-Dirac gradient



(b) Population Simulator

**Fig. 8.** Generating small-world-like populations

The error rate in this case is calculated as follows. Each Hotspot $i$ determines the node geographically closest $N_{Geo}(i)$ and the node closest in ID space $N_{ID}(i)$. Communication takes place with the node numerically closest to a Hotspot. An error occurs if the geographically closest node is not the node with the numerically closest ID. In this case, another node will wrongfully be contacted to deliver information about the Hotspot. Though we are talking about an error here, the system will nevertheless work. It is just the choice of nodes, which is sub-optimal. Thus, the small-world mean error rate (SMER) results from the euclidean difference between the optimal node $N_{Geo}$ and the wrongfully chosen node $N_{ID}$:

$$SMER_{curve} = \left\{ \sum_{i=1}^{n} \Delta geo\Big(N_{Geo}(i), N_{ID}(i)\Big) \Big| \, i \in Hotspots \right\} \tag{2}$$

Figure 9 shows the error rate of the curves with a population, which increases according to the size of the unit square. It appears, that error level of the more complex space-filling curves (Hilbert, Peano, Fass II) is far lower than that of the

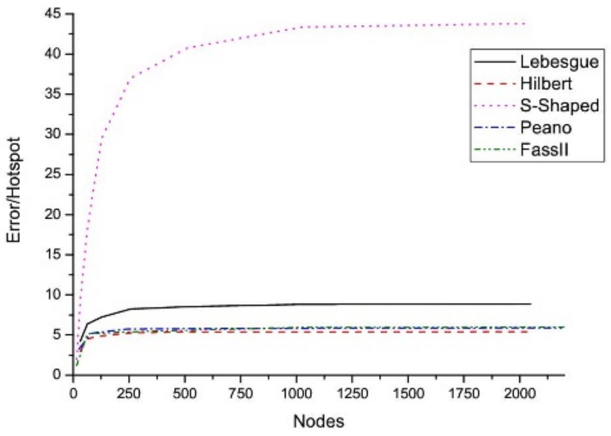


**Fig. 9.** Error per Hotspot

trivial curves. For this simulation we used a grid with up to 2048x2048 zones. When partitioning the world for our peer-to-peer ring, the amount of zones will even be much higher. Therefore, the trivial curves pose no possible solution.

Another interesting aspect is to evaluate the performance of the different curves with varying population levels. Therefore, we kept the amount of Hotspots and Residents constant, regardless the size of the unit square and thus creating dense worlds at the beginning to sparse networks at the end of Figure 10. It becomes obvious, that an efficient node-indexing cannot be achieved by using "simple" curves like the S-shaped or even the Lebesgue curve. This becomes clear when regarding worlds with a sparse population (see result with grid size

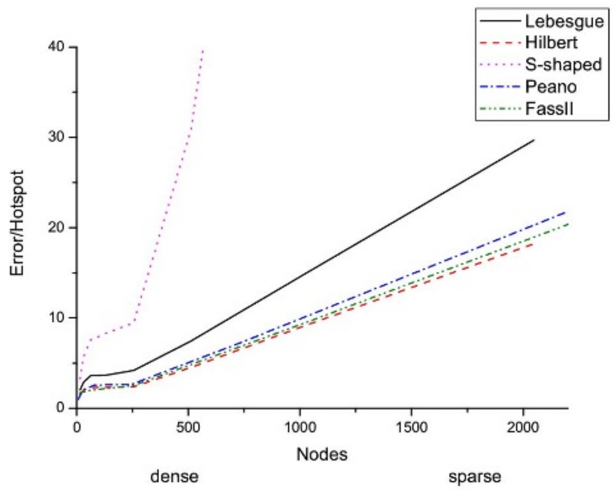


**Fig. 10.** Error per Hotspot with dense and sparse Populations

of 4096x4096). The less Residents there are, the higher the probability to wrongfully choose a geographically farther node over a closer one. The remaining curves perform almost equally, though the Hilbert curve points out the smallest error derivation. Furthermore, the complex curves point out, that their average derivation lies around 1 percent. This value is extremely low, meaning that in the everyday service we would choose the optimal Resident in 99 out of 100 times.

## 5 Related Work

To our knowledge there is no other framework for hosting context-data, which optimizes for locality of their content. However, there has been excessive research in partial aspects of our project.

*P2P Systems:* Protocols in this section are optimized towards the main tasks of P2P systems: insertion, lookup, and deletion of keys. CAN is a popular and efficient representative of this section. Its storage mechanisms are similar to Pastry's, but its routing algorithm differs. CAN's coordinate space is completely logical and bears no relation to any physical coordinate system [20]. Hence it is difficult to host context-information according to its location as peers are responsible for a randomly chosen zone. Furthermore, the coordinate space is partitioned dynamically, which requires many updates of neighborhood nodes in the case of new nodes or node failure. Thus, its runtime of $O(\frac{d \cdot N^{1/d}}{4})$ is worse than Pastry's $O(log(N))$. However, CAN optimizes for routing, whereas we want to optimize for locality of data.

*Data Storage:* This area deals with storage problems and how to distribute large amounts of data among all peers most adequately. One of the largest projects in that area is the OceanStore [21][22] project. Its main research focuses on a utility infrastructure for providing continuous access to persistent information. OceanStore distinguishes between service providers and users subscribing to one of these providers. The providers are comprised of untrusted servers, which raises the necessity to replicate all data on several other servers in order to prevent a loss. Therefore, OceanStore is less suited for hosting context-based information.

*Space-Filling Curves:* In [23] the authors present a P2P information discovery system supporting complex searches using keywords. Their system bases on Chord for the overlay network topology and utilizes the Hilbert SFC for the dimension reduction. However, their main focus lies on the mapping of data elements, which are local in a multi-dimensional keyword space, to indices which are local in the 1-dimensional index space. Two documents are considered local, when their keywords are lexicographically close (e.g. computer and computation) or they share common keywords. This comes at a cost, as using space-filling curves does not guarantee a uniform distribution of data elements in the index space. Therefore, additional load-balancing algorithms have to run to reduce the load of heavily used nodes. Wierum [9] uses a similar index-range metric for comparing the quality of the different curves. However, he intends to use the curves to allow for efficient sorting and searching. Therefore, he only takes the direct neighbors of a zone into account and limits his average error only with regard to those. In our scenario, it is important to find the curve, which minimizes the average error over all zones, as communication will most likely take place between nodes, which are not adjacent.

## 6 Conclusion

In this paper we have presented a method allowing to host context-based information on a self-organizing peer-to-peer system. In contrast to existing context-based systems our algorithm optimizes the data distribution towards geometric locality, keeping the distance information travels short. We showed that finding an optimal curve for node-indexing is not a simple problem, as the S-shaped and even the more complex Lebesgue curve perform poorly. The more complex SFC present far better locality properties, especially the Hilbert curve. As its average derivation error in small-world-like world partitions is around one percent, the curve is perfectly suited for the use in our scenario. We are planning to extend this research to support further query types, such as range queries.

## References

1. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object loaction for routing for large-scale peer-to-peer systems. In: Proceedings IFIP/ACM Middleware 2001. (2001) Heidelberg, Germany.
2. Lehmann, O., Bauer, M., Becker, C., Nicklas, D.: From home to world - supporting context-aware applications through world models. In: PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), Washington, DC, USA, IEEE Computer Society (2004) 297
3. Grossmann, M., Bauer, M., Hönle, N., Käppeler, U.P., Nicklas, D., Schwarz, T.: Efficiently Managing Context Information for Large-scale Scenarios. In: Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications (PerCom2005), IEEE Computer Society (2005)
4. Dudkowski, D., Schwarz, T.: The neXus Homepage. http://www.nexus.uni-stuttgart.de (2005)
5. Knoll, M., Weis, T.: A P2P-Framework for Context-based Information. In: 1st International Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI) at Pervasive 2006, Dublin, Ireland (2006)
6. Furuti, C.A.: Map projections. http://www.progonos.com/furuti/MapProj/CartIndex/cartIndex.html (2006)
7. Gotsman, C., Lindenbaum, M.: On the metric properties of discrete space-filling curves. IEEE Transactions of Image Processing **5**(5) (1996)
8. Niedermeier, R., Reinhardt, K., Sanders, P.: Towards optimal locality in mesh-indexings. In: Fundamentals of Computation Theory. (1997) 364–375
9. Wierum, J.M.: Logarithmic path-length in space-filling curves. In Wismath, S., ed.: Proceedings of the 14th Canadian Conference on Computational Geometry, Lethbridge (2002) 22–26
10. Sagan, H.: Space-Filling Curves. Springer-Verlag, New York, NY, USA (1994)

11. Nielsen, B.: Lindenmayer systemer. http://www.246.dk/lsystems.html (2006) (in Danish).
12. Pögl, M.: Entwicklung eines cache-optimalen 3D Finite-Elemente-Verfahrens für große Probleme. PhD thesis, Technische Universität München, Munich, Germany (2004)
13. Peitgen, H.O., Saupe, D., eds.: The Science of Fractal Images. Springer-Verlag, New York, NY, USA (1988)
14. Alfonseca, M., Ortega, A.: Representation of fractal curves by means of l systems. In: APL '96: Proceedings of the conference on Designing the future, New York, NY, USA, ACM Press (1996) 13–21
15. Abelson, H., diSessa, A.: Turtle Geometry: The Computer as a Medium for Exploring Mathematics. The MIT Press, Cambridge, MA, USA (1981)
16. Tipler, P.A., Mosca, G., Pelte, D.: Physik. 2 edn. Spektrum Akademischer Verlag (2004) (in German).
17. Gutowski, M.W.: Smooth genetic algorithm. Journal of Physics A: Mathematical and General **27**(23) (1994)
18. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393**(6) (1998) 440–442
19. Watts, D.J.: Small Worlds. Princeton University Press, Princeton, NJ, USA (1999)
20. Ratnasamy, S.P.: A Scaleable Content-Adressable Network. PhD thesis, University of California, Berkeley, CA, USA (2002)
21. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: OceanStore: An Architecture for Global-scale Persistent Storage. In: Proceedings of ACM ASPLOS, ACM (2000)
22. Kubiatowicz, J.: The OceanStore Project. http://oceanstore.cs.berkeley.edu/ (2005)
23. Schmidt, C., Parashar, M.: Flexible information discovery in decentralized distributed systems. In: 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 '03), IEEE Computer Science (2003) 226

# Randomized Self-stabilizing Algorithms for Wireless Sensor Networks

Volker Turau and Christoph Weyer

Hamburg University of Technology, Institute of Telematics
Schwarzenbergstraße 95, 21073 Hamburg, Germany
turau@tuhh.de

**Abstract.** Wireless sensor networks (WSNs) pose challenges not present in classical distributed systems: resource limitations, high failure rates, and ad hoc deployment. The lossy nature of wireless communication can lead to situations, where nodes lose synchrony and programs reach arbitrary states. Traditional approaches to fault tolerance like replication or global resets are not feasible. In this work, the concept of self-stabilization is applied to WSNs. The majority of self-stabilizing algorithms found in the literature is based on models not suitable for WSNs: shared memory model, central daemon scheduler, unique processor identifiers, and atomicity. This paper proposes problem-independent transformations for algorithms that stabilize under the central daemon scheduler such that they meet the demands of a WSN. The transformed algorithms use randomization and are probabilistically self-stabilizing. This work allows to utilize many known self-stabilizing algorithms in WSNs. The proposed transformations are evaluated using simulations and a real WSN.

## 1 Introduction

Wireless sensor networks (WSNs) are networks of small, battery-powered, resource-constrained wireless devices equipped with sensors embedded in a physical environment where they operate unattendedly for long periods of time. WSNs pose challenges not present in classical distributed systems, foremost extreme resource limitations, high failure rates, and ad hoc deployment. These boundary conditions and the high number of nodes preclude dependence on manual configuration and control. Inevitably unattended WSNs must self-organize in response to node failures or addition of new nodes, and must adapt to changing environmental conditions. The dynamic and lossy nature of wireless communication caused by the primitive, low-power radio transceivers found in WSNs can lead to situations, where nodes lose synchrony and their programs reach arbitrary states [1]. Traditional approaches to fault tolerance like replication where the effects of faults are shielded or a shutdown and globally reset of the complete network are not feasible. In this work, the concept of self-stabilization pioneered by Dijkstra [2] is applied to WSNs. A distributed system is self-stabilizing if after transient faults, regardless of their cause, it returns to a legitimate state in a finite number of steps regardless of the initial state, and the system remains in a legitimate state until another fault occurs [3,4]. Self-stabilizing algorithms tolerate arbitrary transient failures caused by corruptions of local state, or the disruption of message passing channels, or system resets with unknown initialization. They do not try to handle every individual failure separately, but try to capture the commonality of all failure modes.

Self-stabilization provides a generalized non-masking approach to fault tolerance. This implies that the system experiences the effect of transient faults, in contrast to the replication paradigm. As a consequence applications must be prepared to handle or tolerate these situations. A disadvantage of self-stabilizing algorithms is that a node does not know when the algorithm has stabilized. Self-stabilization fits into the unattended operation style of WSNs, where no outside intervention is necessary. Over the last 20 years many self-stabilizing algorithms have been proposed, quite a few of them are of interest for WSNs: graph coloring [5], articulation points [6], dominating sets [7], depth-first trees [8], and spanning trees [9]. However, the majority of these algorithms is based on models not suitable for the constraints of WSNs: shared memory model, central daemon scheduler, unique processor identifiers, and atomicity. To utilize these algorithms in WSNs, transformations from these strict models into the WSN model are needed. The majority of transformations that have been proposed so far appear to be problem-specific (for an exception see [10]). There is a strong need to devise a general method for systematically transforming algorithms into the realm of WSNs while preserving the stabilization character.

The main contribution of the paper consists of problem-independent stabilization preserving transformations for algorithms that stabilize under the central daemon scheduler in a bounded number of moves in anonymous networks. This is a generalization of problem-specific solutions for graph algorithms such as vertex coloring [5] and minimal independent sets [11]. The key concept is to introduce randomization, as a consequence the transformed algorithms are only probabilistically self-stabilizing. This work enables us to execute self-stabilizing algorithms designed for the central daemon scheduler in WSNs. More importantly, it also helps to develop new and more practical self-stabilizing algorithms for WSNs.

The paper is organized as follows: Section 2 introduces the main concepts of self-stabilization and Section 3 discusses the problems of self-stabilizing WSNs. After that the example used in the experiments covered in Section 6 is presented. Section 5 contains the main contribution, the transformations. In Section 6 preliminary results attained by simulations and through an implementation of the transformation using a real sensor network are presented. The paper ends with related work and a conclusion.

## 2 Self-stabilization

The objective of self-stabilization is to recover from transient faults in a bounded time without any external intervention. The absence of faults is defined by a predicate $\mathcal{P}$ over the global state of the system, $\mathcal{P}$ is defined locally, i. e., based on the local state of each node and the states of their neighboring nodes. More formally, let $N = \{N_1, N_2, \dots, N_n\}$ be a set of sensor nodes and $E \subseteq N \times N$ be a set of bidirectional communication links in a sensor network. The topology of the system is represented as the undirected graph $G = (N, E)$. A set of local variables defines the local state of a node. By $s_i$, we denote the local state of node $N_i \in N$. A tuple of local states $(s_1, s_2, \dots, s_n)$ forms a *configuration* of the sensor network and defines the global state. Let $\Sigma$ be a set of all configurations. A *system* is a pair $(\Sigma, \rightarrow)$, where $\rightarrow: \Sigma \times \Sigma$ is a transition relation. An *execution* is a maximal sequence $c_0, c_1, c_2, \dots$ of configurations such that $c_i \rightarrow c_{i+1}$ for each $i \geq 0$.

A transition is caused by the execution of a program on a node (all nodes run the same program). Programs consist of rules of the following kind:

$$\begin{aligned} \textit{precondition}_1 &\longrightarrow \textit{statement}_1 \\ \textit{precondition}_2 &\longrightarrow \textit{statement}_2 \\ &\dots \end{aligned}$$

The preconditions are Boolean expressions based on the state of a node and the states of its neighbors only (i. e., no global view of the network). The semantics of a program is that whenever a node executes, it executes the statements corresponding to a rule whose precondition evaluates to true. The statements of a rule can only change the local state. It is assumed that reading the states of the neighbors is atomic. The execution of a selected statement is also assumed to be atomic. If more than one precondition is satisfied, then one of them is chosen non-deterministically. A *move* of a node is the execution of a rule. A rule is called *enabled* if its precondition evaluates to true, otherwise it is called *disabled.* A node is called *enabled* if at least one of its rules is enabled.

A configuration $c \in \Sigma$ is called *legitimate* relative to $\mathcal{P}$ if $c$ satisfies $\mathcal{P}$. Let $\mathcal{L} \subseteq \Sigma$ be the set of all legitimate configurations. A system $(\Sigma, \rightarrow)$ is *self-stabilizing* with respect to $\mathcal{P}$ if the following two conditions hold:

1. If $c \in \mathcal{L}$ and $c \rightarrow c'$ then $c' \in \mathcal{L}$ (closure property).
2. Starting from any configuration $c \in \Sigma$ every execution reaches $\mathcal{L}$ within a finite number of transitions (convergence property).

Self-stabilization models the ability of a system to recover from failures under the assumption that they do not continue to occur forever (*eventual-quiescence*). To model long periods of time during which the system operates without errors, it is assumed that eventually the system enters a last operational interval that is infinitely long in which there are no more faults occur. This interval is called the *final interval.* When a fault occurs the system enters a new configuration and the algorithm restarts from this configuration. The same is true for changes of the topology, i. e., adding or removing node or links.

Designing self-stabilization for anonymous networks under the distributed scheduler is a difficult task and for some problems the non-existence of such algorithms has been proven. A distributed scheduler may select two enabled neighboring nodes to execute at the same step, and as a result both nodes may be enabled thereafter. It is not difficult to see that if any two neighboring nodes never execute at the same step, the computation is equivalent to the centralized scheduler. By local mutual exclusion, execution of two neighboring processes at the same step is disabled (see [12,10] for example). If the nodes have unique identifiers, then they can often be used for this purpose, e. g., [5]. Another solution is to use randomization. A system is called *randomized*, if the execution of an enabled node depends on the outcome of a random experiment. A randomized system $(\Sigma, \rightarrow)$ is said to be *probabilistically self-stabilizing* with respect to a predicate $\mathcal{P}$ if it satisfies the closure property as defined above and there exists a function $f : \mathbb{N} \rightarrow [0, 1]$ satisfying $\lim_{k\rightarrow\infty} f(k) = 0$, such that the probability of reaching a legitimate configuration, starting from an arbitrary configuration within $k$ transitions, is $1 - f(k)$ (probabilistic convergence property).

The execution of the transitions of the enabled nodes is controlled by a scheduler. A *schedule* $S$ is a sequence $S_1, S_2, \ldots$ of non-empty subsets of enabled nodes called *rounds*. All nodes in $S_i$ may execute their moves in parallel, but the first move in $S_i$ can only be executed after the last move of $S_{i-1}$ has finished. At the beginning of every round, all nodes evaluate the preconditions of their rules and a subset of the enabled nodes is selected. Then all selected nodes execute the statement of a single enabled rule. Schedules are an abstraction used to model the semantics of concurrent execution, they are not an implementation requirement. Schedules are restricted to satisfy certain fairness and atomicity properties. A scheduler is *fair* if for any schedule $S$ it selects, for all $p \in N$, for infinitely many values of $i$, $p \in S_i$ holds. A *central daemon* scheduler is one that satisfies $|S_i| = 1$ for all $i$; it models the serial activation of one process at each step. A *distributed daemon* scheduler satisfies $|S_i| \leq n$ for all $i$, i. e., all enabled nodes may execute their statements in parallel.

## 3 Self-stabilizing WSNs

Wireless sensor networks are inherently fault-prone due to the shared wireless communication medium: message losses and corruptions due to fading, collisions, and hidden-terminal effects are the norm rather than the exception [1]. In many cases nodes can communicate with each other only with a very low probability. Moreover, node failures due to crashes and energy exhaustion are commonplace. These faults can drive a portion of a WSN to be arbitrarily corrupted and hence to become inconsistent with the rest of the network. Since WSNs are deployed in remote locations, in-situ maintenance is not feasible and therefore sensor network applications should be self-healing. Self-healing ensures eventual compliance with a specification upon starting from a corrupted state. A big challenge for fault-tolerance is the energy constraint of the nodes. Applications cannot impose an excessive communication burden on nodes. As a consequence, self-healing of WSNs must be local and communication-efficient.

Self-stabilization is a specific form of self-healing that has many advantages for WSNs. Large scale WSNs will be operating over a longer period of time and additional nodes can be added at any time. Self-stabilizing eliminates the overhead of initialization all nodes in a consistent manner, actually state variables need no initialization at all. The software of the nodes in a long running network needs an upgrade over time, the software may be distributed over the wireless medium. While switching to the new version, it will be impossible for all nodes to simultaneously switch software. A self-stabilizing system guarantees completion of this change in a finite number of operations. Once a node recovers after failing (e. g., after a temporary power outage or due to a memory crash) its state may be inconsistent with the rest of the system. Self-stabilization also guarantees consistency of the nodes in this case. And finally, errors in transmissions leading to corruption of data may be handled by self-stabilizing algorithms as well.

Most research on self-stabilizing algorithms has concentrated on the central daemon scheduler. The main reason is that proving correctness of algorithms is much easier than in the case of a distributed scheduler. But the concept of a central daemon is against the spirit of Distributed Systems since it does not allow for concurrency. Also, it is difficult to implement this scheduler in a WSN. In the shared memory model each process can read the local states of all neighbor processes without delay. This model is not suitable for sensor networks, instead broadcasts, the communication primitive of wireless networks, should be used. Herman introduced the *cached sensornet transformation* (CST) as an alternative model for WSNs [13]. Let each node $N_i \in N$ in the WSN have a single variable $s_i$ that completely represents the local state of the node. Let $N_i$ have for each neighbor $N_j$ a variable $\nabla_i s_j$, which denotes a cached version of $s_j$. Atomically, whenever $N_i$ assigns a new value to $s_i$, node $N_i$ also broadcasts the new value to its neighbors. Whenever a node $N_j$ receives a new value for $s_i$, it immediately (and atomically) updates $\nabla_j s_i$. Because sending and receiving operations are exclusive in the nodes, we suppose that receiving a cache update message cannot interfere with concurrent assignment and broadcast by the receiving node. To use a self-stabilizing algorithm under this transformation, the rules have to be changed: at every node $N_i$ each reference to $s_j$ is replaced with $\nabla_i s_j$ for all $j$. The execution of a statement does not modify the cache and receiving a broadcast message only changes the cache and not the state of the node.

A system is called *cache coherent* if $\nabla_j s_i = s_i$ for all $(N_j, N_i) \in E$. Cache coherence is invariant under local broadcast provided that no messages are lost. Let $\mathcal{A}$ be a self-stabilizing algorithm under the central daemon scheduler. Then the CST transformed algorithm is also self-stabilizing under the central daemon scheduler provided the initial state of the execution was cache coherent and no messages are lost or corrupt.

## 4 Example

Clustering is a useful technique to control the topology of WSNs. Clustering algorithms should satisfy two properties: In order to allow efficient communication between nodes, every node should have at least one clusterhead in its neighborhood and no two clusterheads should be within each others mutual transmission range. The latter property greatly facilitates the task of establishing an efficient

MAC layer, because clusterheads will not face interference. These properties lead to the concept of a maximal independent set in a graph: An independent set (IS) $I$ of $G$ is a subset of $N$ such that no two nodes in $I$ are neighbors. $I$ is a maximal independent set (MIS) if any node $v$ not in $I$ has a neighbor in $I$. The following self-stabilizing algorithm has been proposed by several authors. Each node has a Boolean variable *in*. A state is called legitimate if the set of nodes $v$ with $v.in = true$ forms a MIS of $G$. The rules of the algorithm are:

```
if $(in = \textbf{\textit{false}} \wedge \forall\ neighbors\ v : (v.in = \textbf{\textit{false}}))$  $\longrightarrow$  $in := \textbf{\textit{true}}$
if $(in = \textbf{\textit{true}} \wedge \exists\ neighbor\ v : (v.in = \textbf{\textit{true}}))$  $\longrightarrow$  $in := \textbf{\textit{false}}$
```

It was been proved in [14] that this algorithm self-stabilizes under the central daemon scheduler after at most $2n$ moves. There is no guarantee about the quality of the produced MIS, i. e., there may exist another MIS containing more nodes. Clearly this algorithm does not stabilize under the distributed daemon scheduler. Suppose the variable *in* of all nodes initially has the value *true*. Then the first rule of all nodes is enabled. If all nodes execute, then the value changes to *false* for every node and all nodes are enabled again. This process continues forever. Applying the transformation described in the next section to this algorithm yields a randomized algorithm that is probabilistic self-stabilizing under the distributed daemon scheduler.

## 5 Transformation of Self-stabilizing Algorithms

This section presents techniques to transform algorithms that self-stabilize under the central daemon scheduler such that they can be used in WSNs under the distributed daemon scheduler. The WSNs under consideration are anonymous networks, i. e., nodes have no globally unique identifiers. Nodes must be able to distinguish their neighbors. It is assumed that messages are not corrupted (e. g., by using error correcting codes). For the time being it is also assumed that no messages are lost. This restriction will be removed in the following part.

Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler in a finite number of moves. The main issue with the distributed daemon scheduler is to enforce the separation of the executions in consecutive rounds. If all nodes have a common understanding of time, then the rounds can be organized under the assumption that the execution times of the statements are bounded by a finite constant. There are several proposals for time synchronization in WSN, e. g., [15]. The first step is to apply the cached sensornet transformation to $\mathcal{A}$, call the resulting algorithm $\mathcal{A}^{\mathcal{C}}$. The main issue with $\mathcal{A}^{\mathcal{C}}$ in WSNs is to guarantee the atomicity of the moves as described above. To deal with concurrent moves within a round, a random element is introduced. It is assumed that each node is equipped with a random number generator *rand*. Furthermore, all nodes have agreed on a constant $p \in (0, 1)$. Let $Rule_{\mathcal{A}^{\mathcal{C}}}$ be the set of rules of algorithm $\mathcal{A}^{\mathcal{C}}$, then for each rule

$$precondition \longrightarrow statement$$

from $Rule_{AC}$ construct a new rule:

$$\textit{precondition} \longrightarrow \textbf{if } (\text{rand}() < p) \textbf{ then } \textit{statement}$$

Call the randomized algorithm for this new set of rules $\mathcal{A}^{\mathcal{CR}}$. Note that the execution of a statement now involves a call to the random number generator and that a statement of an enabled rule is not necessarily executed in the current round. Algorithm $\mathcal{A}^{\mathcal{CR}}$ has the following property.

**Theorem 1.** *Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler after a finite number of moves with respect to a predicate $\mathcal{P}$. If the initial configuration is cache coherent, then algorithm $\mathcal{A}^{\mathcal{CR}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under the distributed daemon scheduler provided that all broadcasts are reliable.*

*Proof.* Suppose that algorithm $\mathcal{A}$ stabilizes after at most $M$ moves. Consider the execution of algorithm $\mathcal{A}^{\mathcal{CR}}$. Since the initial configuration is cache coherent all following configurations are also cache coherent since all messages are sent successfully. Let $(S_i)$ be a schedule under the distributed daemon scheduler. Assume that $S_i \neq \emptyset$ for all $i > 0$ (otherwise $\mathcal{A}^{\mathcal{CR}}$ stabilizes). The probability that exactly one node is executed in round $S_i$ is equal to $\beta_i = c_i p(1-p)^{c_i-1}$ where $c_i = |S_i| \leq n$. Let $(b_i)$ be a binary sequence where $b_i = 1$ if exactly one node executes during round $i$ and 0 otherwise. Note that $\text{Prob}(b_i = 1) = \beta_i$ and $\text{Prob}(b_i = 0) = 1 - \beta_i$ and that $|\{\beta_i \,|\, i \in \mathbb{N}\}| \leq n$. If the sequence $(b_i)$ has a subsequence $b_{j+1}, b_{j+2}, \ldots, b_{j+M}$ of length $M$ where all elements have the value 1, then $\mathcal{A}^{\mathcal{CR}}$ stabilizes after round $j+M$, because this sub-schedule is equivalent to a schedule under the central daemon scheduler. Let $f(k)$ be the probability that such a subsequence is not contained in $b_1, b_2, \ldots, b_k$. Then by Theorem 3 (see Appendix A)

$$\lim_{k\to\infty} f(k) = \lim_{k\to\infty} P_M(k) = 0$$

and hence $\mathcal{A}^{\mathcal{CR}}$ is probabilistic self-stabilizing under the distributed daemon scheduler. $\square$

Note that the requirement that the initial configuration is cache coherent cannot be dropped. The problem is that if there are nodes with non-coherent caches then there can be situations where no nodes are enabled and the predicate $\mathcal{P}$ is not satisfied at the same time. Theorem 1 does not make a statement about the rate of stabilization, e. g., about the probability that the algorithm stabilizes in $k$ moves. The expression $1 - f(k)$ is a lower bound for this probability, but we have no explicit expression for $f(k)$. Also the values of $c_i$ are specific to the algorithm under consideration. The lower bound $1-f(k)$ can be improved using the following observation. A sub-schedule $S_j, S_{j+1}, \ldots, S_t$ is equivalent to a schedule under the central daemon scheduler if the nodes of each round of that sub-schedule that execute form an independent set. The proof of Theorem 1 covers the special case of only a single node executing. Note that if two non-neighboring nodes, that have a common neighbor, execute, then the probability that their broadcasts during the sensornet transformation collide at the common neighbor is high.

**Unreliable Communication.** In the following the assumption about the reliability of broadcasts is dropped, i. e., not all neighbors receive a broadcasted message. Let $q$ be the probability that a message is successfully transmitted from one node to another. We make the assumption that all transmissions are independent. Note that the loss of a message is not regarded as a transient fault, it is a possible behavior of the system that is tolerated by the algorithm. Hence, messages may also be lost during the final interval. The argument of the proof of Theorem 1 can also be applied in this case. The probability that exactly one node is executing in round $i$ and that at the same time all broadcasts of this node succeed is equal to

$$\beta_i = p(1-p)^{c_i-1} \sum_{j \in S_i} q^{d_j}$$

where $d_j$ is the degree of node $j$. The set of all $\beta_i$ is again a finite set ($n^{\Delta}$ is an upper bound). From Theorem 3 it follows that the probability that this algorithm halts after $k$ rounds converges to 1 with increasing $k$. But that does not necessarily mean that the algorithm is probabilistic self-stabilizing under the distributed daemon scheduler. The problem is that the algorithm may reach a non-cache coherent configuration in which no node is enabled. To overcome this problem each node broadcasts the values of its public variables to its neighbors periodically at the beginning of every round, call this algorithm $\mathcal{A}^{\mathcal{CRP}}$. The probability that exactly one node is executing in round $i$ and that at the same time all broadcasts succeed is equal to $\beta_i = q^m c_i p(1-p)^{c_i-1}$ where $m$ is the number of links in the network. The proof of the following theorem is similar to the proof of Theorem 1. Note that the initial configuration does not need to be cache coherent.

**Theorem 2.** *Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under the central daemon scheduler after a finite number of moves with respect to a predicate $\mathcal{P}$. Let the probability that a message is successfully transmitted from one node to another be fixed and assume that these events are independent. Then algorithm $\mathcal{A}^{\mathcal{CRP}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under the distributed daemon scheduler.*

Broadcasting the public variables at the beginning of every round causes two problems: It increases the total energy consumption and if all nodes make their broadcast at the beginning of a round, many collisions will occur, probably leading to a prolonged stabilization time. The second problem can be mitigated if the nodes broadcast their data after a random waiting period. Another solution is that nodes do not broadcast their data in each round, but make this decision dependent on the outcome of a random experiment. Call this new algorithm $\mathcal{A}^{\mathcal{CRPP}}$. Let $r$ be the probability that a node makes a broadcast. Then the probability that exactly one node is executed in round $i$ and that at the same time all broadcasted messages are successfully received is equal to

$$\beta_i = r^n q^m c_i p(1-p)^{c_i-1}$$

where $n$ is the number of nodes in the network. Using Theorem 3 it can be shown that Theorem 2 also holds for algorithm $\mathcal{A}^{\mathcal{CRPP}}$. To further reduce the number of messages sent, the probability of a broadcast could be decreased in every round after a state change, e. g., by reducing the probability to 50%. But then it is no longer possible to prove the probabilistic self-stabilization behavior.

**Periodic Broadcasting with Implicit Acknowledgments.** Once all neighbors of a node know the current state of the node, the node can suspend broadcasting until the state of the node changes again. In order to implement this technique a node needs the information that all neighbors know its current state. To realize this task, nodes include in their broadcasts the latest received states of all neighbors. This way a node can find out whether its current state is known to all neighbors and may then stop the periodic broadcasting. The node still needs to perform broadcasts to signal other nodes that it received their current state. The following code illustrates this procedure. Nodes are in one of two modes: `Broadcast` and `Suspend`, initially the mode is `Broadcast`. Also every time the state of the node changes, the mode immediately changes to mode `Broadcast`.

```
▷ Mode Broadcast
while not all neighbors have acknowledged current state do
    broadcast
end while
mode ← Suspend
```

```
▷ Mode Suspend
if received broadcast from neighbor then
    broadcast
end if
```

Call this new algorithm $\mathcal{A}^{\mathcal{CRPPA}}$. Theorem 2 still holds for this algorithm since the modifications have no influence on the stabilization behavior. An increase in packet size is the price for reducing the number of messages. Larger packets result in larger transmission times and may lead to more collisions, which may slow down the stabilization process. The main advantage of this approach is that after the system has reached a legitimate state, no broadcast messages are needed until the next transient fault.

**Unidirectional Links.** WSNs suffer from unidirectional links where one sensor can communicate with another with a high probability although the probability of the reverse communication is very low. Unidirectional links are not considered useful in the context of WSNs, the reason is the lack of efficient MAC layer protocols that work with unidirectional links (e. g., both RTS-CTS and ACK based schemes cannot be used directly). Almost all self-stabilization algorithms are defined for bidirectional links only. As an example, consider the algorithm presented in Section 4. If this algorithm is executed on a system with unidirectional links then the result is no longer a MIS. We therefore propose to exclude unidirectional links from being used by self-stabilization algorithms. To meet this end the neighborhood protocol in use should discard such links. The protocol described in [1] can be used for this purpose, it can also be combined with the periodic broadcasts in order to reduce the total number of messages sent. A link that continuously shows a low quality is discarded by this protocol, at the same time new links are accepted as they appear. These events have to be regarded as faults and therefore cannot occur during the final interval. In order to avoid links to appear and disappear frequently over time, neighborhood protocols have to find a balance between agility and stability. If the intervals between faults of this kind are long enough, the proposed algorithms may stabilize during an interval, otherwise the stabilization process will be disrupted considerably. Since the different algorithms have different stabilize times a general statement about this kind of stability is impossible.

**Failing Nodes.** In the following the case of completely failing nodes is considered. If a node fails it stops broadcasting its state, but neighboring nodes continue to regard this node as a neighbor. The remedy is to associate with each cache value $\nabla_i s_j$ of a node $N_i$ a *time to live* (TTL) value. The TTL value is renewed every time the node receives a message with the state of neighbor $N_j$ and it is decreased every round in which no such message is received. If a cache value $\nabla_i s_j$ is not confirmed within TTL rounds it is discarded and $N_i$ is no longer regarded as a neighbor of $N_j$. As a consequence a disabled node might get enabled. Thus, the situation is comparable to a discarded link and the discussion from the last section applies.

The concept of TTL values cannot be used in combination with the above described technique to limit the periodic broadcasts with implicit acknowledgments, i. e., algorithm $\mathcal{A}^{\mathcal{CRPPA}}$. The problem is that a node can no longer distinguish a node that suspended broadcasting from a failed node. As a consequence, the failed node could stay permanently in the neighborhood list of a node. This may in turn lead to a non-coherent cache. But TTL values can be combined with algorithm $\mathcal{A}^{\mathcal{CRP}}$. The value of TTL should be large enough to distinguish the case of a failed node from a node that is temporary not able to communicate with its neighbor. Otherwise nodes with an instable link may disappear and appear repeatedly in the neighbor list of a node with negative consequences for the stabilization process. The handling of newly introduced nodes requires no special treatment. In case the join and leave rates are low, the algorithm may stabilize during intervals with fixed topology. This kind of stabilization behavior depends on the number of moves required to stabilize the system after the failure/introduction of a node. The technique of TTL values can also be combined with algorithm $\mathcal{A}^{\mathcal{CRPP}}$. In this case the relationship between TTL and $r$, the probability that a node makes a broadcast, needs to be carefully tuned. Otherwise, the algorithm will not stabilize during intervals with fixed topology. A detailed analysis of these situations is outside the scope of this paper.

## 6 Experiments

The stabilization behavior of the proposed transformations were further analyzed in a series of experiments with a MIS algorithm. The experiments are based on simulations and on a real WSN. Let $\mathcal{A}$ be the MIS algorithm based on the two rules presented in Section 4. The stabilization behavior of algorithm $\mathcal{A}^{\mathcal{CR}}$ was analyzed in a simulation. At the beginning of each round the set $N_{\text{en}}$ of enabled nodes was determined and after this step all nodes in $N_{\text{en}}$ executed the statements of the enabled rule (without checking whether the node was still enabled). As a consequence a node may execute a rule, even though it is no longer enabled. The reason for this mode of operation is to simulate the interleaving execution of the nodes. The algorithm was run on several graph classes:

$G_{n,q}$: Graphs with $n$ nodes and any pair of nodes is connected by an edge with probability $q \in [0, 1]$.
$K_n$: Complete graphs with $n$ nodes.
$U_{n,d}$: Unit disc graphs with $n$ nodes where the locations of the nodes are randomly selected in a square of side length $d$.

The Unit disc graph model is included, because it is regularly used in theoretical models of WSNs. For each graph algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ was executed 500 times.

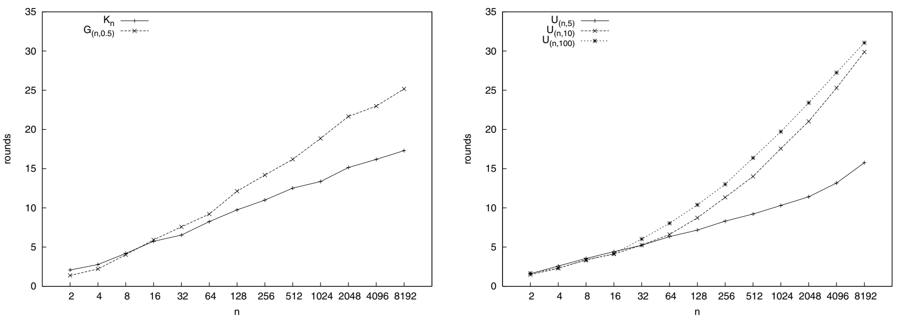


**Fig. 1.** Average number of rounds of algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ before stabilization for various classes of graphs

Figure 1 shows the average number of rounds the algorithm needed to stabilize for the different classes of graphs (note the logarithmic scale of the x-axis). The data shows that the number of rounds for classes $K_n$ and $G_{n,q}$ is roughly proportional to $\log_2 n$. For Unit disc graphs the number of rounds grows slightly faster. Figure 2 shows the average number of the sum of moves executed by all nodes before stabilization. The data shows that roughly $n/2$ moves were needed on the average independently of the class of graphs. Interestingly our experiments also showed that the average number of moves for $\mathcal{A}^{\mathcal{CR}}$ is only slightly higher than in the case of a central daemon scheduler.

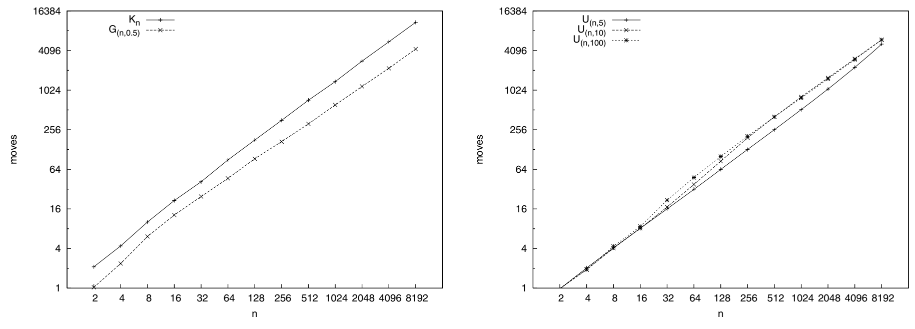


**Fig. 2.** Average total number of moves of algorithm $\mathcal{A}^{\mathcal{CR}}$ with $p = 0.5$ before stabilization for various classes of graphs

The main weakness of the transformations described above with respect to WSNs are the assumptions about the atomicity of the cached sensornet transformation and about the reliability of message delivery. To analyze the impact of these assumptions an experiment with a real WSN was carried out. The experiment was based on algorithm $\mathcal{A}^{\mathcal{CRP}}$. The sensor network consisted of 25 nodes of type ESB, a sensor node platform developed by the Free University Berlin [16]. A node consists of the micro controller MSP 430 from Texas Instruments, the transceiver TR1001, which operates at 868 MHz at a data rate of 19.2 kbit/s, some sensors, and a RS232 serial interface. Each node has 2 KB RAM and 64 KB EEPROM. The lowest layer of the implementation is a synchronization protocol that is used to force the nodes to operate in rounds, each round had a duration of 10 seconds. In order to reduce the possibility of collisions, the *cached sensornet transformation* was implemented such that each node randomly selected an instant during each round and broadcasted its state. At the end of each round the enabled nodes executed an enabled rule with fixed probability $p$. To analyze the influence of the value of $p$ on the stabilization time, the experiment was repeated four times with $p = 0.25, 0.5, 0.75$ and 1.0. The sensor nodes were distributed in a grid-style inside a large lecture hall. Each node could communicate with its immediate neighbors, depending on the position in the rectangular grid a node had between 3 and 8 neighbors. Messages from nodes further away were discarded. The transmissions power of the nodes was reduced such that this topology was realized and no avoidable interference was caused.

The duration of each experiment was 400 seconds, i. e., 40 rounds. Initially no node was a clusterhead (i. e., the variables `in` had the value `false`). The algorithm reached a legitimate configuration for the probabilities $p = 0.25, 0.5$ and 0.75 within the first 19 rounds. Without randomization (e. g., $p = 1.0$) the algorithm did not reach a legitimate configuration within 40 rounds. In this case in every round more than 50% of all nodes made a move, as a consequence the number of clusterheads alternated between a few nodes (0,1 or 2) and about 15-20 nodes. The particular style of initialization of the variable `in` caused this pattern to emerge. In the first round all nodes were enabled, these nodes turned into clusterheads enabling all nodes in round two. At the end of this round no

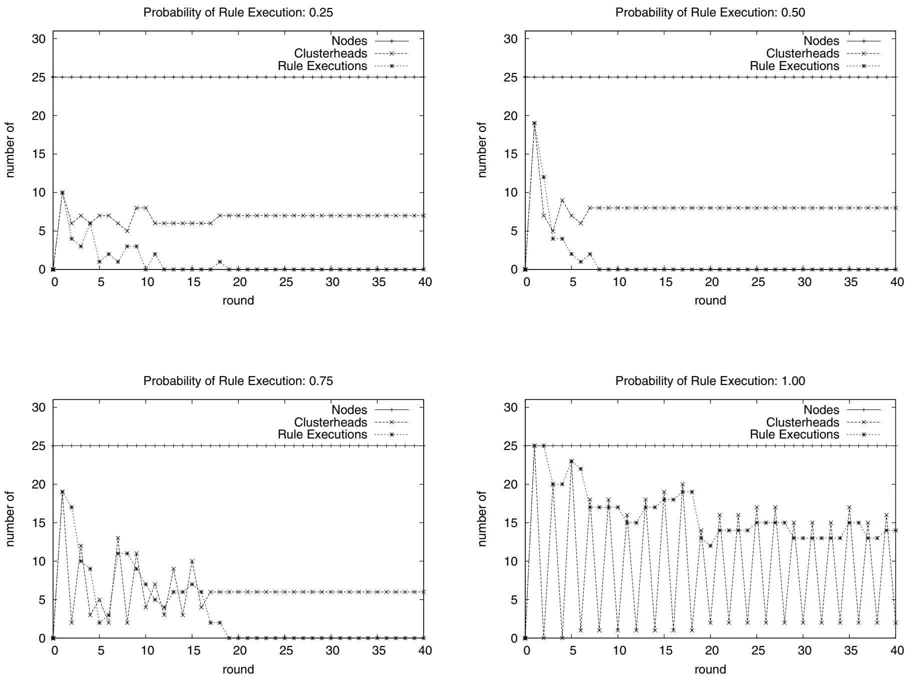


**Fig. 3.** Wireless sensor network with 25 nodes executing algorithm $\mathcal{A}^{\mathcal{CRP}}$ to compute a minimum independent set. The experiment was repeated with four different values of $p$. The diagrams depict the number of nodes, clusterheads and rule executions during the first 40 rounds.

node was a clusterhead. This pattern would have been repeated if no messages had been lost. When a messages is lost after a node has changed its state, the caches at the neighbors of this node are no longer coherent. Because of this phenomenon only 20 nodes made a move in round three.

The other three experiments suggest that the optimal stabilization time is achieved for $p = 0.5$. In this case, the system stabilized after 7 rounds making a total of 44 moves. In case $p = 0.25$ the system could have stabilized after round 12, but the execution of the corresponding rule was deferred for another six rounds until round 18. The reason is that with $p \to 0$ the behavior of the transformation converges to that of the central daemon. The sequence of experiments also shows that the number of moves decreases with the value of $p$: 136, 44, and 36. A detailed analysis is part of future work.

## 7 Related Work

Self-stabilization as a tool to achieve fault tolerance in WSNs was first investigated by Herman [13]. He surveys standard models of self-stabilization and relates these to WSNs. In particular, the *cached sensornet transformation* – a construction that transforms a sensor network to a central daemon model – is introduced. Based on the assumption that all links are bidirectional and that nodes possess a CSMA/CA capability, probabilistic stabilization of an algorithm for maximal independent sets is proven.

The work of Herman is extended by Kulkari et al. [17]. To overcome the read/write model – which is used in many existing algorithms, but not applicable to WSNs – the *write all with collision* model (WAC) is introduced. WAC captures the computations of sensor networks. Intuitively, in one atomic action, a node can update its own state and the state of all its neighbors. If two nodes simultaneously try to update the state of a node, then the state of this node is unchanged. Transformations from existing models to the WAC model and vice versa are presented. To obtain the transformed program that is correct in the WAC model, the nodes are organized in a ring. Such a ring can be statically embedded in any arbitrary graph by first embedding a spanning tree in it and then using an appropriate traversal mechanism to ensure that each process appears at least once in the ring. The nodes execute one after the other as they appear on the ring. For timed systems a transformation using a collision-free time-slot based protocol is presented. Under some assumptions it is shown that if the given algorithm is self-stabilizing then the transformed algorithm is also self-stabilizing for a fixed topology. It is also argued that in some cases for untimed systems self-stabilization preserving transformations into the WAC model are not possible. A distributed algorithm for TDMA slot assignment in WSNs that is self-stabilizing to transient faults and dynamic topology change is presented in [18]. The work of Römer et al. on role assignment in WSNs is in parts related to self-stabilizing algorithms [19]. The authors present heuristics to maintain local cache tables at the nodes. They employ randomized delays in order to avoid temporary inconsistencies due to the lack of sequentialization.

Refining self-stabilizing algorithms using tight scheduling constraints into corresponding algorithms for weaker scheduling constraints, while preserving the stabilization property, has been subject of serious research in the last decade. In most cases the core of the transformations is a self-stabilizing local mutual exclusion algorithm based on unique node identifiers [10]. Kakugawa et al. have developed an algorithm that transforms a serial model program to a distributed model [20]. A timestamp based self-stabilizing concurrency control (CC) protocol is incorporated in the transformed program. After the CC protocol stabilizes, it is guaranteed that for each execution of the transformed distributed model program, there always exists an equivalent execution of the original serial model program. Therefore, if the original program is correct with respect to self-stabilization, the transformed program is also self-stabilizing. A drawback of this approach is the efficiency of transformed algorithms, they require more messages than the ones coded from scratch. Furthermore, these transformations cannot be easily adopted to the case of unreliable broadcasts.

Mizuno and Nesterenko considered distributed systems where all processors have unique identifiers. They propose a procedure to transform a self-stabilizing algorithm under the central daemon scheduler to an equivalent self-stabilizing algorithm that runs on an asynchronous shared memory parallel computing system [12]. Timestamps are used to guarantee mutually exclusive execution of guarded commands among neighbor processes.

## 8 Conclusion

The transformations presented in this paper allow to utilize self-stabilizing algorithms developed for the central daemon scheduler in WSNs. Foremost these are algorithms for coloring, spanning trees, independent and dominating sets [5,6,7,8,9]. Furthermore, this work simplifies the design of self-stabilizing for WSNs, developers can work with the central daemon scheduler and do not have to take into considerations the imponderabilities of wireless communication. The proposed transformations are easy to implement and do not pose much overhead. The result of our simulations and experiments demonstrate for a self-stabilizing MIS algorithm that the transformed algorithm stabilizes quickly. The main drawback of self-stabilizing algorithms for WSNs is the dynamic nature of the communication topology. We conjecture that the transformed algorithms have a very good stabilization behavior if used in conjunction with a stable neighborhood protocol to deal with connections that are susceptible to interference. It remains to analyze the influence of the probability threshold used in the transformations. In future we will investigate criteria for self-stabilizing algorithms operating in dynamic topologies.

## References

1. Turau, V., Weyer, C., Witt, M.: Analysis of a Real Multi-hop Sensor Network Deployment: The Heathland Experiment. In: $3^{rd}$ Int. Conf. on Networked Sensing Systems (INSS06). (2006) 6–13
2. Dijkstra, E.: Self stabilizing systems in spite of distributed control. Communications of the ACM **17**(11) (1974) 643–644
3. Dolev, S.: Self-Stabilization. MIT Press (2000)
4. Schneider, M.: Self-stabilization. ACM Comput. Surv. **25**(1) (1993) 45–67
5. Gradinariu, M., Tixeuil, S.: Self-stabilizing vertex coloration and arbitrary graphs. In Butelle, F., ed.: OPODIS. (2000) 55–70
6. Karaata, M.: A self-stabilizing algorithm for finding articulation points. Theoretical and Mathematical Aspects of Computer Science **10**(1) (1999) 33–46
7. Goddard, W., Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. In: IPDPS, IEEE Computer Society (2003) 240–243
8. Chaudhuri, P.: A self-stabilizing algorithm for minimum-depth search of graphs. Information Sciences **118**(1-4) (1999) 241–249
9. Higham, L., Liang, Z.: Self-stabilizing minimum spanning tree construction on message-passing systems. In: Distributed Computing $15^{th}$ Int. Symposium, Springer LNCS:2180. (2001) 194–208

10. Beauquier, J., Datta, A.K., Gradinariu, M., Magniette, F.: Self-stabilizing local mutual exclusion and daemon refinement. Chicago Journal of Theoretical Computer Science **2002**(1) (2002)
11. Shukla, S., Rosenkrantz, D., Ravi, S.: Observations on self-stabilizing graph algorithms for anonymous networks. In: $2^{nd}$ Workshop on Self-Stabilizing Systems. (1995) 7.1–7.15
12. Mizuno, M., Nesterenko, M.: A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. Inf. Process. Lett. **66**(6) (1998) 285–290
13. Herman, T.: Models of self-stabilization and sensor networks. In Das, S.R., Das, S.K., eds.: IWDC. Volume 2918 of LNCS., Springer (2003) 205–214
14. Hedetniemi, S., Hedetniemi, S., Jacobs, D., Srimani, P.: Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. Comput. Math. Appl. **46**(5–6) (2003) 805–811
15. Römer, K., Blum, P., Meier, L.: Time synchronization and calibration in wireless sensor networks. In Stojmenovic, I., ed.: Handbook of Sensor Networks: Algorithms and Architectures. John Wiley & Sons (2005) 199–237
16. ScatterWeb. `http://www.scatterweb.net` (2006)
17. Kulkarni, S.S., Arumugam, U.: Transformations for Write-All-with-Collision Model. In Papatriantafilou, M., Hunel, P., eds.: OPODIS. Volume 3144 of LNCS., Springer (2003) 184–197
18. Herman, T., Tixeuil, S.: A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks. In: ALGOSENSORS. Volume 3121 of LNCS., Springer (2004) 45–58
19. Frank, C., Römer, K.: Algorithms for generic role assignment in wireless sensor networks. In: $3^{rd}$ ACM Conf. on Embedded Networked Sensor Systems. (2005)
20. Kakugawa, H., Mizuno, M., Nesterenko, M.: Development of self-stabilizing distributed algorithms using transformation: case studies. In: $3^{rd}$ Workshop on Self-Stabilizing Systems. (1997) 16–30

## A Appendix

The proof of the following theorem is omitted due to space limitations.

**Theorem 3.** *Let $D$ be a set of real numbers strictly between $0$ and $1$ and $(\beta_i)$ an infinite sequence with $\beta_i \in D$. Furthermore, let $(b_i)$ be an infinite sequence with $b_i \in \{0,1\}$ and $\mathrm{Prob}(b_i = 1) = \beta_i$ for all $i \geq 0$. For $m \in \mathbb{N}$ let $(1)_m$ be the finite sequence $1, 1, \ldots, 1$, i. e., $1$ is repeated $m$ times. Let $P_m(k)$ be the probability that $(1)_m$ is not a subsequence of $b_1, b_2, \ldots, b_k$. If $D$ is a finite set or $\lim\limits_{i\to\infty} \beta_i > 0$ then $\lim\limits_{k\to\infty} P_m(k) = 0$ for all $m \in \mathbb{N}$.*

# The Case for Virtualized Wireless Access Networks

Frank A. Zdarsky, Ivan Martinovic, and Jens B. Schmitt

disco | Distributed Computer Systems Lab
University of Kaiserslautern, 67655 Kaiserslautern, Germany
{zdarsky, martinovic, jschmitt}@informatik.uni-kl.de

**Abstract.** Densely populated areas such as city centers are often sprinkled with numerous wireless access points operated for commercial or private use. Together they create what's known as *tragedy of the commons* phenomenon: without proper coordination, a significant amount of license-exempt radio frequency resources is wasted due to contention at shared medium access. In this paper we argue that both commercial and private operators would benefit if their access points were enabled to cooperate and form a single virtual access network that manages available radio resources itself in a globally optimal way. On top of this virtual network, each operator would then allocate resource shares for his own disposal. As a first step towards this vision we then present a distributed algorithm and protocol that allows previously unrelated access points to form and manage a single network in a self-organized manner and demonstrate its effectiveness.

## 1 Introduction and Motivation

The airspace is becoming crowded as increasingly many IEEE 802.11-based wireless access points are deployed in close proximity of each other. Many of these access points are part of small home or company networks that restrict access to few selected clients, while others belong to networks that try to reach city-wide coverage and offer Internet access to a wide audience. Such large-scale wireless access networks are often deployed for profit by wireless ISPs. However, wireless community networks (e.g. [1]), which offer free Internet access based on spare capacity donated by their members, are rapidly gaining in popularity. Furthermore, some cities are deploying so-called "municipal wi-fi" networks offering Internet access free of charge[2]. As a consequence of these parallel infrastructure deployments, "hotspot areas" may be covered by several networks at the same time. In fact, as observed in [3], areas with densities of more than 10 (and even up to 80!) overlapping access points are not uncommon in some major U.S. cities.

Considering that the number of non-overlapping channels available for IEEE 802.11 wireless LANs is very low and that this number will be reduced further by channel combining techniques intended to increase transmission speeds, it is not surprising that network performance in environments with high access point densities is severely degraded due to contention at shared medium access. As a result it becomes challenging at least to provide a certain service with a reliably high quality, which is unfortunate in particular in the light of such applications as Voice/Video over WLAN. While each provider may locally optimize his network, choosing the best installation sites and operation parameters for its access points, the lack of information about neighboring access points means that from a global perspective the available license-exempt frequency resources are not utilized to their full potential. This is especially true compared to a well-planned wireless network operating in licensed frequency bands.

Economists use the term “tragedy of the commons”[4] to denote a class of phenomena in which the use of a common good (here the license-exempt frequency resources) by utility-maximizing individuals leads to very low overall utilizations of the good. Their solution is to either restrict the use of the common good to selected individuals or to introduce binding cooperation mechanisms between individuals[5].

We contend that all operators of access points, commercial or private, would benefit from the introduction of cooperating mechanisms between their wireless LANs as well. In this paper we therefore introduce the concept of a *virtualized wireless access network*. The basic idea is to allow wireless access points from different physical networks and operators to form a single virtual access network that self-manages its available resources as efficiently as possible. Then, on top of this virtual access network, each contributor to the network may then create a logical access network that uses a share of the managed resources which is proportional to the amount of the original contribution. Logical networks should appear to wireless clients as independent physical networks and allow providers to offer the same set of services as in their own physical network and under the provider’s branding.

The virtualized wireless access network has advantages for both private users as well as commercial users:

– Wireless ISPs benefit from reducing their deployment and maintenance cost and may be able to serve customers in locations where they do not have own installation sites for access points or where these sites are not profitable. Most importantly, however, they do not have to concern themselves with radio management of their access network anymore and may offer a more reliable service to their customers, as contention from other wireless LANs is reduced.
– Private users benefit from lower contention from other LANs as well, but also from the possibility of free access in other locations of the city, where participating access points are available.

In the following section we discuss related work, before giving a more in-depth overview of technical aspects regarding virtual wireless access networks in section 3. We demonstrate in section 4 that cooperation between wireless LANs leads to a significant reduction in global network contention. Following this we introduce in section 5 a distributed algorithm and protocol that allows access points to form a virtual network and manage its resources in a distributed, self-organized manner and evaluate the algorithm in section 6, before concluding the paper with a brief outlook.

## 2 Related Work

Our proposed concept of a virtualized wireless access network is in fact the logical extension of the concept of a *virtual access point*[6] that has recently caught on with wireless ISPs, as it allows them to share access point deployment, and maintenance cost. The difference is that we envision a single, city-scale access network to which private households may contribute (and benefit from) just the same as commercial or public ISPs. This network should be self-forming and self-managing, as the nature of this network does not lend well to a centrally managed network.

One objective of a self-managing virtual wireless access network should be to utilize available resources efficiently by minimizing contention in the network. A number of contributions in the literature have already treated the general wireless LAN planning problem. These schemes are of little use in our context, as they assume that a very regular and optimal placement of access points is possible. Nevertheless we mention them for completeness: [7] have formulated an access point placement problem with the objective of minimizing contention, [8] a channel assignment problem. Joint placement and channel assignment schemes have been proposed, where co-channel overlapping may be allowed [9] or not [10]. In contrast to these contributions on the *planning* of wireless LANs, in [11] we proposed a model for the case where access point locations are already given and the problem is to determine the configuration of transmission power, channel assignment and associations of stations to access points that minimizes contention in the given network.

Online radio resource management schemes, on the other hand, are more closely related, as they assign operating channels, control transmit power, and perform load balancing of stations over access points online. Both centralized and decentralized schemes have been proposed in the literature (e.g. [12] and [13], respectively) and are used in existing products (e.g. [14] and [15], respectively). Their focus lies on controlling contention inside a single, closed, well-planned network domain, though. The problem of actively controlling inter-domain contention has received little attention in the literature. [16] suggest the use of a radio resource broker that tries to control contention between different wireless LANs by assigning to each LAN the set of channels and transmission powers that it may use. This centralized approach seems to be most appropriate for a small number of networks whose operators have reached agreements concerning the use of such a broker. Finally, [3] suggest algorithms that allow access points to adjust transmission power levels and rates automatically and independently from other access points.

## 3 Virtualized Wireless Access Networks

A *virtualized wireless access network* (VWAN), as used in this paper, is characterized by the fact that it consists of a set of wireless access points (APs) from different physical access networks (usually belonging to different administrative domains). These APs manage available resources themselves rather than requiring central management. They manage resources in a way that maximizes the *global* utilization of the network, and they allow logical subnetworks to be formed on top of the VWAN. In this section we describe the operation of a VWAN and discuss the most important design issues, some of which are still open at this point.

*Creation and Maintenance of a Virtualized Wireless Access Network.* Adding an access point to an existing VWAN should be an automatic process and should not require manual intervention of an operator. Therefore, an AP should on power-up perform a passive or active scan of the wireless medium for other APs that are already part of the VWAN. This is recognized by the presence of a specific Information Element (IE) contained in the periodically broadcasted beacon management frames of member access points. This IE also indicates the IP address at which the AP sending the beacon may be contacted by the new member. All further communication between the new AP and the old members is then sent via the wired backbone. During the life-time of the VWAN, APs are constantly monitoring whether a reconfiguration of the network with respect to channel assignment, transmit power settings, association of stations to access points, etc. would lead to a higher utilization of network resources. If so, this reconfiguration is performed automatically, without requiring intervention by the owners of the APs. In section 5 we propose an algorithm and protocol that enables such self-management of access points.

*Creation of Logical Subnetworks.* Every operator of an AP that is part of the VWAN may monitor the current state of the whole network, e.g. querying information about the number, locations and utilizations of other APs in the network. To create his own (logical) wireless access network, the operator may then select a subset of available physical APs and assign a resource share of each AP to the logical access network. This allows operators to serve customers in areas where they do not have infrastructure of their own. The amount of resources that an operator may reserve for his logical network should be a function of what he has contributed to the VWAN. How exactly the "exchange rate" is determined is still an open issue. However, it might be reasonable for operators to earn more credit for establishing access points in areas where only few access points already exist, spend more on resources of popular access points, and have a discount on using resources on own access points.

*Sharing of Access Points.* The sharing of a physical AP between different operators is enabled by a concept called *virtual access point* (VAP)[6]. A VAP is a logical entity that exists within a physical AP. When a single physical AP supports multiple VAPs, each VAP appears to stations to be an independent physical AP. A VAP is an emulation of a physical AP at the MAC layer. This has the advantage that no additional radio hardware is needed to implement VAPs. On the other hand this means that all VAPs of a single physical AP operate on the same channel, which is not a real restriction in this context, though.

The most common way of implementing VAPs is to assign each VAP its own BSSID (the IEEE 802 48-bit MAC address of the AP) and to let each VAP broadcast its own beacon frames. As beacons contain the SSID (the 32 byte network identifier string) and the capability information for each network, it is possible for providers to use their own branding for their VAPs and offer different authentication mechanisms, transmission rates, etc., despite sharing a single physical AP. A disadvantage of sending one beacon frame per VAP is the increased bandwidth overhead, which means that this approach does not scale abritrarily well. However, as nicely argued in [6], other approaches of using only a single BSSID and single or multiple SSIDs per beacon do not offer the same functionality or are not compatible with legacy stations. Furthermore, the same scalability issues would arise when using different physical APs. A solution may be to aggregate some logical wireless access networks under the same VAP, which should at least be possible for those operators that are not commercial ISPs, but maybe members of a wireless community network.

*Sharing of Internet Uplinks.* As argued in the introduction, we see benefits in allowing home users to participate in the VWAN as well. This raises questions on whether their uplink to the Internet is sufficient for this type of application. However, this problem is analogous to that in wireless community networks. Some of the more sophisticated open-source AP firmwares that specialize on wireless community networks (e.g. DD-WRT [17]) are already using techniques which may to some extent also be applied in this context.

A first issue is the available bandwidth in the wired uplink to the Internet. Using QoS mechanisms it is not difficult to isolate the traffic of different VAPs and assign them the share of resources that the VAP's operator has paid for. But is the capacity of the Internet uplink sufficient to make it worthwhile for a commercial wireless ISP (WISP) to use APs of home users? At the moment, many households have Internet uplinks that are much slower than an IEEE802.11a wireless access link. However, considering the increasing availability of high-speed Internet access, such as ADSL2+, in private households, due to the convergence of Internet, Television, and Telephony (so-called "triple-play"), the gap between wireless and wired link capacities in home networks may become much narrower in the future. Furthermore, in wireless community networks using mesh-routing, but also in so-called mushroom networks[18], it is not uncommon to route traffic to the Internet via wireless links to multiple wired uplinks, thereby increasing the available uplink capacity.

Secondly, there is the problem that home users are usually assigned a single IP address by their ISP, which is furthermore often only dynamically assigned. Again, wireless community networks show that with NAT and port forwarding this need not be an issue. When IPv6, and in particular MobileIPv6, continue to gather momentum, this problem may also be mitigated.

Until then our approach is to set up static tunnels from each VAP in a foreign network to the respective provider's home network and use GRE to tunnel data between the two. This has the advantage that to both provider and its customers this process is transparent: they will not notice that they are in a different network, other than an increase in round-trip-time.

*Security Issues.* There are several security issues that have to be solved to make VWANs fully usable. For example, can a WISP trust a home user (and vice versa) not to tamper with the wireless AP in order to eavesdrop on connections or tamper with accounting? How can it be monitored that a VAP actually provides the assured bandwidth share? We cannot yet offer satisfactory solutions to these problems and leave them for future work.

## 4 Effect of Cooperation on Contention

In this section we describe an experiment that exemplifies the significant reductions in network contention that may be gained through cooperation between operators of wireless LANs for different access point densities. This evaluation is based on the mathematical optimization framework that we proposed in [11]. It allows to determine the channel assignment, transmit power setting and station association that minimizes the contention level in a network. The contention level counts how many nodes actually interfere with a given node's transmission, summed over all nodes. We refer the reader to our previous work for further details.

Our reference scenario contains 25 access points (APs) arranged on a 5x5 grid. The distance between grid lines is chosen as twice an AP's transmission range, in other words the distance at which a node can just receive the AP's signal at its minimum required power level. We then generate two stations per AP, one located somewhere at the maximum transmission range, the other located between the AP and the maximum transmission range, with uniform distribution.

Path losses between each pair of nodes are calculated based on the empirical indoor propagation loss model recommended in ITU-R P.1238-2 [19]. The maximum transmission power of each node is assumed to be 20dBm (or 100mW), which is the maximum power allowed for IEEE 802.11b wireless LANs in Europe. We assume that a node detects a busy medium when interfering signals are stronger than -84dBm and that it requires a minimum signal strength of -82dBm to successfully decode a signal. These are typical values for an Orinoco Gold IEEE 802.11b adapter. The number of non-overlapping channels is assumed to be 3.

The APs of the reference scenario described above are then squeezed towards the center or stretched away from it to yield new scenarios of different AP densities. This technique is analogous to the one used in [3]. We create scenarios with stretch factors between 0.0 and 1.5, where a stretch factor of 1.0 denotes the reference scenario, and a stretch factor of 0.0 the situation where all APs are squeezed together at the center.

For all scenarios we use our optimization framework to determine the minimal possible contention level that may be achieved by standard non-cooperating wireless LANs and compare them to those that may be achieved by wireless LANs which cooperate in reducing contention. In the non-cooperative case, access points select their operating channel independently from their neighbors by scanning for free channels and then choosing a free channel or, if no free channel is available, one of the busy channels randomly. The results are shown in Fig.1, which for reference also includes a theoretical lower bound (TLB) that denotes the lowest contention level for any set of networks with the given number of access points and stations.

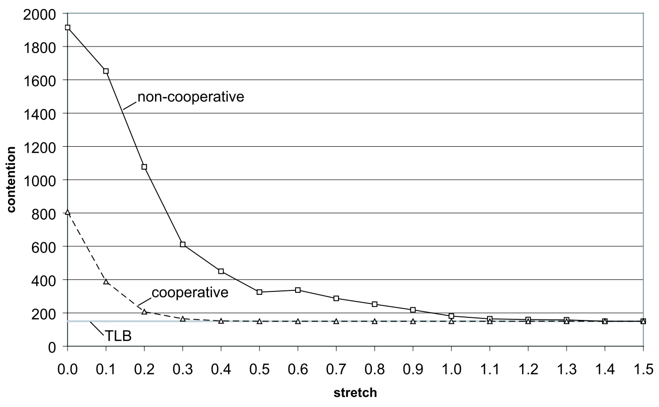


**Fig. 1.** Contention levels with and without cooperation for varying access point densities

The results show that in very sparse wireless LAN scenarios, non-cooperative wireless LANs achieve the same or only slightly worse contention values than cooperative LANs. As the density increases (that is the stretch factor decreases), contention in the non-cooperative approach increases rapidly, as more and more co-channel overlapping occurs. The slight decrease at a stretch of 0.5 can be explained by the fact that starting from this stretch APs are in direct transmission range of each other and therefore try to change to a free channel if possible. In contrast, cooperation between wireless LANs, that is coordinating the use of operating channels and station associations, reduces contention significantly (for example already by a factor of 2 at stretch 0.5) even when AP densities are high.

Note that contention occurring at stretch factors $>1.0$ is due to the fact that while nodes at this distance are outside their transmission range as specified above, the strength of the received signal may still be greater than the signal strength at which the Clear Channel Assessment function of a node reports a busy channel.

## 5 Distributed Coordination Algorithm

In this section we propose a distributed algorithm that allows neighboring access points to cooperatively reduce contention, based solely on joint knowledge about their vicinity. It consists of five modular building blocks:

- *Data dissemination*, in which an access point (AP) discovers other APs within its horizon and collects information about the stations (STAs) that each of these APs is aware of and is able to cover at the required signal strength.
- *Local negotiation*, in which an AP suggests a local reconfiguration of the network to all APs within its horizon, waits for their feedback on how this reconfiguration would affect network performance in their vicinity and then decides either to commit or abandon this reconfiguration.
- A *fitness function* with which to evaluate the current state of the network within an APs horizon and the effect of a proposed reconfiguration.
- A *local reconfiguration algorithm* that is used to find better local reconfigurations.
- A *coordination mechanism* to determine, which APs are allowed to propose local reconfigurations and when.

An AP's *horizon* defines which other APs and STAs in its geographical vicinity it knows and cooperates with in finding improvements. When choosing the extent of the horizon, one has to make the typical trade-off between the chances for finding the globally optimal configuration and the computational effort and signaling overhead. In our experiments we have defined the horizon of an AP $i$ as the set of all APs whose transmissions AP $i$ can receive directly or can infer from listening to stations from other APs in range.

### 5.1 Data Dissemination

APs initially find out about their neighbors by scanning for periodic beacon signals on all available channels. Upon receiving a beacon from a previously unknown neighbor, the AP sends out a WELCOME message to its new neighbor, both on the wireless link and on the wired backbone network. This assumes that the IP address of the new neighbor is known. The most simple solution is to let each AP include its IP address as an additional Management Frame Information Element in its broadcasted beacons.

Both the WELCOME message and the reply to it (WELCOME_ACK) contain information about the sending AP as well as about all STAs which the sending AP is currently aware of and whose minimum signal strength requirements it can meet. By sending these messages over both the wireless link and the backbone, we can further gain information about whether the wireless link is asymmetric or not, that is if one access point is able to hear the other but not vice versa.

Furthermore, all active APs periodically send UPDATE messages to all APs within their horizon containing their current STA information list. This information has an explicit expiration time, so if an AP does not receive UPDATE messages from a neighbor for a certain duration, it assumes the neighbor has deactivated without signing off. UPDATE messages are always sent via the wired backbone, so that this soft-state approach does not consume valuable wireless resources.

We also consider the case that two APs that cannot hear each other directly nevertheless produce contention in each other's BSS. This may happen when an STA is located in between the AP it is associated to and another AP that is within contention range. The STA may then notify its own AP of the contending AP's presence so that both APs may contact each other using the mechanism described above.

## 5.2 Local Negotiation

Based on its knowledge about APs and STAs within its horizon, an AP may run a local optimization algorithm to search for better configurations for itself and its neighboring APs. If an AP finds a configuration that improves contention within its own horizon, it suggests the new configuration to its neighbors by sending them an OFFER message with the new configuration.

Upon receiving an OFFER, every neighbor determines the effect of the configuration change on their part of the network. Note that the sets of nodes within the horizons of the APs sending the OFFER and receiving the OFFER is usually not identical, although the intersection should usually be large. All receivers of an OFFER then answer with an OFFER_REPLY message containing the predicted change in contention that would result from actually committing the configuration change. If the net effect of the reconfiguration proposal is positive, the initiating AP sends a COMMIT message to all neighbors, who then update the local knowledge about their neighborhood and possibly change the radio channel they operate on or instruct individual STAs to reassociate with a different AP.

There are three cases in which the initiating AP sends a WITHDRAW message to its neighbors in order to cancel a reconfiguration attempt. The first case is that the initiator calculates a negative or zero net effect of the reconfiguration proposal. Secondly, it may happen that one of the receivers of an OFFER message is already processing a reconfiguration proposal by a different AP which has not been committed or rejected yet. It then refuses the new OFFER by answering with a BUSY message. Finally, if at least one of the neighbors does not respond to the OFFER within a certain time interval, the initiator assumes the message was lost or the receiver has deactivated.

## 5.3 Reconfiguration Algorithms

In order to find a reconfiguration that yields a lower amount of contention, an AP applies an optimization algorithm to the set of APs and STAs within its horizon, including itself. We have experimented with two optimization algorithms: a problem-specific genetic algorithm[11] and with a greedy heuristic which we termed “balance or conquer”.

This heuristic is inspired from previous findings that balancing of STAs between APs, where possible, leads to low contention values if there is no contention between different BSSes. In the presence of inter-domain contention, however, load balancing may actually be detrimental to reducing contention.

The “balance or conquer” heuristic owes its name to its repertoire of four strategies for improving contention:

1. Try to transfer STAs to (from) other APs such that the number of STAs per channel (not per AP!) is roughly the same within the horizon (= balance). Change your own channel, if necessary.
2. Find another AP whose stations you can cover completely and take them all (= conquer), effectively switching the other AP off.
3. Try transferring all stations to other APs, balancing the number of STAs per channel, effectively switching yourself off.
4. If currently switched off, try to incrementally take over STAs (starting with the nearest one) from other APs, as long as this does not increase contention. Change your channel, if necessary.

During a single run of the heuristic, an AP instantiates the optimization model with the knowledge it has collected. It then computes the change in contention that would result from applying each of the four strategies and then greedily picks the one with the highest presumed benefit.

## 5.4 Coordination of Reconfigurations

The last building block of our algorithm is concerned with the question *when* APs attempt to find and propose an improved configuration. We have used both an uncoordinated approach, in which each AP performs reconfiguration attempts as a Poisson process. Furthermore, we have used two token-passing algorithms, where an AP currently holding a token waits for a random time interval before attempting to propose a reconfiguration. Whether this proposition was successful or not, it then passes the token on to a randomly chosen neighboring AP. The two token-based approaches differ in that the first approach starts with a single token that circulates the network, while in the second all APs initially hold a token. When an AP receives a new token from a neighbor while already holding one, the new token is destroyed, so that eventually only one token remains in the network. Lost or destroyed tokens could be replaced by letting each AP generate a new token at a very small rate, which could vary with the amount of contention—and therefore the necessity for a new token—within an AP’s horizon.

The rationale behind experimenting with different reconfiguration coordination approaches is that one can expect the global level of contention in the system to decrease more rapidly when a high number of access points concurrently tries to find and propose reconfigurations, as is the case with the uncoordinated approach. On the other hand, when reconfigurations are made at different locations of the network at the same time, there is a chance that the effect of one reconfiguration is counterproductive with respect to another reconfiguration in the long run.

## 6 Experiments and Results

### 6.1 Performance of the Distributed Algorithm

In this section we conduct simulations to study how our distributed algorithm compares both to standard WLAN and the optimal solution in a cooperative scenario.

We use 10 different scenarios, each with 50 APs and 100 STAs within a 1km by 1km simulation area. A scenario is generated as follows: In a first step, 16 of the APs are placed to regularly cover the simulation area. Afterwards, the remaining APs are placed uniformly over the simulation area. The location of each STA is chosen by picking an AP randomly and then placing the STA within a distance of 10% to 90% of the transmission range of the AP, drawn from a uniform distribution. Node transmission and reception powers as well as the path losses are chosen as in the previous section.

As reference solution for each scenario we use the behavior of typical wireless LAN, but under the cooperation assumption. That is, STAs associate with the AP from which they receive the strongest signal, irrespective of the administrative domain the AP belongs to. Furthermore, all APs choose an unused channel or pick one randomly if all channels are already occupied. This reference solution also serves as the starting point for our distributed algorithm. To estimate the optimal configuration, we use a run over 100,000 iterations of our genetic algorithm, equivalent to roughly an hour's worth of computation on a standard PC.

We perform simulations both using the genetic algorithm (GA) and the balance-or-conquer (B|C) as local reconfiguration heuristics. In order to study the effect of concurrent reconfigurations versus sequential reconfigurations, we further use three different reconfiguration coordination approaches with both algorithms: Uncoordinated reconfiguration (0 tokens), token-passing with 1 token and $N$ initial tokens, where $N{=}50$ (the number of access points). If no tokens are passed in the network, the generation of reconfiguration attempts per AP is a Poisson process with rate 1/s. If one or more tokens are present, the holding time of a token is exponentially distributed with mean 1s. Each simulation instance runs for one hour of simulation time and is repeated for each of the ten scenarios.

The resulting average contention values (both absolute and relative decrease compared to WLAN) and their standard errors are shown in Table 1.

Figure 2 additionally shows the development of the contention level over time for one of the simulated scenarios. As the global GA is only (albeit a very good) heuristic, it does not necessarily find the global minimum. As the optimization

**Table 1.** Comparison of contention levels achieved by the distributed algorithm using GA and B|C

| | WLAN | GA | Local GA | | | Local B\|C | | |
|---|---|---|---|---|---|---|---|---|
| initial tokens | | | 0 | 1 | $N$ | 0 | 1 | $N$ |
| mean | 512.5 (0.0%) | 374.5 (-26.4%) | 409.8 (-19.8%) | 411.5 (-19.3%) | 413.5 (-18.9%) | 454.0 (-11.1%) | 416.8 (-18.2%) | 425.3 (-16.6%) |
| std. error | 18.8 | 7.2 | 13.6 | 11.9 | 10.9 | 15.9 | 10.6 | 11.6 |

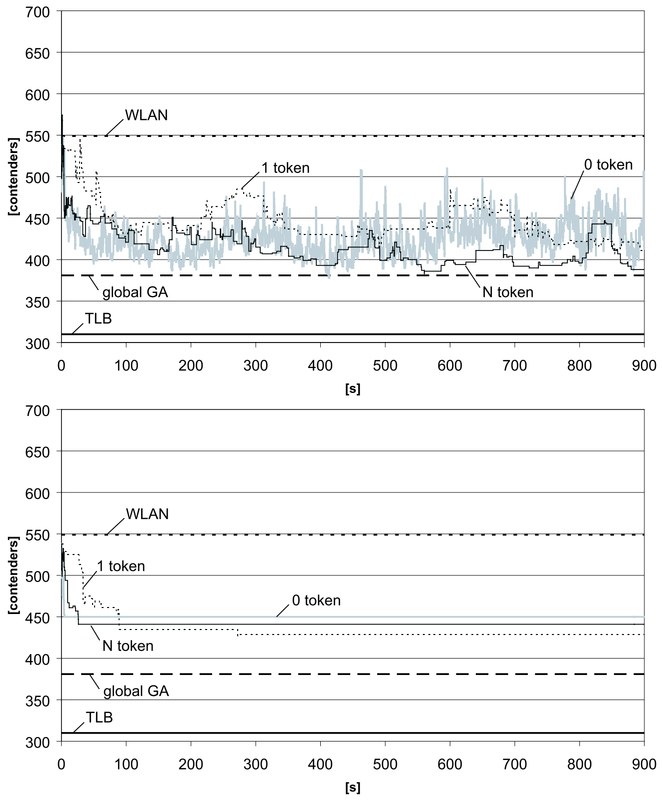


**Fig. 2.** Performance of GA (top) and B|C (bottom) as local reconfiguration algorithms compared to global minimum and WLAN

problem is far too complex to exactly determine the true minimum, we have additionally included the theoretical lower bound (TLB) which we derived in [11].

In our simulations, the GA version of our distributed algorithm manages to realize on average 65.5% of the improvement potential compared to WLAN, the B|C version 61.8%, both for the 1 token case. This corresponds to a decrease in network-wide contention by 19.3% and 18.2%, respectively. We note that both versions switch off a significant number of APs to achieve this result (12.2% and 13.6%, respectively), rather than balancing STAs across available APs.

Although both versions achieve comparable results, this does not mean that both versions are equally suitable for real-world application. The computational effort per search for a better local reconfiguration is on the order of two magnitudes higher for the genetic algorithm than for B|C, while only achieving slightly better results. Furthermore, the stability of the contention levels is not the same between the two versions as can be directly seen from Fig.2 as well.

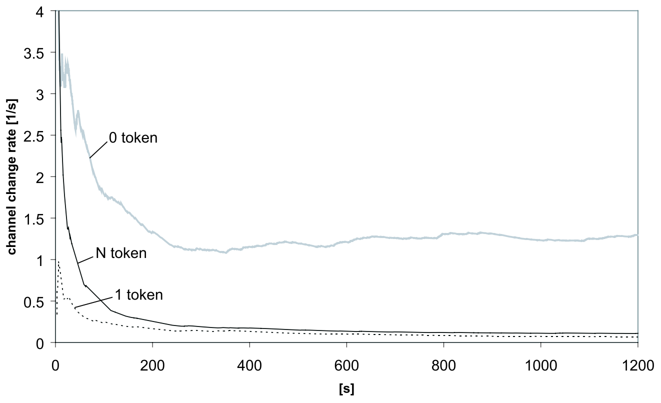


**Fig. 3.** Channel change rate of GA as local reconfiguration algorithm

We also observe that the choice of the reconfiguration coordination mechanism has a strong effect on the speed of the improvements in contention, but also on the quality of the attained contention level. Using no coordination between reconfiguration attempts of different APs leads to very quick improvements compared to the 1 token approach. Interestingly, though, in almost all cases the B|C heuristic is able to converge to lower contention levels the slower the rate of reconfigurations. The *N* token case is usually somewhere in between, reacting as the uncoordinated case when a large number of tokens is still present. Over time it converges to the behavior of the 1 token case, as more and more tokens are destroyed. Figure 3 shows the channel changes per second (as a total over the whole network) for the local GA algorithm and the 0, 1, and N token cases, which again supports the aforementioned observations.

## 6.2 Importance of Coordination

Finally, we would like to find out how important the local negotiation part is for our distributed algorithm. We therefore conduct a set of experiments in which we remove the negotiation process, so that an AP finding a better configuration immediately commits the necessary changes instead of sending offers to all other APs within its horizon asking for feedback. The results of one of the scenarios are shown in Fig.4. Indeed, when an AP does not ask its neighbors for potential

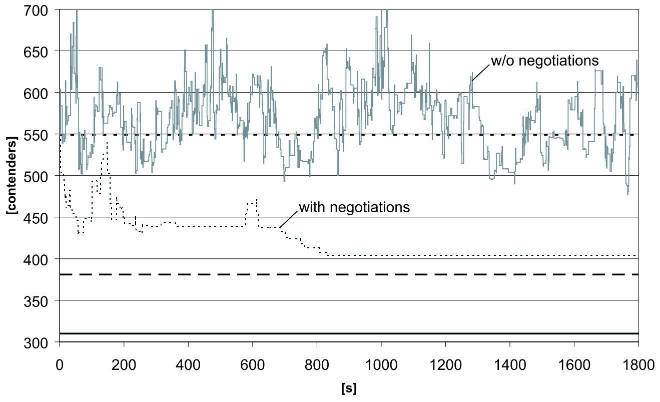


**Fig. 4.** Comparison of algorithm performance with and without negotiations

negative effects of a configuration change, it frequently happens that this AP reconfigures to gain a small improvement, but that this reconfiguration has strong negative effects on the network just outside its horizon. Affected APs may in turn attempt to improve their situation, possibly undoing the original changes. As a consequence, contention levels fluctuate heavily and may on the average even be higher than with plain WLAN.

## 7 Conclusions

In this paper we have introduced the concept of a virtualized wireless access network, which consists of wireless access points from many different operators, including those of both commercial WISPs and private households. Virtual wireless access networks are self-forming and self-managing, with the objective of minimizing contention between the participating access points and stations and of using the scarce license-exempt frequency resources as efficiently as possible. On top of this resource-efficient network, its various contributors may then create logical networks on which they may offer services under their own brand. We have argued why participation in a virtualized wireless access network may be beneficial for both commercial and private contributors. Furthermore, we have proposed a distributed algorithm and protocol allowing access points to cooperatively manage radio resources and have shown its effectiveness.

Currently we are working on a proof-of-concept based on a set of set of LinkSys WRT54G routers and the DD-WRT open source embedded Linux system, which already contains many of the required features.

## References

1. NYCwireless (2006) `http://www.nycwireless.net.`
2. Singer, M.: 'Wireless Philadelphia' Sparks Concern (2004) `http://www.internetnews.com/wireless/article.php/3442851` (last access: 2006-02-01).

3. Akella, A., Judd, G., Seshan, S., Steenkiste, P.: Self-Management in Chaotic Wireless Deployments. In: 11th International Conference on Mobile Computing and Networking (MOBICOM '05), Cologne, Germany (2005)
4. Hardin, G.: The Tragedy of the Commons. Science **162**(3859) (1968) 1243–1248
5. Ostrom, E.: Coping with Tragedies of the Commons. Annual Review of Political Science **2** (1999) 493–535
6. Aboba, B.: Virtual Access Points. 802.11 wg document, IEEE (2003) `http://www.drizzle.com/ãboba/IEEE/11-03-154r1-I-Virtual-Access-Points.doc` (last access: 2006-02-01).
7. Amaldi, E., Capone, A., Cesana, M., Malucelli, F.: Optimizing WLAN Radio Coverage. In: IEEE International Conference on Communications (ICC 2004), Paris, France (2004) 180–184
8. Leung, K., Kim, B.J.: Frequency Assignment for IEEE 802.11 Wireless Networks. In: 58th IEEE Vehicular Technology Conference (VTC 2003 Fall), IEEE (2003) 1422–1426
9. Ling, X., Yeung, K.: Joint Access Point Placement and Channel Assignment for 802.11 Wireless LANs. In: IEEE Wireless Communications and Networking Conference (WCNC 2005). (2005)
10. Lee, Y., Kim, K., Choi, Y.: Optimization of AP Placement and Channel Assignment in Wireless LANs. In: IEEE Conference on Local Computer Networks (LCN 2002). (2002)
11. Zdarsky, F.A., Martinovic, I., Schmitt, J.B.: On Lower Bounds for MAC Layer Contention in CSMA/CA-Based Wireless Networks. In: 3rd ACM/SIGMOBILE International Workshop on Foundations of Mobile Computing (DIALM-POMC'05), Cologne, Germany (2005) 8–16
12. Hills, A., Friday, B.: Radio Resource Management in Wireless LANs. IEEE Communications Magazine **42**(10) (2004) 9–14
13. Wang, Y., Cuthbert, L., Bigham, J.: Intelligent Radio Resource Management for IEEE 802.11 WLAN. In: IEEE Wireless Communications and Networking Conference (WCNC 2004), Atlanta, Gergia USA (2004) 1365–1370
14. WS5100 Wireless Switch Reviewer's Guide. Product brochure, Symbol Technologies (2005) `ftp://symstore.longisland.com/Symstore/pdf/wireless/WS5100ReviewersGuide.pdf` (last access: 2006-02-01).
15. AutoCell—The Self-Organizing WLAN. White paper, Propagate Networks (2003) `http://www.propagatenet.com/resources/docs/whitepaper_autocell.pdf` (last access:2006-02-01).
16. Matsunaga, Y., Katz, R.: Inter-Domain Radio Resource Management for Wireless LANs. In: IEEE Wireless Communications and Networking Conference (WCNC 2004), Atlanta, Georgia, USA (2004) 2183–2188
17. DD-WRT (2006) `http://www.dd-wrt.org`.
18. Mushroom Networks (2006) `http://www.mushroomnetworks.com`.
19. ITU-R P.1238-2: Propagation data and prediction methods for the planning of radio communication systems and radio local area networks in the frequency range of 900 MHz to 100 GHz (2001)

# Job Scheduling for Maximal Throughput in Autonomic Computing Systems

Kevin Ross[1] and Nicholas Bambos[2]

[1] UCSC School of Engineering
kross@soe.ucsc.edu
[2] Stanford University School of Engineering
bambos@stanford.edu

**Abstract.** Autonomic computing networks manage multiple tasks over a distributed network of resources. In this paper, we view an autonomic computing system as a network of queues, where classes of jobs/tasks are stored awaiting execution. At each point in time, local resources are allocated according to the backlog of waiting jobs. Service modes are selected corresponding to feasible configurations of computing (processors, CPU cycles, etc.), communication (slots, channels, etc.) and storage resources (shared buffers, memory places, etc.)

We present a family of distributed algorithms which maximize the system throughput by dynamically choosing service modes in response to observed buffer backlogs. This class of policies, called projective cone scheduling algorithms, are related to maximum pressure policies in constrained queueing networks, and are shown to maintain stability under any arrival combination within the network capacity. They operate without knowledge of the arrival rates and require minimal information sharing between regions.

## 1 Introduction

Autonomic computing systems and networks manage several processes simultaneously via resource sharing, which leads to multiplexing gains and economies of scale/scope. However, it also leads to highly complex resource scheduling issues, some of which we address in this paper. Specifically, we consider real-time scheduling of computation jobs/tasks that share infrastructure resources. Those typically include processors or CPUs cycles, communication and/or switching (micro)channels and slots, memory places, disk blocks, etc. Jobs arrive requiring different sets of resources,which must be simultaneously allocated to the various waiting tasks for their execution.

At an appropriate level of modeling abstraction, one can view a computing system as having various queues, where classes of jobs/tasks are queued up awaiting execution. At any point in time, each region of the network can be set to one of several available service modes; each queue then receives service at a mode-dependent rate. Modes correspond to feasible configurations of diverse processing resources, respecting various compatibility and synchronization constraints. This paper develops a framework for dynamically configuring the infrastructure resources of an autonomic network so as to maximize the system throughput and load balance the various queues appropriately.

While the service within each stage or region is independent in terms of service rates, the decisions in each region influence other regions by forwarding completed or partially completed jobs. Therefore a coordinated allocation policy needs to be implemented in order to maximize the throughput in the network. Ideally, one global scheduler would allocate all resources by considering the network effects, but in practice this cannot be coordinated due to the complexity and scale of these systems. We present distributed (i.e. regional) scheduling/routing algorithms that guarantee globally maximal throughput for general processing networks. The networks considered can be operationally partitioned into distinct network regions, where a set of infrastructure resources are assigned/dedicated to each region. Within each region, a local scheduler allocates the available resources, completing jobs and forwarding them to other regions or out of the network. Each resource allocation option corresponds to a set of service rates to the various queues. Since computer resources are very flexible, these allocations need to encapsulate general combinations of service.

We find that the maximum possible throughput can be guaranteed throughout the network, with minimal information sharing between different network regions. In particular, each region needs only to know the backlog its own queues and any queues it is forwarding into. The intuition behind the algorithms presented is that they push the maximum number of jobs from large buffers to small.

Autonomic computing systems in data centers, grid computing networks, utility computing environments, etc. have become important technologies in recent years [6,13,16,20,23]. However, on-line job scheduling issues and throughput limits in such large-scale systems are not well understood. Some performance studies have led to interesting scheduling research [5,8,11,12,21,22]. For the special case where completed jobs always immediately exit the system, regional resource allocation is similar to that in a generalized switch. In a switch, cross-connects correspond to virtual output queues undergoing service. In that context, throughput has been widely studied, see for example [2,4,9,14,17]. However, autonomic networks have downstream effects, where completed jobs at one server forward jobs to another queue. The potential for such localized switching algorithms to lead to network instability was shown in [1].

The algorithms we present are based on maximum weighted-matching policies, first proposed by Tassiulas and Ephrimedes [19]. We extend these and similar results [3,7,10,18] by considering the distributed allocation of resources. Several techniques have been applied for throughput analysis, for example, [3] investigates 'maximum pressure policies' by using fluid models rather than Lyapunov techniques. We build upon the analysis methodology proposed in [2] and [14] that analyzed switches (single-pass queueing networks) where, upon service, fragments immediately depart the network. We extend substantially the model and analysis framework in [14].

Our focus in this paper in on the *distributed decision making* in such networks and our emphasis is on *autonomous regional algorithms*, which have not been explored before. They promise to be useful and scale nicely to realistic large networks, because of their distributed nature. The results here show the fundamental performance limits of multi-stage, multi-class processor networks, and how to operate the networks in a distributed way when resources are constraining the performance. While the contribution is primarily of a theoretical nature, the algorithms described show a lot of promise for scalable implementation. Due to space limitations we focus on a thorough analysis of the distributed throughput result and leave the treatment of other issues to future work.

## 2 The Network Model and Its Dynamics

We consider an autonomic computing network as a processing system comprised of $Q$ interconnected first-in-first-out (FIFO) queues of infinite buffer capacity, indexed by $q \in \mathcal{Q} = \{1, 2, ..., Q\}$. The network is divided into *regions*, indexed by $r \in \{1, 2, ..R\}$, as illustrated in Figure 1. Within each region is a set of queues $\mathcal{Q}_r$ that can be served by the resources in region $r$, and another set of queues $\mathcal{F}_r$ which may receive jobs forwarded from queues in $\mathcal{Q}_r$. The regions $\mathcal{Q}_r$ are disjoint[1] but connected in that the queues in $\mathcal{F}_r$ may be part of the same or another region.

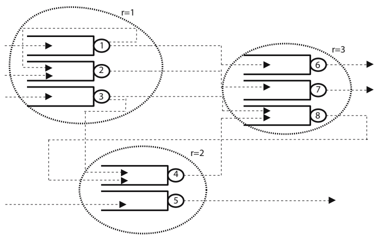


**Fig. 1. Network of Queues in Regions.** The network is divided into regions, with several resources available in each region. Allocating resources corresponds to serving the queues in the region at various rates. Fragments can be forwarded between queues in different regions, with merging, splitting and feedback encapsulated in this framework. In this example, $\mathcal{Q}_1 = \{1, 2, 3\}$ is the set of queues in region 1, and $\mathcal{F}_1 = \{2, 4, 6, 7, 8\}$ is the set of queues that can receive jobs forwarded from $\mathcal{Q}_1$.

Time is slotted and indexed by $t \in \{0, 1, 2, 3, ...\}$. In each timeslot, traffic load entering a queue may come from outside the network or be forwarded from an upstream queue. We use the term *fragment* to denote a unit of job service load in each queue. For simplicity, we assume that each job can be 'broken up' arbitrarily into fragments (or groups of fragments). Vectors are used to encode the network backlog state, arrivals, and service in each time slot. Specifically, $X_q(t)$ is the number of fragments in queue $q \in \mathcal{Q}$ at time $t$ and the overall backlog state of the network is $X(t) = (X_1(t), X_2(t), ..., X_q(t), ..., X_Q(t))$.

Requests to the autonomic network are recognized in terms of job fragments arriving to individual queues. The vector of external arrivals to the network is $A(t) = (A_1(t), A_2(t), ..., A_q(t), ..., A_Q(t))$, where $A_q(t)$ is the number of fragments arriving to queue $q$ at time $t$ from outside the network (as opposed to being forwarded from other queues). The following is assumed for each $q \in \mathcal{Q}$

$$\lim_{t \to \infty} \frac{\sum_{s=0}^{t-1} A_q(s)}{t} = \rho_q \in [0, \infty) \tag{1}$$

that is, the long-term average rate of external load arriving to each queue is well-defined, non-negative and finite. The long-term average load vector is $\rho = (\rho_1, \rho_2, ..., \rho_q, ...,$

[1] That is, $\mathcal{Q}_{r_1} \cap \mathcal{Q}_{r_2} = \emptyset$, for $r_1 \neq r_2$, and $\cup_r \mathcal{Q}_r = \mathcal{Q}$.

$\rho_Q$), but we do not assume any particular statistics that may generate the traffic traces. For example, arrivals can be correlated between queues or over time, and arrivals may occur to any part of the network. In each timeslot, the number of arriving fragments $A(t)$ satisfies $A_q(t) \leq A_q^{\max}$ for some finite value $A_q^{\max}$ for each $q \in \mathcal{Q}$. When a fragment enters a region, it is stored in its entry FIFO queue until service is applied to it. Upon service it can either be forwarded to another queue in $\mathcal{F}_r$, or it can depart the network.

Each region has a set of resources available, which can actually be *any* set, but for intuition one can think of these as some number of CPU's. These resources can be allocated to the queues in the region via many combinations. For example, a single CPU could be assigned to any one of the queues, or could perhaps be shared over more than one queue. Each potential allocation is called a mode, and we denote $\mathcal{M}_r$ to be all of the modes available in region $r$. Since it is never advantageous to allocate resources to empty queues, we restrict the set of modes available to be those which serve at most the number of job units waiting in each queue.

Each mode $m_r \in \mathcal{M}_r$ has a corresponding departure vector $D^{m_r}$, and an entering (feed) vector $E^{m_r}$. These represent the number of fragments departing and entering each queue under mode $m_r$. For notational simplicity we consider $Q$-length vectors for $D^{m_r}$ and $E^{m_r}$, with $D_q^{m_r} = 0$ for $q \notin \mathcal{Q}_r$ and $E_q^{m_r} = 0$ for $q \notin \mathcal{F}_r$.

These terms can be easily seen by considering the network in Figure 1. One of the modes from region 1 could be the mode that forwards a single fragment from queue 1 to 2 and 4 fragments from queue 2 to queue 8. This would be represented by the vectors $D^{m_1} = (1, 4, 0, 0, 0, 0, 0, 0), E^{m_1} = (0, 1, 0, 0, 0, 0, 0, 4)$.

While we use the language of *forwarding* between queues, the relationship between $D_q^{m_r}$ and $E_q^{m_r}$ can actually be *arbitrary*. This model allows for service at various queues to *create* backlog *anywhere* in the network. Most previous work considers the important special case where individual fragments are forwarded through a fixed network path. However, this model allows for the cases where $(i)$ fragments from one queue become multiple fragments in multiple downstream queues, and $(ii)$ fragments from one queue creates backlog in a physically separate parts of the network. These allow the coordination of various network tasks, for example job completions may trigger backup or security operations elsewhere in the network.

At each timeslot and in each region, a mode $m_r(t) \in \mathcal{M}_r(X(t))$ is selected from the available modes. The set of available modes is assumed to be such that there are enough fragments in each queue to complete all transfers. We make the following three assertions on the available modes: $(i)$ the set $m_r(X)$ of modes available in region $r$ for a particular backlog vector $X$ is all of the available modes that apply service at no more than the available backlog levels, that is $\mathcal{M}_r(X) = \{m_r \in \mathcal{M}_r : D_q^{m_r} \leq X, q \in \mathcal{Q}_r\}$, $(ii)$ given one feasible mode $m_r$, any individual fragment transfer can be canceled and this leads to another feasible mode, and $(iii)$ canceling departures to one queue cannot cause *more* arrivals to another. If we are given a set of modes that does not satisfy these conditions, one can complete the set of available modes by adding the modes corresponding to canceled service.

Each queue receives service in only one region, but fragments may be forwarded to it from several regions. Hence, the total departure rate at a given queue is the total service provided to it by the regional mode, while the effective arrival rate includes the rate of external arrivals and the rate of forwarding from upstream queues. In this work we focus on local mode selection algorithms, where each region independently selects its service mode, corresponding to allocating processors to jobs.

While mode selection is regional/local, the net effect of all local decisions is a global mode enabled at each timeslot. The mode *global* $m(t)$ selected at time $t$ is comprised of all the regional modes. Since mode selections are potentially independent, the set $\mathcal{M}$ of all possible network modes is $\mathcal{M} = \{m_1, m_2, ..., m_R : m_r \in \mathcal{M}_r, r = 1, 2, .., R\}$. A mode $m = \{m_1, m_2, ..., m_R\}$ selected from $\mathcal{M}$ is a specific set of regional modes. That is, the mode $m(t)$ at time $t$ is the set of all regional mode selections at time $t$.

Consider the backlog change in an individual queue $q$, due to the service mode in its region and any upstream regions. Note that $D_q^m = \sum_{r=1}^R D_q^{m_r}$ are the departures from queue $q$ under mode $m$. Since service to a single queue can only come from one region, only one $D_q^{m_r}$ is non-zero for each $q$. Similarly, $E_q^m = \sum_{r=1}^R E_q^{m_r}$ is the total number of fragments entering queue $q$ from all regions under mode $m$. This can include several nonzero terms, since any region may forward fragments to any other region.

For simplicity in notation, we use the vector form of all equations, omitting the subscript $q$ except where necessary. Since the quantity of interest is the effective backlog change under the selected mode, we define $S^m = D^m - E^m$ to be the backlog *change vector* at time when the system is serviced under mode $m$. This vector is the backlog change (positive or negative) applied to each queue in each timeslot.

Having defined carefully the terms in the backlog evolution, the vectors representing backlog and backlog change follow the simple evolution equation:

$$X(t+1) = X(t) + A(t) - S(t) \tag{2}$$

where $S(t) = S^{m(t)} = D^{m(t)} - E(t)^{m(t)}$ is chosen by $R$ independent mode selection algorithms.

For simplicity, all queues are considered to be *store-and-forward*. Hence, current fragment arrivals are registered *at the end* of the slot while fragment service and departures *during* the slot. Therefore, it is not allowed for any fragments to both arrive and depart in the same slot. Moreover, we assume *per-class/flow queueing* in the sense that if jobs/fragments are differentiated by class/flow they are queued up in separate (logical) queues in the system. Such class/flow differentiation may reflect distinct paths/routes of nodes that various jobs/fragments may need to follow through the network or diverse service requirements they might have at the nodes, depending on the class/flow they belong to.

## 3 Stability and Throughput

We seek to develop distributed resource allocation algorithms with good global performance. In particular, we focus on maximizing the throughput of the entire network via local/regional scheduling algorithms. An arrival load vector is feasible if there exists a sequence of service modes to match load. The combination of regional mode-selection algorithms is throughput-maximizing if it generates such a mode sequence for *any* servicable arrival rate.

We utilize the concept of *rate stability* in our throughput analysis of the system. In particular, we seek algorithms which ensure that the long-term fragment departure rate from each queue is equal to the long-term arrival rate. Such algorithms must satisfy $\lim_{t\to\infty} \frac{\sum_{s=0}^{t-1}[A_q(s)+E_q(s)]}{t} = \lim_{t\to\infty} \frac{\sum_{s=0}^{t-1} D_q(s)}{t}$. Since the left hand side represents the effective arrival rate to the queues. From the definition of change vectors, this is equivalent to

$$\lim_{t\to\infty} \frac{\sum_{s=0}^{t-1} S_q(s)}{t} = \lim_{t\to\infty} \frac{\sum_{s=0}^{t-1} A_q(s)}{t} = \rho_q \tag{3}$$

for each $q \in \mathcal{Q}$, that is, there is fragment *flow conservation* through the system. The objective of this analysis is to develop algorithms for these systems to select $m(t)$, (and hence $S(t) = S^{m(t)}$), in each timeslot in a way that ensures (3) will always hold; such a system is *rate stable*.

Given the available regional modes, and hence the global set $\mathcal{M}$ of modes that can be applied in each timeslot, we can define the maximum allowable arrival rate to the network. Since the evolution is described in terms of *change vectors*, this has a nice geometric intuition.

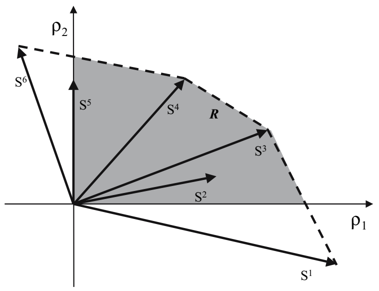


**Fig. 2. The stability domain.** The set of allowable arrival rate vectors $\rho$ is called the stability domain $\mathcal{R}$. For any $\rho$ in the region $\mathcal{R}$ above, there is a convex combination of service modes which would apply a total service rate to each queue which is at least the arrival rate to that queue. For $\rho$ outside there is no such combination. Service modes with total negative components such as $S^1$ and $S^6$ above may contribute to the stability domain without being inside the stability domain itself.

The **stability domain** $\mathcal{R}$ of the network described above is the set of all load vectors $\rho$ for which rate stability in (3) is maintained under at least one feasible scheduling algorithm. The stability domain can be expressed as

$$\mathcal{R} = \left\{ \rho \in \Re_+^Q : 0 \le \rho \le \sum_{m\in\mathcal{M}} \phi^m S^m, \quad \text{for some } \phi^m \ge 0 \text{ with } \sum_{m\in\mathcal{M}} \phi^m = 1 \right\} \tag{4}$$

where each $m = \{m_1, m_2, ..., m_R\}$ is comprised of $R$ regional modes.

The intuition is that a load vector $\rho$ is in the stability domain $\mathcal{R}$ if it is dominated (covered) by a convex combination of the change vectors $S^m$ induced under the different modes $m \in \mathcal{M}$ (and corresponding modes $m_r$ in each $\mathcal{M}_r$). This is illustrated in

Figure 2. The stability domain is the intersection of the convex hull of available modes with the positive quadrant. A similar regional stability domain could be defined, but (4) captures all of the cross-regional relationships. If $\rho \notin \mathcal{R}$ it is easily seen that rate stability cannot be maintained for all queues under any mode-selection algorithm.

If $\rho$ were known in advance, selecting each mode $m$ for a fraction $\phi^m$ of the time (in particular, using the corresponding regional modes for that fraction) would guarantee maximal throughput. This could be achieved through round robin or randomized algorithms. However, we are interested in dynamic scheduling algorithms which maintain rate-stability (as in (3)) for all $\rho \in \mathcal{R}$, without any prior knowledge of $\rho$. A scheduling algorithm which ensures that (3) holds for any allowable arrival rate $\rho \in \mathcal{R}$ is referred to as *throughput maximizing*.

## 4 Projective Cone Scheduling Algorithms

The dynamic selection of mode $m_r(t)$ in each region at each timeslot, based on queue backlog information without knowledge of the load vector $\rho$, is the focus of this work. We introduce a class of algorithms to select the regional modes based on a weighted comparison of the total service applied to queues in the region under each mode. Limited special cases of these algorithms have previously been shown to be throughput maximizing in switches [14], where a single region (R=1) without forwarding ($E^m = 0$) was considered. For these algorithms, it is assumed that all of the change vectors associated with modes are known, and the backlog state (but not arrival rate) can be observed.

Projective Cone Scheduling (PCS) algorithms are characterized by a fixed $Q \times Q$ positive diagonal matrix $\mathbf{W}$ where each element $\mathbf{W}_{qq} = w_q > 0$ is the relative weight for queue $q$, and all off-diagonal elements $\mathbf{W}_{pq} = 0, p \neq q$. Given an arbitrarily fixed $Q \times Q$ positive diagonal matrix $\mathbf{W}$, the **projective cone scheduling (PCS) algorithm** selects the mode $m_r(t)$ which satisfies

$$\begin{aligned} m_r(t) &= \arg\max_{m_r \in \mathcal{M}_r(X(t))} \langle S^{m_r}, \mathbf{W}X(t) \rangle \\ &= \arg\max_{m_r \in \mathcal{M}_r(X(t))} \{ \textstyle\sum_{q \in \mathcal{Q}_r} X_q(t) w_q D_q^{m_r} - \sum_{q \in \mathcal{F}_r} X_q(t) w_q E_q^{m_r} \} \end{aligned} \tag{5}$$

If there is more than one maximizing mode, any one can be arbitrarily selected. Notice that the selection of $m_r(t)$ is based on three values: $(i)$ the level of backlog the queues served and forwarded to by the region, $(ii)$ the weighted values $w_q$ in the matrix $\mathbf{W}$, and $(iii)$ the level of effective service applied to the queues under mode $m_r$. Therefore, the algorithm favors pushing the *most* jobs from *longest* queues to the *shortest* queues (or out of the network) at the (weighted) *highest possible rate*.

Because of the distributed selection, if each region maximizes $\langle S^{m_r}, \mathbf{W}X \rangle$, over all modes $m_r \in \mathcal{M}_r(X)$, then in total the mode $m = (m_1, m_2, ...m_R)$ will also maximize $\langle S^m, \mathbf{W}X \rangle = \sum_{r=1}^{R} \langle S^{m_r}, \mathbf{W}X \rangle$ over all combinations of modes that could be selected.

The name PCS of this algorithm class comes from two key geometric observations. First, the inner product formulation means that the selection of $m(t)$ is equivalent to maximizing the length of the projection of vector $S(t)$ in the direction of $\mathbf{W}X(t)$. Second, the cone structure of PCS algorithms was observed in [14]. Indeed, the algorithms can be described in terms of the conic spaces that each service configuration corresponds to. For each mode $m \in \mathcal{M}$, let $\mathcal{C}^m$ be the set of backlog vectors $X$ for which $S^m$ would be chosen under the PCS algorithm, that is,

$$\mathcal{C}^m = \left\{ X \in \Re^Q_{0+} : \langle S^m, \mathbf{W}X \rangle = \max_{m' \in \mathcal{M}(X)} \left\langle S^{m'}, \mathbf{W}X \right\rangle \right\} \tag{6}$$

This is the set of backlog vectors on which $S^m$ has the maximal projection amongst all modes in $\mathcal{M}$. Observe that the PCS algorithm can now be defined as follows:

$$\boxed{\text{when } X \in \mathcal{C}^m, \text{ use the service mode } m} \tag{7}$$

The cones $\{\mathcal{C}^m, m \in \mathcal{M}\}$ form a (soft) partition of the backlog space[2]. The partition is soft since some backlog vectors lie on the boundary of more than one cone, and whenever the backlog is on that boundary any of the 'boundary' modes can be selected. In fact, each region is itself a cone algorithm allocating resources according to the queues affected by that region. This corresponds to projecting the $Q$-dimensional backlog space vector onto the plane with $X_q = 0$ for all queues outside of $\mathcal{Q}_r$ and $\mathcal{F}_r$.

When a system operates under a PCS algorithm with fixed matrix $\mathbf{W}$, the evolution can be observed through this cone structure. As the backlog changes (due to arrivals and departures), the backlog vector $X(t)$ moves around the cones which make up the backlog space, using the service mode corresponding to the current cone $\mathcal{C}^m$ where $X(t)$ is located. Figure 3 illustrates the relationship between the available service modes and the backlog cones $\mathcal{C}^m$. In general, some cones may actually be degenerate (like those corresponding to non-extremal service vectors) and several cones may share common boundaries. The degenerate cones correspond to service modes for which there is no backlog vector $X$ which maximizes $\langle S, \mathbf{W}X \rangle$ at that configuration. These would never be selected by the PCS algorithms.

Due to the conic structure, and geometric nature of the stability proof, PCS algorithms can easily be extended to *delayed* PCS ones. These are relaxations of the original PCS algorithm, where the optimal service mode (according to $\max \langle S, \mathbf{W}X \rangle$) may not be activated immediately upon the backlog entering the appropriate cone. We define a PCS algorithm to be *Delayed* if for some integer $K$, whenever the system backlog $X(t)$ enters the cone $\mathcal{C}^{m^*}$ (for any fixed mode $m^*$) and stays there drifting for more than $K$ slots, it will use $m \in \arg\max \langle S^m, \mathbf{W}X \rangle$ from the $K^{th}$ time until leaving the cone[3]. It is clear that for $K = 0$ we have the original PCS algorithms. The delay $K$ allows for many interesting and practical extensions of PCS stability. For the special case of a single switch, such extensions were studied in [14]. For example, if preemption is not allowed in job scheduling, a delayed PCS algorithm could be implemented by forcing modes to stay fixed until all jobs are completed.

**Theorem 4.1 (Throughput Maximization of PCS Algorithms).** *If service modes in the described queueing network are selected by a delayed PCS algorithm with $(i)$ fixed,*

[2] That is, $\bigcup_{m \in \mathcal{M}} \mathcal{C}_m = \Re^Q_{0+}$.

[3] This has been carefully defined to allow for the possibility that more than one mode is maximizing for each $X$; it need not use exactly $m^*$.

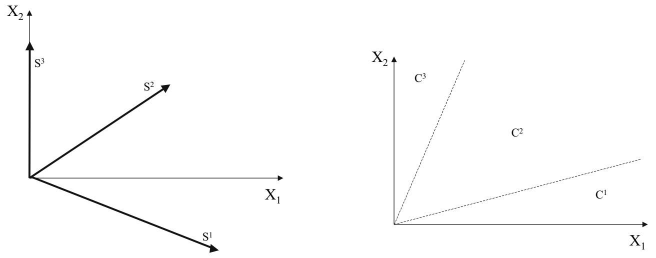


**Fig. 3. PCS as a Cone Algorithm.** The above graphs demonstrate the geometry of the PCS algorithms. In this example, the system has one region and two queues with three service modes $\mathcal{M} = \{1, 2, 3\}$. The associated departure and entering rates are $D^1 = (4, 0), E^1 = (0, 1), D^2 = (3, 2), E^2 = (0, 0), D^3 = (0, 3), E^3 = (0, 0)$. The first graph plots the service change vectors $S^m$ that are available. The second graph shows how the cones $\mathcal{C}^m$ subdivide the workload space when the identity matrix $\mathbf{W} = \mathbf{I}$ is used. For example, if the backlog $X$ is within the cone $C^1$ then service mode 1 is used. Whenever the backlog drifts into a new cone, the new service configuration is selected.

*finite delay of $K$ timeslots, $(ii)$ the arrival process satisfying $\rho \in \mathcal{R}$ and bounded arrivals, $(iii)$ the set of available service modes satisfying the three assertions on mode availability, and $(iv)$ fixed positive diagonal matrix $\mathbf{W}$, then every queue in the network will be rate stable. That is, PCS algorithms are throughput maximizing.*

The proof is provided in appendix A, and follows a geometrically motivated methodology, leveraging some previous results from [14]. Some key differences from [14] include $(i)$ the service modes in [14] are restricted to those without forwarding or feedback (the corresponding service modes have $E^m = 0$ for each $m$, hence $S^m = D^m \geq 0$), $(ii)$ the service mode set in [14] is independent of the current state, whereas in this work the mode set is defined by the state (idle service to an empty queue may happen in [14] but cannot in this work), $(iii)$ in [14], the diagonal matrix $\mathbf{W}$ was replaced by a more general $\mathbf{B}$ matrix which allows nonpositive off diagonal elements (this class of policies turns out not to guarantee maximal throughput in the network case described here), and $(iv)$ in [14], mode selection is done by a global scheduler rather than by autonomous regions.

The intuition for the proof is that if the network is not rate stable, there must be a subsequence of time on which the backlog is growing in some queues. Observing that the backlog growth can be described in terms of the cones in PCS, it is shown that unbounded growth within any of the cones is impossible, and hence the system must be kept stable for any such policy.

Here we present a simple example to highlight some of the benefits of PCS algorithms. We consider the network illustrated in Figure 4, a single region comprised of four queues and one server. Service at rate one can be applied to any one queue in a single timeslot. Two of the queues receive fragments from outside the network, and can forward to either of the downstream queues. Fragments departing the downstream queues all leave the system immediately.

We compare PCS with two simple algorithms. Randomized and round robin algorithms can achieve maximum throughput if they have prior knowledge of the long term

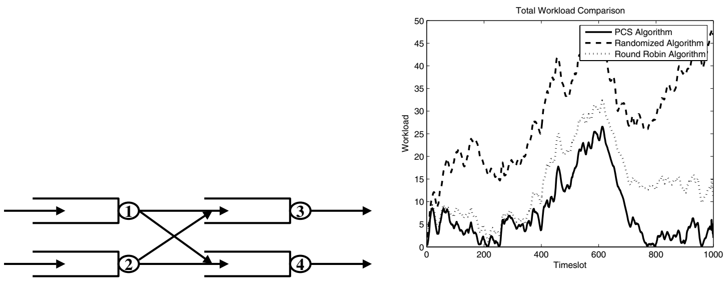


**Fig. 4. Regional Scheduling Example.** The left figure shows the region on which the comparison simulations were run. The network has four queues and the scheduler has to decide which one of the four queues to serve in each timeslot. Each of the 'front' queues can forward to either of the 'back' queues, and the back queues send the fragments out of the network entirely. This region allows six different service modes. For example, the mode forwarding from queue 1 to 3 is represented by $D^1 = (1,0,0,0), E^1 = (0,0,1,0), S^1 = (1,0,-1,0)$.
The performance of randomized, round robin and PCS algorithms is compared in the second figure. The total backlog over 1000 timeslots is recorded, and it is seen that the PCS algorithm performs significantly better than the other two. The round robin performs better than the randomized algorithm due to its periodic sequence of forwarding to then serving the back queues.

arrival rate vector $\rho$, as long as $\rho \in \mathcal{R}$ from definition 4 [15]. Both classes use the set $\{\phi^m\}_{m \in \mathcal{M}}$ which satisfy (4), and assume this set can be known or calculated in advance. Therefore, by using each mode $m$ the fraction $\phi^m$ of the total time, rate stability is guaranteed.[4]

Randomized algorithms use the policy in every timeslot $t$ to *select mode $m$ with probability $\phi^m$*. They are very simple to implement, requiring only a coin-flip operation and no online calculation at each timeslot. Round robin algorithms use the same principle, but with a deterministic ordering of modes instead of a randomized selection. For some fixed batch size $T$, round robin algorithms use the algorithm *for each $m$, use mode $m$ for $\phi^m T$ timeslots*. If $\phi^m T$ is not an integer number of timeslots, then rounding should be done in such a way to ensure that the long term average fraction of time spent using configuration $m$ is $\phi^m$.

We compare the performance of PCS algorithms, randomized algorithms and round robin algorithms in Fig. 4. The three algorithms were each applied to the network with four queues and both tandem and parallel features. This comparison provides an illustration of the features of each algorithm. The total backlog in the four queues is recorded over the same sequence of arrivals under each policy. The PCS algorithms perform significantly better than the randomized and round robin algorithms, with lower backlogs and hence lower delays over the network. The round robin algorithms perform an intermediate level between the two.

PCS algorithms have many attractive features, including the ability to naturally load balance the network [15]. Further, the delayed-PCS algorithms account for any time-

[4] In each case, if mode $m$ is not available for $X(t)$, the nearest available mode is chosen. For example, if $X_q < D_q^{m(t)}$ for some $q$, then $D^m$ is reduced until the mode becomes available. This corresponds to serving as many fragments as possible in each timeslot.

lag in information gathering, and allow regions to act asynchronously. Due to space limitations, we leave further discussion to future work.

## 5 Conclusions and Further Research

We have introduced a general methodology for modeling autonomic computing networks with distributed resource management. This models regional job scheduling in generalized networks. From a performance perspective, we have shown that the class of distributed PCS algorithms guarantee maximal throughput in these networks. They can be implemented using very efficient local search principles even in large networks. Further, the matrix $\mathbf{W}$ of implementation parameters allows a network manager to shape the quality of service performance according to priorities. The complexity reduction and performance control of PCS algorithms is the subject of ongoing research.

## References

1. M. Andrews and L. Zhang. Achieving stability in networks of input-queued switches. *ACM/IEEE Trans. on Networking*, 11(5):848–357, 2003.
2. M. Armony and N. Bambos. Queueing dynamics and maximal throughput scheduling in switched processing systems. *Queueing Systems*, 44(3):209, 2003.
3. J. G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53(2):197–218, 2005.
4. J. G. Dai and P. Prabhakar. The throughput of data switches with and without speedup. *IEEE INFOCOM*, pages 556–564, 2000.
5. S. Hariri. Autonomic computing: research challenges and opportunities. In *IEEE International Conference on Pervasive Services, ICPS*, 2004.
6. V. Kapoor. Services and autonomic computing: a practical approach for designing manageability. In *IEEE International Conference on Services Computing*, volume 2, pages 41 – 48, 2005.
7. E. Leonardi, M. Mellia, M.A. Marsan, and F. Neri. On the throughput achievable by isolated and interconnected input-queueing switches under multiclass traffic. In *IEEE INFOCOM*, 2002.
8. L. Mastroleon, N. Bambos, C. C. Kozyrakis, and D. Economou. Autonomic power management schemes for internet servers and data centers. In *IEEE Global Telecommunications Conference, (GLOBECOM)*, volume 2, pages 943–947, 2005.
9. N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267, 1999.
10. M.J. Neely, E. Modiano, and C.E. Rohrs. Dynamic power allocation and routing for time varying wireless networks. *IEEE JSAC*, 2005.
11. G. Paleologo and N. Bambos. Autonomic admission control for networked information servers. telecommunications network design and management. *Operations Research and Computer Science Interfaces*, 23:227–244, 2003.
12. A. Ranganathan and R.H. Campbell. Autonomic pervasive computing based on planning. In *International Conference on Autonomic Computing*, pages 80–87, 2004.
13. J. Rolia, X. Zhu, and M. Arlitt. Resource access management for a utility hosting enterprise applications. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 549–562, 2003.

14. K. Ross and N. Bambos. Local search scheduling algorithms for maximal throughput in packet switches. In *IEEE INFOCOM*, 2004.
15. K. Ross and N. Bambos. Packet scheduling across networks of switches. In *ICN*, 2005.
16. D. Seao, A. Ali, W. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *International Conference on Computer Architecture*, pages 432–443, 2005.
17. A. Stolyar. Maxweight scheduling in a generalized switch: State space collapse and equivalent workload minimization in heavy traffic. *Annals of Applied Probability*, 4(1):1–53, 2004.
18. L. Tassiulas and P.P. Bhattacharya. Allocation of interdependent resources for maximal throughput. *Stochastic Models*, 16(1), 1999.
19. L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, 1992.
20. B. Towles, W. J. Dally, and S. Boyd. Routing ii: Throughput-centric routing algorithm design. In *ACM Symposium on parallel algorithms and architectures*, 2003.
21. W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *International Conference on Autonomic Computing*, pages 70 – 77, 2004.
22. S.R. White, J.E. Hanson, I. Whalley, D.M. Chess, and J.O. Kephart. An architectural approach to autonomic computing. In *International Conference on Autonomic Computing*, pages 2 – 9, 2004.
23. L. J. Zhang, H. Li, and H. Lam. Services computing; grid applications for today. *IT Professional*, pages 5–7, July-August 2004.

## A Appendix

We present the stability proofs in this appendix. First note that from (2) we see that in the long term

$$X(t+1) = X(0) + \sum_{s=0}^{t-1} A(s) - \sum_{s=0}^{t-1} S(s) \tag{8}$$

where $X(0)$ is the vector of initial backlog levels. Further, from (8) and (3) we see that

$$\lim_{t\to\infty} \frac{X(t)}{t} = 0 \tag{9}$$

is equivalent to rate stability.

■

**Proposition A.1** *For any backlog vector $X \in \Re_+^Q$ and any $\rho \in \mathcal{R}$, there exists $m^* \in \mathcal{M}$ for which*

$$\left\langle S^{m^*}, \mathbf{W}X \right\rangle \geq \langle \rho, \mathbf{W}X \rangle \tag{10}$$

*for any set of service modes.*

**Proof:**
By definition 4 of stability, $\rho_q \leq \sum_{m\in\mathcal{M}} \phi^m S_q^m$ for all $q$ for some $\{\phi^m\}_{m\in\mathcal{M}}$, with $\sum_{m\in\mathcal{M}} \phi^m = 1, \phi^m \geq 0$. Multiplying through by $[\mathbf{W}X]_q = w_q X_q \geq 0$, we see that $\rho_q[\mathbf{W}X]_q \leq \sum_{m\in\mathcal{M}} \phi^m [\mathbf{W}X]_q S_q^m$ for all $q$. Now, summing over all $q \in \mathcal{Q}$,

$$\begin{aligned}\langle \rho, \mathbf{W}X\rangle &\leq \sum_{m\in\mathcal{M}} \phi^m \langle S^m, \mathbf{W}X\rangle \\ &\leq \sum_{m\in\mathcal{M}} \phi^m \max_{m^*\in\mathcal{M}} \left\langle S^{m^*}, \mathbf{W}X\right\rangle = \max_{m^*\in\mathcal{M}} \left\langle S^{m^*}, \mathbf{W}X\right\rangle \end{aligned} \tag{11}$$

∎

Further, by the three assertions of mode availability, we must have such an $m^* \in \mathcal{M}(X)$ for large $X$. That is, as the backlog vector grows, the PCS-optimal mode can always be selected.[5]

Now we are ready to complete the stability proof of theorem 4.1. First, note that the backlog 'drifts' between different cones as defined in (6).

Consider an arbitrarily fixed arrival trace $A(t), t \in \{0, 1, 2, ...\}$ satisfying (1) with $\rho \in \mathcal{R}$, where $\mathcal{R}$ is defined by (4). This infinite sequence of arrivals could be the result of a deterministic or probabilistic structure, and is assumed to be fixed.

The objective of this proof is to show (9) that $\lim_{t\to\infty} \frac{X(t)}{t} = 0$. Since $\mathbf{W}$ is positive-diagonal it suffices to show that $\lim_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{W}\frac{X(t)}{t}\right\rangle = 0$.

Arguing by contradiction, let us assume that $\limsup_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{W}\frac{X(t)}{t}\right\rangle > 0$ and that this limit is attained on the increasing, unbounded time sequence $\{t_a\}_{a=1}^{\infty}$ with

$$\lim_{a\to\infty} \frac{X(t_a)}{t_a} = \eta \neq 0 \tag{12}$$

and the backlog blows up along the vector direction $\eta$ on $\{t_a\}_{a=1}^{\infty}$. Given this assumption, the contradiction is established by showing there must exist a distinct (but related) time sequence $\{s_b\}_{b=1}^{\infty}$ which has the property

$$\lim_{b\to\infty} \left\langle \frac{X(s_b)}{s_b}, \mathbf{W}\frac{X(s_b)}{s_b}\right\rangle > \lim_{a\to\infty} \left\langle \frac{X(t_a)}{t_a}, \mathbf{W}\frac{X(t_a)}{t_a}\right\rangle = \langle \eta, \mathbf{W}\eta\rangle \tag{13}$$

But by definition, $\{t_a\}_{a=1}^{\infty}$ should achieve the superior limit, and the existence of the sequence $\{s_b\}_{b=1}^{\infty}$ contradicts the original assertion. The series is constructed and two key properties are identified, then it is shown how these properties lead to the contradiction.

**Constructing the sequence $\{s_b\}_{b=1}^{\infty}$.** It is instructive to consider service cones in terms of the backlog vectors they contain. For that reason we define the cone of each backlog vector as

$$\mathcal{C}(X) = \{X' \in \Re_{0+}^{Q} : \exists\, m \text{ with } X \in \mathcal{C}^m, X' \in \mathcal{C}^m\} \tag{14}$$

[5] It is important to note here that it is on this proposition that the more general projective scheduling algorithms in [14] break down for the network case. In particular, it can not be guaranteed that $m^* \in \mathcal{M}(X)$ is available when the matrices $\mathbf{W}$ are replaced by $\mathbf{B}$ matrices which have nonpositive off diagonal elements. One can construct scenarios in which $\langle S^m, \mathbf{B}X\rangle < \langle \rho, \mathbf{B}X\rangle$ for all $m \in \mathcal{M}(X)$ and $\rho \in \mathcal{R}$.

to be the set of all of the backlog vectors which share a common cone to $X$, and could hence be optimally operated on by the same mode.

The sequence is constructed in a similar way to in [14], with a simplification due to allowing no idle service. Let $r_a = \max\{t < t_a : X(t) \notin \mathcal{C}(\eta)\}$ be the last time before $t_a$ that the backlog $X(t)$ is not included in the cone $\mathcal{C}(\eta)$. This is the last time that $X(t)$ crosses from outside $\mathcal{C}(\eta)$ to inside, hence, $X(t) \in \mathcal{C}(\eta)$ for every $t \in [r_a + 1, t_a]$ and the backlog drifts in $\mathcal{C}(\eta)$ throughout that interval. By convention $r_a = 0$ if the backlog has always been in $\mathcal{C}(\eta)$ before $t_a$. Let $s_a = \max\{r_a, (1 - \epsilon_3)t_a\}$ for some $\epsilon_3 \in (0, 1)$ and for all $a > 0$. The second term guards against the case where the backlog eventually remains in the same cone indefinitely.

**Lemma A.1** *For $\{s_a\}_{a=1}^{\infty}$ and $\{t_a\}_{a=1}^{\infty}$ as constructed above,*

$$\liminf_{a\to\infty} \frac{t_a - s_a}{t_a} = \epsilon > 0 \tag{15}$$

**Proof:** The proof follows the same argument as that presented in lemma 1.1 in [14]. ∎

Thus, sequences can always be established for which:[6]

**I.** $\lim_{b\to\infty} \frac{t_a - s_a}{t_a} = \epsilon \in (0, 1)$ and $s_a < t_a$ for all $a$.
**II.** $X(t) \in \mathcal{C}(\eta)$ for all $t \in [s_a + 1, t_a]$ and each $a$. This implies that the backlog $X(t)$ drifts within the cone surrounding $\eta$ throughout the time interval $[s_a + 1, t_a]$.

Given this result one can select increasing, unbounded subsequences $\{s_b\}_{b=1}^{\infty}$ and $\{t_b\}_{b=1}^{\infty}$ of $\{s_a\}_{a=1}^{\infty}$ and $\{t_a\}_{a=1}^{\infty}$ on which the limit is equal to the inferior limit.

**Establishing the Contradiction.** Here the argument is established that the properties of the constructed sequences lead to the aforementioned contradiction.

**Lemma A.2** *If sequences $\{s_a\}_{a=1}^{\infty}$ and $\{t_a\}_{a=1}^{\infty}$ satisfy the properties I and II above then the superior limit is not attained on the sequence $\{t_a\}_{a=1}^{\infty}$ as initially assumed. This establishes the required contradiction.*

**Proof:** [7]
Consider the change in backlog from time $s_a$ to time $t_a$, that is,

$$X(t_a) - X(s_a) = \sum_{t=s_a}^{t_a-1} (A(t) - S(t)) = \sum_{t=s_a}^{s_a+K-1} (A(t) - S(t)) + \sum_{t=s_a+K}^{t_a-1} (A(t) - S(t)) \tag{16}$$

For sufficiently large $s_a, t_a$ (and hence sufficiently large $X(t)$), we have $S(t) = S^{m^*} = \arg\max_{m\in\mathcal{M}} \langle S^m, \mathbf{W}X\rangle$ from time $s_a + K$ until $t_a$ where $m^*$ is the mode selected by the standard PCS algorithm from all possible modes.

[6] Note that in [14] an additional property is utilized. Here that property is not necessary due to the definition of modes, and the lack of idle service potential.
[7] This proof follows simpler than the stability proof in [14], due to the careful definition of $\mathcal{M}(X)$.

Multiplying each element in (16) by $(\mathbf{W}\eta)_q = w_q\eta_q$, and summing over all terms we have

$$\begin{aligned}&\langle X(t_a) - X(s_a), \mathbf{W}\eta\rangle \\ &= \sum_{t=s_a}^{s_a+K-1} (A(t) - S(t)) + \sum_{t=s_a+K}^{t_a-1} \langle A(t), \mathbf{W}\eta\rangle - \left\langle S^{m^*}, \mathbf{W}\eta\right\rangle (t_a - s_a - 1) \quad (17)\end{aligned}$$

where $m^* \in \arg\max \langle S^m, \mathbf{W}\eta\rangle$. The last term follows from property II since the service configuration $S^*$ is used from from time $s_a + K$ until $t_a$. Finally, observe that $\lim_{a\to\infty} \frac{\sum_{t=s_a}^{t_a-1} A(t)}{t_a - s_a} = \rho$ from property I and part of lemma 1.2 in [14]. Dividing (17) by $(t_a - s_a)$ and letting $a\to\infty$, the first term (being bounded) disappears, and we have

$$\lim_{a\to\infty} \left\langle \frac{X(t_a) - X(s_a)}{t_a - s_a}, \mathbf{W}\eta\right\rangle = \langle \rho, \mathbf{W}\eta\rangle - \langle S^*, \mathbf{W}\eta\rangle = -\gamma(\eta) \le 0 \quad (18)$$

from proposition A.1. Now, since $\lim_{a\to\infty} \frac{X(t_a)}{t_a} = \eta$, using property I and (18) the following inequality holds

$$\begin{aligned}\lim_{a\to\infty} \left\langle \frac{X(s_a)}{s_a}, \mathbf{W}\eta\right\rangle &= \lim_{a\to\infty} \left\{ \left\langle \frac{X(s_a) - X(t_a)}{s_a}, \mathbf{W}\eta\right\rangle + \left\langle \frac{X(t_a)}{s_a}, \mathbf{W}\eta\right\rangle \right\} \\ &= \lim_{a\to\infty} \left\{ \frac{s_a - t_a}{s_a} \left\langle \frac{X(t_a) - X(s_a)}{t_a - s_a}, \mathbf{W}\eta\right\rangle + \frac{t_a}{s_a} \left\langle \frac{X(t_a)}{t_a}, \mathbf{W}\eta\right\rangle \right\} \\ &\ge \frac{\epsilon}{1-\epsilon}\gamma(\eta) + \frac{1}{1-\epsilon} \langle \eta, \mathbf{W}\eta\rangle > \langle \eta, \mathbf{W}\eta\rangle \quad (19)\end{aligned}$$

The last inequality is due to the facts that $\epsilon \in (0, 1)$ and $\gamma(\eta) \ge 0$. Now, successive thinnings of the components of the backlog vector will obtain an increasing unbounded subsequence $\{s_b\}_{b=1}^{\infty}$ of $\{s_a\}_{a=1}^{\infty}$ such that $\lim_{b\to\infty} \frac{X(s_b)}{s_b} = \psi$ and from (19) $\langle \psi, \mathbf{W}\eta\rangle > \langle \eta, \mathbf{W}\eta\rangle$. But $\mathbf{W}$ is *positive-diagonal*, so (A) implies that $\langle \psi, \mathbf{W}\psi\rangle > \langle \eta, \mathbf{W}\eta\rangle$ or

$$\lim_{b\to\infty} \left\langle \frac{X(s_b)}{s_b}, \mathbf{W}\frac{X(s_b)}{s_b}\right\rangle = \langle \psi, \mathbf{W}\psi\rangle > \langle \eta, \mathbf{W}\eta\rangle = \limsup_{t\to\infty} \left\langle \frac{X(t)}{t}, \mathbf{W}\frac{X(t)}{t}\right\rangle \quad (20)$$

giving a contradiction to the definition of $\eta$. This completes the proof of Lemma A.2. ∎

Lemma A.2, together with the construction in the previous section, completes the proof of Theorem 4.1. ∎

# Investigating Global Behavior in Computing Grids

Kevin L. Mills and Christopher Dabrowski

National Institute of Standards and Technology
Gaithersburg, Maryland 20899 USA
{kmills, cdabrowski}@nist.gov

**Abstract.** We investigate effects of spoofing attacks on the scheduling and execution of basic application workflows in a moderately loaded grid computing system using a simulation model based on standard specifications. We conduct experiments to first subject this grid to spoofing attacks that reduce resource availability and increase relative load. A reasonable change in client behavior is then introduced to counter the attack, which unexpectedly causes global performance degradation. To understand the resulting global behavior, we adapt multidimensional analyses as a measurement approach for analysis of complex information systems. We use this approach to show that the surprising performance fall-off occurs because the change in client behavior causes a rearrangement of the global job execution schedule in which completion times inadvertently increase. Finally, we argue that viewing distributed resource allocation as a self-organizing process improves understanding of behavior in distributed systems such as computing grids.

## 1 Introduction

The Internet provides a communications infrastructure for distributed applications with global reach and massive scale. Already, designers have specified software components [1] that developers can use to construct and deploy computing and data grids [2], [3]. How will such distributed systems behave? One possibility is that distributed systems are complex adaptive systems [4] consisting of interconnected components, where change in any component propagates to many other components through feedback-driven interactions over space and time. Such interactions may arise through indirect coupling (e.g., sharing resources) exhibited as individual actors adapt their behavior based on information gained through feedback. Complex systems often exhibit the property of self-organization [4], which drives global system behavior (e.g., job scheduling and execution) from less organized states toward coherent patterns. We suspect self-organization will arise with increasing frequency as distributed systems pervade the globe. Unfortunately, little is known about how to detect, predict, and shape global behaviors in distributed systems.

Here, we study an important aspect of global behavior in grid systems, envisioned to offer high-performance computing as a commodity for those who require substantial processing cycles to design more effective drugs or engine components, to do financial risk analyses, to model global climate, to understand our universe, and so on. We consider distributed protocols for allocating processor resources deployed across a global grid. Our research shows such protocols can yield a self-organizing, global pattern of job scheduling and execution, as various independent clients sense the state of available processors and adapt accordingly. We investigate distributed resource allocation in a moderately loaded grid that is subjected to an attack intended to reduce substantially the available computing resources. We introduce a small change in client behavior to mitigate effects of the attack, but find unexpectedly that the global pattern of job execution degrades rather than improves. This result illustrates that surprising global behavior can arise in a distributed system, and motivates our interest in finding techniques to reveal, understand, and shape system behavior.

In this paper, we make three main contributions. First, we define a grid simulator combining model components representing selected, standard specifications. We chose specifications based on the current posture of the Global Grid Forum, which suggests that future grid systems will be built from a combination of web services [5] and open grid services [6]. For functions lacking completed specifications, we modeled components from the Globus Toolkit 4 [7], an available grid framework that provides a significant level of capability. Our simulator allows us to model a plausible distributed system in significant detail. Second, we show that a moderately sized grid can exhibit unanticipated and undesirable global behavior arising from adaptive processes. We illustrate that adaptation by many individual actors can lead to self-organization on a global scale. Third, we describe and apply a multidimensional analysis approach to reveal underlying causes for observed global behavior. The analysis approach is adapted from the physical sciences, where spatiotemporal analyses [8] have long been a standard technique to model system dynamics. While we use spatiotemporal analysis, we increase the number of dimensions to account for logical partitions within the system we study. For example, we consider completion times among various job classes over space and time (a 4D analysis). We find multidimensional analyses provide more insight into system behavior than can be obtained by summarization through averages and variances.

The remainder of the paper is organized as five sections. Section 2 outlines related work to simulate grid systems and to investigate distributed resource allocation in grids. Section 3 describes our analysis approach. We discuss our detailed grid simulation model in Section 4, before describing our experiment design and metrics in Section 5. Section 6 presents our simulation results, investigates causes underlying an unexpected outcome, and describes scheduling and execution of jobs in computing grids as a self-organizing process.

## 2 Related Work

While there is significant research on simulating grids, little of that work studies scalability, effects of failures and attacks, or global behaviors. SimGrid [9], GridSim [10], and MicroGrid [11] provide toolsets for simulating grid applications on large-scale networks. These grid simulations do not combine model components representing selected web services, Globus Toolkit 4 components, and open grid services. Further, these simulators aim mainly to provide overall assessment of performance in network protocols and middleware, rather than isolating causal behaviors that unexpectedly affect global performance.

Numerous researchers have investigated resource allocation in large (simulated) grids using a decentralized approach in which clients employ independent schedulers. For instance, Ernemann et al. [12] report results suggesting that geographically distributed grids improve overall response time. Various studies [12], [13], [14], [15] have applied market-based economic models to optimize resource use and minimize cost when scheduling jobs in distributed grids. Other researchers [16], [17], [18] have investigated prioritization schemes, considering factors such as quality-of-service and workflow requirements. Only a few grid-scheduling studies consider effects of uncertainty. Krothapalli and Deshmukh [19] consider performance of alternative scheduling approaches given partial information. Chen and Maheswaran [20] consider how resource failure affects scheduling. Subramani et al. [21] attempt to identify and address causes of unexpected performance degradations in grids. These studies rely extensively on summary measures of performance, and provide little insight into underlying global behavior. Our paper contributes to such investigations and demonstrates an analysis approach providing insight into global system behavior and causes.

## 3 Analysis Approach

We conduct simulations defined by a set of parameters (e.g., space, demand, negotiation strategy, and failure-response behavior) and observe system dynamics over time with respect to various logical partitions (e.g., event type and job class). We represent the entire system state as a multidimensional space. To investigate selected system dynamics, we project various views of this space, using a three-step procedure: (1) subset the space along dimensions of interest, (2) partition the subset into equivalence classes, and then (3) transform each equivalence class into measures of interest. Subsequently, we plot derived views in 2D, 3D, or 4D, depending upon the characteristics of the equivalence classes.

We represent system state(s) as a space, $U$, of multidimensional points, $\vec{x}$, i.e.,

$$U = \{(\vec{x} = (n_x, d_x, a_x, p_x, s_x, j_x, e_x, i_x, o_x))\}. \tag{1}$$

Each of the dimensions is defined in Table 1. To explain our analysis procedure, we will derive two views used later in the paper to explore the effects of failure-response behavior ($s$) on two event types: reservations created (designated $E_1$) and task completions (designated $E_2$).

We begin by examining the effects of failure-response behavior on reservations created when demand ($d$) is 50% and the probability ($p$) of spoofing selected nodes is ½. We define a subspace, $V_1$, such that

$$V_1 = \{(\vec{x} \mid \vec{x} \in U \wedge d_x = 50 \wedge p = 0.5 \wedge e_x = E_1)\}. \tag{2}$$

Next, we partition subspace $V_1$ into equivalence classes, $Q_i$, where every class consists of points with a common time interval ($i$), specifically

$$Q_i = \{\vec{x} \mid \vec{x} \in V_1 \wedge i_x = i\}. \tag{3}$$

**Table 1.** Definition of dimensions locating each point in system state space

| Dimension | Variable | Range |
|---|---|---|
| Space | $n$ | $1 \le n \le N$, where $N$ is the number of observation points |
| Demand | $d$ | $10 \le d \le 100$, where d mod 10 = 0 |
| Negotiation Strategy | $a$ | $a \in \{A_1,\ldots,A_g\}$, where $g$ = number of strategies |
| Spoofing Probability | $p$ | $0 \le p \le 1$ |
| Failure-Response Behavior | $s$ | $s \in \{S_1,\ldots,S_h\}$, where $h$ = number of behaviors |
| Job Class | $j$ | $j \in \{J_1,\ldots,J_w\}$, where $w$ = number of job classes |
| Event Type | $e$ | $e \in \{E_1,\ldots,E_z\}$, where $z$ = number of event types |
| Time Interval | $i$ | $1 \le i \le (T / I)$, where $T$ is simulation run time and $I$ is the observation interval size and $i = t/I$ for $t$ = current simulation time |
| Observation | $o$ | integer |

Subsequently, we map portions of each equivalence class to a specific value by defining operators, $G_s(i)$, where

$$G_s(i) = \sum_{\substack{\vec{x}\in Q_i \\ s_x = s}} o_x . \tag{4}$$

This yields a 2D view, where one dimension represents a particular failure-response behavior ($s$) and the other dimension denotes specific time intervals ($i$) and each cell ($s$, $i$) contains an aggregate count of reservations created, obtained from equation 4. As discussed later, plotting such views for different $s$ reveals large differences in the pattern of reservations created over time. To investigate such differences in more detail, we define a new view of the system state space to consider the evolution of task completion times for particular job classes ($j$).

We begin by defining the subspace, $V_2$, of interest:

$$V_2 = \{\vec{x} \mid \vec{x} \in U \wedge d_x = 50 \wedge p_x = 0.5 \wedge e_x = E_2 \wedge (s_x = S_1 \vee s_x = S_2) \wedge (j_x = J_1 \vee j_x = J_2))\} \tag{5}$$

and then a relation, $R$, to form equivalence classes on $V_2$:

$$\vec{x}R\vec{y} \Leftrightarrow (s_x = s_y \wedge j_x = j_y), \tag{6}$$

where $\vec{y} \in V_2$. Here, $V_2 / R = \{Q_k\}$ forms ($k$ = 1, 2, 3, 4) equivalence classes, each combining one of two selected failure-response behaviors with one of two selected job classes. Next, we define a scaling factor, $f$, derived from the maximum observation in subspace $V_2$:

$$f = \max(o_x \mid \vec{x} \in V_2) + 1, \tag{7}$$

and then we define the following operators:

$$G_k(\vec{x}) = \underset{\vec{x} \in Q_k}{o_x} + 1 + (k-1) \times f. \tag{8}$$

## 4 Simulation Model

We find that simulation provides an excellent vehicle to investigate global behavior, for several reasons. First, simulation models allow construction of systems of large scale, which can be expensive to achieve in a test bed. Second, simulation models allow complete access to system state, which is impractical in a large deployed system. Third, simulation models provide rigorously controllable and repeatable conditions, which are difficult to ensure in a test bed of significant size. Fourth, simulation models allow incorporation of various levels of abstraction, while deployed systems typically include incidental complexity that is impractical to remove. Prior to conducting experiments, we verified our model for correct operation through extensive trials, removing several biases and errors in the process. In our experiment we used our model for qualitative analysis of system dynamics, rather then to make quantitative performance predictions; therefore, our verification process focused on ensuring correct interpretation of the standard specifications we modeled, rather than validating the model against measured performance data. In this section, we describe our simulation model, concentrating on the grid computing aspects.

### 4.1 Network and Web Services Model

We define a topology of sites, each located at a point (*x*, *y*, *z*). The *x*-*y* coordinates locate a site in the Internet and the *z* coordinate defines the distance in router hops from the site to the Internet. The model uses differences in *x*-*y* coordinates to compute Euclidean distances among sites, and then converts those distances to Internet router hops by assuming that routers are separated by a specified distance. The distance in hops between two sites is defined by the distance between the sites in Internet router hops plus the number of *z*-coordinate hops required for each site to reach the Internet. Messages flowing between sites are delayed in proportion to distance in hops. Messages flowing within sites incur simulated local-network transmission delays. Nodes are defined and allocated to sites. Each node has a *mailbox*, which simulates a sockets interface and related transport protocols, including multicast. Each node simulates CPU-execution time required by processes executing on the node. Nodes also include a standard set of services modeled after web services for messaging [22], addressing [23], and stateful resources [1]. At selected sites, our model deploys simulated information and index servers, which we model as service groups. We also define a two level hierarchy of index servers, linked together through simulated query aggregators

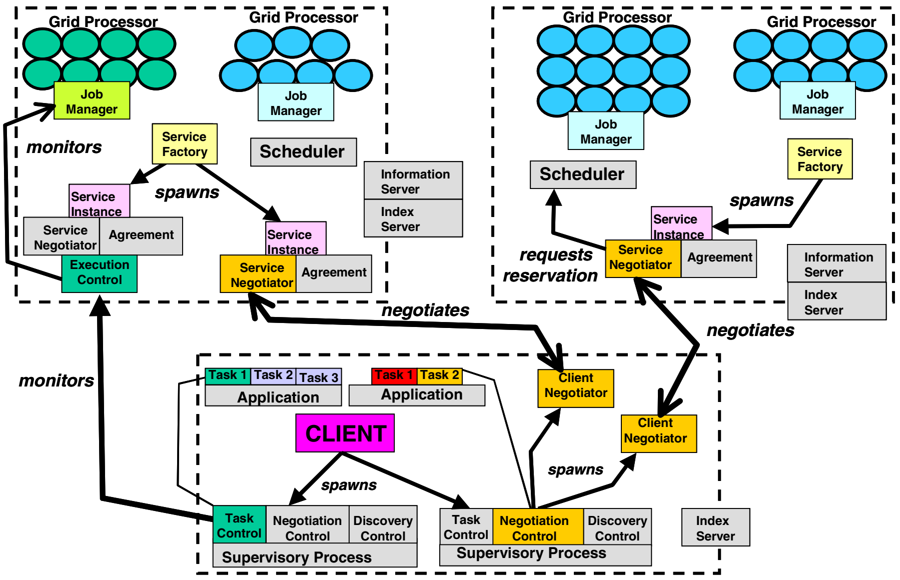


**Fig. 1.** Snapshot of system execution: client spawns supervisory processes for two applications

(modeled after the index service of Globus Toolkit 4) to form a monitoring and discovery service that grid clients use to discover the nature and location of available resources.

## 4.2 Grid Computing Model

At the application level, we model two main components, service providers and clients, found in well-known grid models (e.g., [2]). Service providers control availability of grid resources. Grid clients discover resources and enter into agreements for their use to execute client jobs. We describe these components and the procedures for reaching agreements and executing jobs. Figure 1 provides a simplified view of the model.

***Service Providers.*** We designate selected sites as service-provider sites, where we deploy grid services, service factories, and schedulers. We model grid services in two parts: (1) application code, which executes jobs provided by clients; and (2) grid processors, which provide platforms upon which application code executes. Grid processors may be either clusters or vector computers, both capable of parallel execution. Each grid processor implements a simulated job manager (modeled after the Distributed Resource Management System [28]) that responds to job requests by locating and loading appropriate application code and obtaining input files from the requesting client. The job is then queued until its start time.

Each grid service has a *service description*, whose attributes include: type of application code (or task type) and descriptions of available grid processors on which the application code may run. Each *grid processor description* identifies processor type (*pType*: cluster or vector), available parallelism (or *pFactor*), and processor speed (*pSpeed* in cycles/second). A *service factory* manages each service description.

To advertise service availability, a service factory registers the service description with a local information server. Remote clients discover service descriptions through an indexing server and then contact an associated service factory to enter into agreements for services. For each client request, a service factory spawns a transient entity called a *service instance*, which provides a *service negotiator* to negotiate a service *agreement* locally on behalf of a remote client. The service instance contains an agreement document [29] and maintains negotiation status. If an agreement is reached, the service instance launches an *execution controller* to act as local proxy on behalf of the remote client. The execution controller submits jobs and status requests and reports job status to the client. We model each service instance (containing service negotiator, agreement, and execution controller) as a single container with lifetime determined by the negotiation duration or, if successful, by job length.

Each service-provider site contains a *scheduler* that controls reservation of CPU time on all grid processors within the site. For a client to obtain an agreement, a service negotiator must first reserve (through the site scheduler) time on an appropriate processor component. Each site scheduler is independent of other schedulers and accepts reservations on a first-come, first-serve basis with backfilling. All clients are given equal priority. As a policy, each scheduler attempts to allocate tasks with smaller *pFactors* to smaller clusters, thus saving large grid clusters for tasks requiring greater parallelism.

***Grid Clients and Applications.*** We model each grid client as a set of independent applications, each with one or more tasks. While tasks in an application must execute sequentially in a workflow, each task contains subtasks that may execute in parallel. Each task is described by: task type (application code), *pFactor* (number of parallel processors required for subtasks), *pType* (cluster or vector), and *pCycles* (CPU cycles needed to run the task). The task duration, *tDuration*, may be computed as:

$$tDuration = tCycles / (pFactor \times pSpeed) \tag{9}$$

Clients create separate supervisory processes for each application. For each application task, the supervisor initiates service discovery, followed by negotiation to obtain an agreement to execute the task on a processor, and then monitors execution through a service instance. Since tasks are sequenced within an application, negotiation for a task is triggered when the previous task finishes. An application is complete when its last task finishes, at which time the supervisory process terminates.

We model supervisors as multiple components for: service discovery, agreement negotiation, and execution monitoring (including fault detection and recovery and job rescheduling). The discovery component activates on task completion to find the next task requiring services, and first queries a local index server for references to any remote information server with service descriptions matching task requirements. For each reference retrieved, the discovery component queries the associated information server to obtain matching service descriptions, which are then cached locally for use by the negotiation component. The negotiation component identifies the next incomplete task having no service agreement, ranks and selects cached discoveries for a task, and then creates a client negotiator for each selection. Selection criteria give higher priority to more recent discoveries and those that have not been tried previously. Similar to Condor-G [3], the negotiation component prioritizes discoveries during subsequent negotiations (as described below) to favor those that can execute the task sooner. If negotiation produces a successful agreement, a monitoring component registers with the service instance for notification of the outcome and accompanying output.

***Agreement Negotiation.*** Negotiation commences when a client negotiator is created to obtain an agreement for a specific task. The negotiator, provided with a service description and the address of a service factory, queries the factory to obtain an agreement template, containing a set of possible agreement terms and, optionally, a list of creation constraints. We use creation constraints to convey existing reservations for grid processors associated with a service factory. Client negotiators use this information to determine an earliest possible start time. Execution ceases for any client negotiator obtaining an unacceptably late start time; otherwise, each client negotiator instantiates an agreement template for a selected task and end time on a subset of available processors, and then forwards the template (as an agreement offer) to a service factory, which spawns a service negotiator.

Our model allows negotiation to proceed according to one of two strategies: single-reservation request (SRR) or multiple-reservation request (MRR). In SRR, which closely follows WS Agreement [29], the service negotiator immediately acknowledges the client's request and contacts the scheduler to obtain a reservation, requesting start and end times on the processors specified in the agreement offer. If a reservation is granted, the service negotiator forwards an acceptance to the client; otherwise, the negotiator forwards a rejection (which can occur because multiple applications may be competing for the same processors). In SRR, an offer is considered obligating; thus, acceptance instantiates an agreement that both parties must observe. Independent of outcome, the negotiation terminates.

***Negotiation Feedback.*** In MRR, negotiation may continue after initial rejection. MRR rejections contain an updated reservation list for applicable processors. A client negotiator may use this feedback to compose a "follow-up" offer to the service negotiator. This process may repeat, with additional follow-up offers, until an agreement is reached or negotiation is terminated. In contrast, SRR requires restarting negotiation to retrieve the template with the updated reservation list to use this feedback in preparing a new offer. The MRR strategy also differs from the SRR strategy in that client offers are not obligating. Using MRR, a client may create multiple client negotiators to simultaneously negotiate agreements for the same task and then accept the best one. The client may replace an existing observed agreement with a new agreement, if obtained prior to task execution. This again contrasts with the SRR strategy, where offers from client negotiators must be sequenced to prevent concurrent observed agreements. Each client was configured to negotiate with either MRR or SRR, while service negotiators handled both strategies. Both SRR and MRR adapt to feedback, as is characteristic of actors in self-organizing systems [4]. As we show in Sections 6.2 and 6.3, repeated attempts to secure services through negotiation lead to interactions where adaptation to feedback drives global resource allocation.

## 5 Experiment Description

We deployed our model in a simulated grid and conducted an experiment to compare the effectiveness and overhead of SRR and MRR. Below, we describe the experiment topologies, workload and design, and then define the metrics of comparison.

### 5.1 Experiment Topologies and Workload

In each experiment repetition, we generated a random topology by varying (uniformly between -4,000 and 4,000) the *x*-*y* coordinates (*z* = 2) of each site, which limited maximum inter-site distance to 16 hops. Each topology consisted of 42 sites: 30 service sites and 12 client sites. Each client site provided 25 applications for a total of 300 applications comprising 600 tasks. Each service site hosted a variable number of grid processors (allocation shown in Table 2) and one service factory to register service descriptions with a site-local information server. Service factories dynamically create service instances in response to client offers–one instance per client negotiator. Each service site contained one scheduler, and also contained a local information server and index server, which itself was subordinate to 12 index servers, one at each client site.

**Table 2.** Resources at Service Sites

| Site Type | Processors at Site | Number of Sites |
|---|---|---|
| 1 | (1) 500-processor cluster | 12 |
| 2 | (2) 500-processor cluster | 6 |
| 3 | (1) 500-processor cluster<br>(2) 1000-processor vector | 6 |
| 4 | (2) 500-processor cluster<br>(1) 5000-processor cluster | 6 |

**Table 3.** Description of Task Types

| Task Type | *pType* | *pFactor* | *pCycles* |
|---|---|---|---|
| T1 | Cluster | 500 | 2.25e6 |
| T2 | Vector | 1000 | 1.005e7 |
| T3 | Cluster | 5000 | 2.50e7 |

**Table 4.** Description of Application Types

| Application Type | Number, Type and Sequencing of Tasks |
|---|---|
| A1 and A2 | (2) of task T1, executed sequentially |
| A3 | (3) of task T1, executed sequentailly |
| A4 | (1) of task T1 followed by (1) of task T3 |
| A5 | (1) of task T2 |

With no comprehensive studies of grid workflows to rely on, we chose relatively basic workflows in order to provide a simple baseline for analysis. We simulated applications consisting of one to three compute-intensive, parallelized tasks (each requiring between 500 and 5000 processors). We selected task types (Table 3) with execution times (average 1.38 hours/task) on the same order as observed in selected processor workload trace studies [30], [31]. Task definitions were combined to form application workflows of five types (Table 4).

We chose to experiment with a grid under moderate (50%) workload so that attacks would generate stress. Our attack model would not stress a lightly loaded grid, while a heavily loaded grid would already be operating under stress. We required each client site to have 25 applications (five instances of each application type), and calibrated simulated processor speeds to ensure these 300 applications consumed 50% of the capacity shown in Table 2. We activated service providers and applications after a random initial delay. Applications started after a further random delay (up to 2 hours) to simulate start of a workday. We assumed users requiring applications to complete within a 100,000 s deadline (just over one day), but allowed up to 200,000 s for applications to complete in order to measure the extent to which applications exceeded the deadline.

## 5.2 Experiment Design and Metrics

Initially, we considered effects on system performance of an attack carried out through *spoofing* by authorized but malicious service providers. Spoofing occurs when a bogus service factory at a miscreant site returns a faked template showing all the site's grid processors without reservations, leading a client negotiator to assume its task can be run immediately. This causes client negotiators to submit offers to the bogus service factory, which makes no further response, thus denying service to the client. Consequently, spoofed client negotiators time out (after 30 s) and terminate. Spoofing causes clients to lose time pursuing bogus resources, delaying application completion. We were interested to see how our simulated grid responded to this threat, and especially to discern performance differences between the two negotiation strategies. To assist clients, we defined an *adaptive failure-response* behavior in which clients react to negative feedback – service factories that caused timeouts were not retried for 5000 s, and after three consecutive timeouts a service factory was not retried for a specific task.

We configured our clients so that half negotiated under SRR and half under MRR. We subjected this configuration to three scenarios: (1) normal conditions (50% workload, no spoofing), (2) spoofing without failure response, and (c) spoofing with failure response. For each scenario, we executed repetitions of a simulated workday. Spoofing sites, chosen randomly with probability ½, remained so for the duration of a repetition. On average, 15 of 30 sites were spoofing in any repetition, eliminating half the system capacity and driving workload to 100%. Both spoofing scenarios were subjected to an identical sequence of 545 randomly generated topologies.

We measured two main aspects of system performance: application completion time ($T_c$) and overhead. We recorded frequency distributions (interval 10,000 s) for $T_c$ across all topologies for each combination of scenario and negotiation strategy, and used those distributions to compute probability density functions (PDFs). For applications completing by goal $T_g$ (= 100,000s), we computed average application duration, $\overline{D}_{app}$, as a proportion of $T_g$:

$$\overline{D}_{app} = \Big( \sum_{i=1}^{N_{app}} (T_c - T_d)_i / T_g \Big) / N_{app}, \quad \left[T_C \le T_g\right] \tag{10}$$

where $T_d$ denotes time the discovery process commenced for application $i$, $T_c$ is completion time for application $i$, and $N_{app}$ is number of applications (with $T_c \le T_g$) in the experiment. We denote the average application duration across all repetitions as $\overline{\overline{D}}_{app}$. We also computed P($T_c \le T_g$), probability an application completes by $T_g$.

We measured overhead based on messages transmitted to complete negotiation. (Note that in these experiments we allowed up to five active client negotiators for each application task.) We defined as the minimum number of messages, $X_{ngt}$ = 25, the expected minimum when five client negotiators activate simultaneously for a task using the SRR negotiation strategy. We computed the average $O_{ngt}$ for a task as

$$\overline{O}_{ngt} = \sum_{i=1}^{N_{task}} \left[ (M_{ngt})_i / X_{ngt} \right] / N_{task}, \tag{11}$$

where $M_{ngt}$ is the number of negotiation messages counted to obtain agreements for task $i$ and $N_{task}$ is the number of tasks. $\overline{\overline{O}}_{ngt}$ denotes $\overline{O}_{ngt}$ averaged over all repetitions.

## 6 Results and Discussion

Table 5 summarizes system performance. Spoofing caused application duration to increase by 30% and $P(T_c \le T_g)$ to fall by 15%. Spoofing also caused negotiation overhead to increase fifteen times. However, incorporating adaptive failure response to combat spoofing decreased this overhead by about 50%, due to fewer interactions with spoofing sites. Unexpectedly, though, incorporating failure response caused an increase in application duration and a decrease in the probability of completing applications by $T_g$. This surprising result is supported by the probability density functions (PDFs) for application-completion times, plotted in Figure 2 for three scenarios: (a) no spoofing, (b) spoofing without failure response, and (c) spoofing with failure response. As expected, a large number of applications completed later when spoofing occurred, as client negotiators lost time making offers to bogus sites before eventually reaching legitimate sites. Unexpectedly, adaptive failure response led to a distinct right-shift in the PDF, as more applications completed later. This puzzling result required further investigation. To understand and explain the unexpected outcome, we identified topologies that exhibited the greatest performance degradation when using adaptive failure response. Then we selected one of those topologies for multidimensional analyses, using the approach outlined earlier in Section 3.

**Table 5.** Summary of system performance

| | Application Duration $\overline{\overline{D}}_{app}$. | | | $P(T_c \leq T_g)$ | | | Negotiation Overhead $\overline{\overline{O}}_{ngt}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Total | SRR | MRR | Total | SRR | MRR | Total | SRR | MRR |
| No Spoofing | 0.355 | 0.363 | 0.348 | 1.000 | 1.000 | 1.000 | 2.02 | 1.15 | 2.89 |
| Spoofing without failure response | 0.660 | 0.728 | 0.526 | 0.845 | 0.728 | 0.963 | 30.74 | 40.09 | 22.05 |
| Spoofing with failure response | 0.712 | 0.816 | 0.612 | 0.800 | 0.711 | 0.890 | 17.46 | 18.90 | 16.11 |

## 6.1 Multidiminsional Analysis

Let failure-response behavior be $s \in \{NoFR, FR\}$, where *NoFR* signifies no failure response and *FR* signifies failure response, and let event types of interest be $e \in \{RC, TC\}$, where *RC* denotes reservations created and *TC* denotes tasks completed. We first examined effects of *s* on *RC*, instantiating subspace $V_1$ (equation 2) by choosing $e_x$=*RC*. We defined equivalence classes using equation 3, selecting observation interval *I*=1,000 s (the smallest granularity we could plot conveniently) over *T*=200,000 s to yield 200 equivalence classes $1 \leq i \leq 200$, summed across failure-response behavior *s* and job class *j*. Each equivalence class was then restricted to *s* = *FR* or *s* = *NoFR* and aggregated (equation 4) to generate two time series, as shown in Figure 3, revealing two different patterns of reservation creation. Most notably, Figure 3 shows, at *i* = 100 (*t* = 100,000 s), a large spike in reservations created appeared when failure response was employed. Figure 3 also shows that over the interval $1 \leq i \leq 50$ more reservations were created when failure response was used – while for the interval $51 \leq i \leq 100$, no reservations were created when failure response was used. These large shifts in the pattern of reservation creation suggested to us that the schedule of job executions was being altered.

To examine this at finer resolution, we defined additional subspaces to consider task completions ($e_x$=*TC*) under two negotiation strategies $a \in \{SRR, MRR\}$ for particular job classes $j \in \{A1T1, A1T2, A3T1, A3T2, A3T3\}$ chosen by selecting task types from Table 3 comprising application types A1 and A3 from Table 4. Using the approach shown in equation 5, we defined a subspace, $V_2$, to to consider the tasks for application A1. We then defined a relation, $R_1$, (similar to equation 6) to form eight equivalence classes $\{Q_k\}$, $k = 1 \ldots 8$, on $V_2$, with each partition representing a combination of failure-response behavior (*s*), job class (*j*), and negotiation strategy (*a*). Using equation 7, we determined a scaling factor, *f*, and operators as defined in equation 8 for each partition. Applying these operators yielded eight time series, shown in Figure 4 (a) and (b). Similarly, we defined another subspace, $V_3$, to which we applied $R_1$ to form 12 equivalence classes. We then determined a scaling factor and operators to yield 12 time series, shown in Figure 4 (c) and (d).

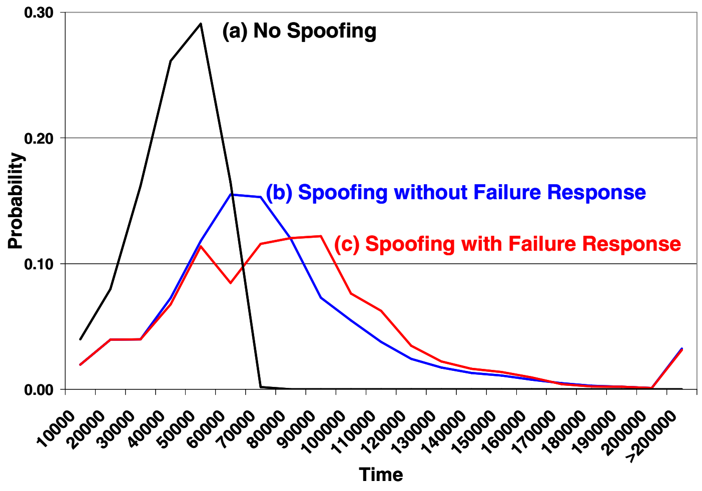


**Fig. 2.** Comparative PDFs for Application-Completion Times given: (a) No Spoofing, (b) Spoofing without Failure Response, and (c) Spoofing with Failure Response

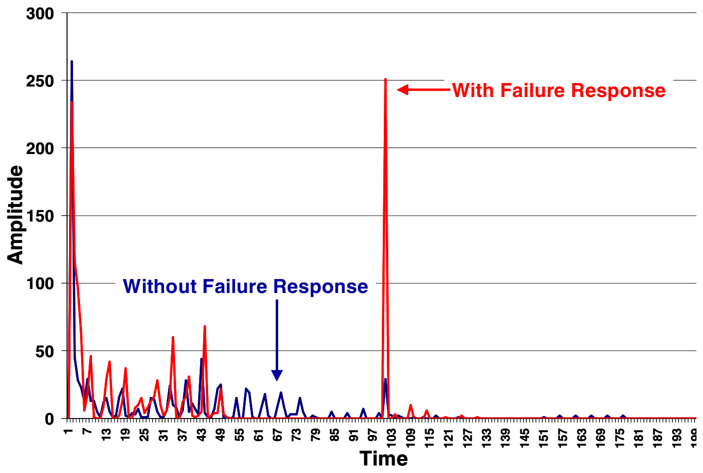


**Fig. 3.** Two Time Series: (a) Reservations Created without Failure Response and (b) Reservations Created with Failure Response

The 20 time series shown in Figure 4 reveal shifts in the temporal evolution of job completions. Specifically, when failure response was employed for clients using SRR, initial tasks for both applications (A1T1 and A3T1) scheduled and completed earlier than when failure response was not used, while their second tasks (A1T2 and A3T2) completed in roughly the same time range irrespective of failure-response behavior. Third tasks (A3T3) for clients using SRR were completed later when failure response was used. Employing adaptive failure response helped clients using SRR become more competitive in obtaining reservations earlier in time. As Figures 4(b) and 4(d) show, this led clients using MRR to have more difficulty obtaining early reservations for second (e.g., A1T2 and A3T2) and third (A3T3) tasks within an application. Thus, while the initial tasks for clients using MRR completed within the same time range regardless of whether adaptive failure response was used, second and third tasks were delayed when failure response was activated.

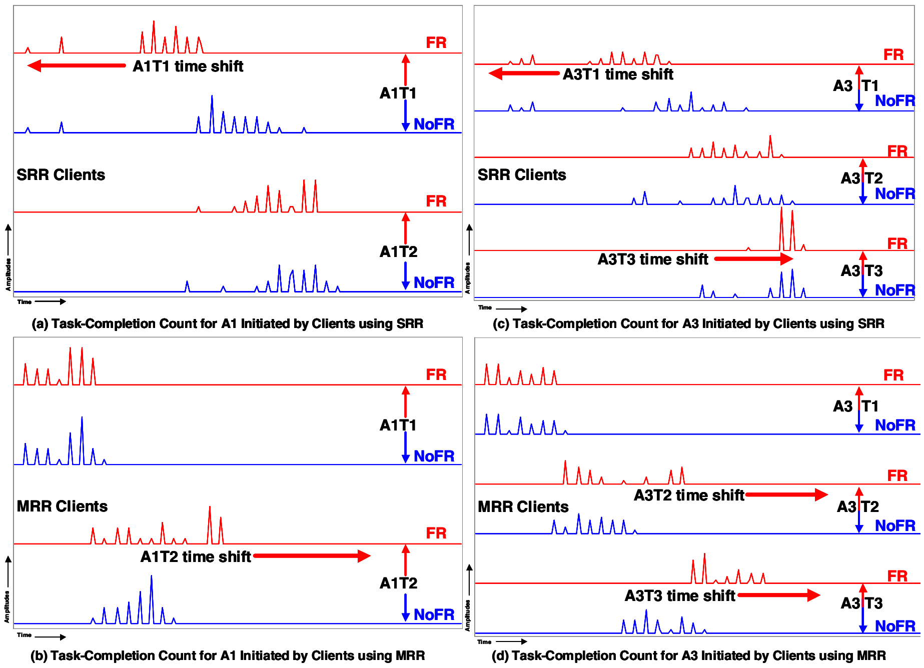


**Fig. 4.** Time Series of the Count of Task Completions Showing Time Shifts in the Execution Schedule When Adaptive Failure Response is used to Combat Spoofing Attack

## 6.2 Self-organization and Adaptive Failure Response

This result demonstrates that taking a seemingly sensible action, here having clients resist spoofing, can lead to unanticipated system-wide performance degradation. In our example, introducing the adaptive failure response feedback mechanism (recall sec. 5.2) helped clients using SRR negotiation to become more competitive. This led to an unintentional reordering in the global schedule so that many second and third tasks executed later, and completion of jobs was delayed for most clients. This reordering arises from a self-organizing process in which numerous feedback-driven interactions among independent actors during negotiation form the global schedule. Multidimensional analysis allowed us to observe and explain this phenomenon.

We explored this behavior under several different conditions, as described in detail elsewhere [32]. First, we altered the scheduling algorithm to permit applications to reserve resources early (optimistically) for all tasks, rather than waiting until a previous task completed before seeking resources for the next task. Here, spoofing again caused performance degradation. Use of adaptive failure response caused further decline, with SRR clients benefiting at the expense of MRR clients. Multidimensional analysis of task completions revealed that when failure response was used, the global job execution schedule was again reordered so that second and third tasks executed later. Next, we introduced a different mix of applications and tasks, maintaining the basic workflow. Again, spoofing caused performance to degrade substantially, while use of failure response caused a slight degradation in which SRR clients benefited while MRR clients suffered. Multidimensional analysis of task completions revealed a similar task-reordering phenomenon to that observed with the original job mix. Finally, we defined a workload where every application consisted of only a single task, thus removing task dependencies in multi-task applications. While spoofing again increased application completion times, the introduction of failure response had little effect on the relative performance of MRR and SRR clients. Nevertheless, multidimensional analysis of task completions revealed underlying shifts in completion times, albeit in muted form. Thus, under the various conditions we examined, adaptive failure response either further degraded or failed to improve application-completion times, and a comparable reordering of job completions was observed in each case.

## 7 Conclusions

Scheduling and execution of jobs in a grid can exhibit a self-organizing behavior arising from distributed resource-allocation protocols. In the cases we studied, this self-organizing behavior leads to unexpected increase in application-completion times when adaptive failure response is used to combat a spoofing attack. Without tools such as multi-dimensional analysis to explore global behavior, much time and expense might be wasted attempting to identify and explain causes of this unexpected system performance. We believe that the key to understanding and controlling large, distributed systems is to view processes, such as distributed-resource allocation, as self-organizing. Doing so could provide a basis for creating measurement techniques and control algorithms to manage distributed systems of scale and complexity. Otherwise, with inadequate analytic tools, unanticipated consequences of underlying self-organizing processes will likely hamper adoption of promising technologies, such as grid computing.

In this paper, we explored a limited model of a computing grid that uses simple processes to allocate distributed resources among applications with basic workflows. Numerous researchers have proposed more complex regimes (such as market-based approaches) for distributed resource allocation, and we expect future proposed grid standards to include more nuanced algorithms, which may lead to other unexpected global behaviors caused by underlying self-organizing processes. Given this, we plan to explore behaviors that might arise if more sophisticated approaches are deployed. Subsequently, we will investigate techniques for influencing global behaviors in distributed systems.

## References

1. The WS Resource Framework, V1.0. Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago (2004)
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration. Global Grid Forum (June 2002)
3. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing. San Francisco (August 7-9, 2001) 55-67
4. Holbrook, M.B.: Adventures in Complexity: An Essay on Dynamic Open Complex Adaptive Systems, Butterfly Effects, Self-Organizing Order, Coevolution, the Ecological Perspective, Fitness Landscapes, Market Spaces, Emergent Beauty at the Edge of Chaos, and All That Jazz. Academy of Marketing Science Review (2003)
5. Web Services Architecture. W3C Working Group Note (February 11, 2004)
6. The Open Grid Services Architecture, Version 1.5. Global Grid Forum (March 10, 2006)
7. I. Foster et al.: A Globus Primer or, Everything You Wanted To Know About Globus But Were Afraid to Ask, an Early and Incomplete Draft (May 8, 2005)
8. Bak, P.: How Nature Works: the science of self-organized criticality. Copernicus, New York (1996)
9. Legrand, A., Marchal, L., Casanova, H.: Scheduling Distributed Applications: The SimGrid Simulation Framework. Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03). Tokyo (May 12-15, 2003) 138-145
10. Buyya, R. Murshed, M.: GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid Computing. Concurrency and Computation: Practice and Experience. Vol. 14. (2002) 1175-1220
11. Liu, X., Xia, H., Chien, A.: Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics. Journal of Grid Computing. Vol. 2, No. 2 (2004) 141–161
12. Ernemann, C., Hamscher, V., Yahyapour, R.: Benefits of Global Grid Computing for Job Scheduling. Proceedings of the Fifth IEEE International Workshop on Grid Computing (GRID 2004). Pittsburgh. (November 8, 2004) 374-379.
13. Wolski, R., Brevik, J., Plank, J., Bryan, T.: Grid Resource Allocation and Control Using Computational Economies. In Berman, F, Fox, G., Hey, T. (eds.): Grid Computing: Making the Global Infrastructure a Reality. Wiley and Sons, New York. (2003) 747–772
14. Gomoluch, J., Schroeder, M.: Market-based Resource Allocation for Grid Computing: A Model and Simulation. Proceedings of the First International Workshop on Middleware for Grid Computing. Rio de Janeiro. (June 16-20, 2003) 211-218
15. Yeo, C.S., Buyya, R.: Service Level Agreement based Allocation of Cluster Resources: Handling Penalty to Enhance Utility. Proceedings of the 7th IEEE International Conference on Cluster Computing. Boston. (September 27-30, 2005)
16. In, J., Avery, P., Cavanaugh, R., Ranka, S.: Policy Based Scheduling for Simple Quality of Service in Grid Computing. Proceedings of the Eighteenth International Parallel and Distributed Processing Symposium (IPDPS'04). Santa Fe. (April 26-30, 2004) 23
17. He, X., Sun, X., Von Laszewski, G.: A QoS Guided Scheduling Algorithm for Grid Computting. Journal of Computer Science and Technology, Special Issue on Grid Computing. Vol. 18, No. 4 (2003) 442-450

18. Cooper, K., et al.: New Grid Scheduling and Rescheduling Methods in the GrADS Project. Proceedings of the Eighteenth International Parallel and Distributed Processing Symposium (IPDPS'04). Santa Fe. (April 26-30, 2004) 199
19. Krothapalli, N. Deshmukh, A.: Dynamic allocation of communicating tasks in computational grids. IIE Transactions. Vol. 36, No. 11. (2004) 1037-1053
20. Chen, H. Maheswaran, M.: Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems. Proceedings of the Sixteenth International Parallel and Distributed Processing Symposium (IPDPS 2002). Fort Lauderdale (April 15-19, 2002)
21. Subramani, V., Kettimuthu, R., Srinivasan, S., Sadayappan, P.: Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-11 '02). Edinburgh (July 24-26, 2002) 359
22. SOAP V1.2 Part 1: Messaging Framework. W3C Recommendation (June 24, 2003)
23. WS Addressing. BEA Systems Inc., International Business Machines Corporation, and Microsoft Corporation, Inc. (March, 2004)
24. WS Resource Lifetime, V1.1. Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago (March, 2004)
25. Publish-Subscribe Notification for Web Services, V1.0. Akamai Technologies, Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation, SAP AG, Sonic Software Corporation, Tibco Software Inc. and The University of Chicago (March 2004)
26. WS Services Topics, V1.0. Akamai Technologies, Computer Associates International, Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation, SAP AG, Sonic Software Corporation, Tibco Software Inc. and The University of Chicago (March, 2004)
27. WS Service Group, V1.0. Computer Associates International Inc., Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and The University of Chicago (March, 2004)
28. Distributed Resource Management Application API Specification 1.0. Global Grid Forum (June, 2004)
29. Web Services Agreement Specification (WS-Agreement). Global Grid Forum (September, 2005)
30. Parallel Workloads Archive. The Hebrew University of Jerusalem. http://www.cs.huji.ac.il/labs/parallel/workload/
31. Shan, H., Oliker, L.: Job Superscheduler Architecture and Performance in Computational Grid Environments. Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. Phoenix (November 15-21, 2003) 44
32. Mills, K., Dabrowski, C.: Investigating Global Behavior in Computing Grids: the Extended Report. Draft technical report. U.S. National Institute of Standards and Technology (Available from the authors).

# Using Decentralized Clustering for Task Allocation in Networks with Reconfigurable Helper Units

Daniel Merkle, Martin Middendorf, and Alexander Scheidler

Parallel Computing and Complex Systems Group
Department of Computer Science
University of Leipzig
Augustusplatz 10-11
D-04109 Leipzig, Germany
{merkle, middendorf, scheidler}@informatik.uni-leipzig.de

**Abstract.** Computing systems are studied that consist of many (partially) autonomous workers and helpers which are connected via a network and where the helpers perform service tasks for the workers. In order to execute different service tasks the helpers have reconfigurable hardware. We address the problem to design a decentralized system where the requests of the workers are executed by suitable helpers and where the total reconfiguration costs of the helpers are small. A system is proposed that uses a combination of a fully decentralized and dynamic clustering algorithm and a self-organized task allocation system. The clustering algorithm is used to classify the service requests that are sent as packets through the network in order to give the helpers hints which packets are suitable to be executed by them. Simulations have been done for static and dynamic scenarios where we investigate the reconfiguration costs and the number of dropped packets, i.e., requests that could not be satisfied. The results show that the proposed system has a strong adaptive behavior and that the decentralized clustering is able to reduce the reconfiguration costs significantly.

## 1 Introduction

In this paper we consider computing systems that consist of many (partially) autonomous components called workers. From time to time the workers need service of different types. These requirements for service might occur more or less regularly and are expected by the workers or the requests are unexpected due to failures and unforseen by the workers. The service tasks are done by helper components of the computing system. It is assumed that the computing system follows the paradigm of autonomic computing (e.g., [2]) or organic computing (e.g., [1]). This means that ideally there does not exists any central control and the connections or relations between the components of the system are flexible. Moreover the components itself might be flexible, e.g. they might reconfigure themselves according to the needs of the environment or the user(s) of the system. Here we assume that each helper can potentially perform all the different service tasks. But for each type of service task a helper might have to reconfigure itself in order to provide the resource requirements that are necessary to perform the service task. It is assumed that each worker knows what type of service it actually needs. Therefore the worker can include information about the type of service it needs into its requests for service.

Worker-helper scenarios have already been studied in the literature. But to consider reconfigurable helpers in such a scenario is a new aspect that is interesting for organic computing systems. Some initial work in this area has been done in [5]. However, in this work it was assumed that the workers choose randomly a helper to ask for service. But in real systems which have a decentralized organisation the workers and helpers might be connected via a network without directly knowing from each other.

In this paper we address the problem how to organize the worker helper system in a decentralized way so that the service requests of the workers that are executed by a helper are suitable for this particular helper. A helper would not like to satisfy just any service request because it might need much reconfiguration before it can start performing the service tasks. When reconfiguration costs are high there is the danger of an inefficient execution of the service tasks. Because the helpers do not have a global view of the system they can not just chose a service task that is suitable for them and can be done with only a small amount of reconfiguration. When a helper is too selective some service tasks might never been taken by any helper. But if a helper is not selective enough then its work becomes inefficient.

The approach that we take in this paper is to use a decentralized clustering scheme that clusters the service packets that are sent through a network. Such a clustering scheme for packets in a network which is executed with the help of the routers in the network has been proposed in [3,4]. For a general overview over decentralized clustering methods that have been proposed in the literature see [4]. The clustering scheme of [3,4] is used here to classify the service packets according to their type so that packets for services that have similar hardware requirements are put into the same cluster. Each helper can then specialize to service requests from one cluster. Then, preferably it performs only service requests from "its own" cluster. Since the hardware resources that are needed for the service tasks within one cluster are similar this will lead to small reconfiguration costs. Several problems and questions that emerge for such a system are addressed here. For example, the helpers should not specialize all to the same cluster. Can the system handle situations where the classes of service tasks, that are needed to be done, changes? How long should a service request packet be sent around in the network before it might be dropped (in that case the worker might be informed so that it can decide whether to sent another service packet).

The rest of the paper is organized as follows. In Section 2 the decentralized algorithm that is used for clustering in networks is presented. The model of the computing system and the self-organized task allocation scheme is introduced in Section 3. The combination of the clustering scheme with the self-organized tasks allocation is described in Section 4. Experimental results are presented in Section 5. Conclusions are given in Section 6.

## 2 Decentralized Packet Clustering in Networks

In [3,4] decentralized packet clustering algorithms in networks were investigated for static and dynamic situations (i.e., the data vectors used for the clustering change over time). The corresponding clustering problem - called Decentralized Packet Clustering Problem (DPC) - is to find for a set of packets $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ that are sent through the network consiting of routers a good clustering, i.e., a partitioning $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of $\mathcal{P}$. Each packet $P_i \in \mathcal{P}$ contains a data vector $v_i$ that is used for the clustering. Instances of the DPC problem differ with respect to network topology, number of routers, and how packets are sent through the network. In the dynamic version called d-DPC the data vector of a packet can change at every time step.

A successful and yet simple algorithm to solve the DPC problem is algorithm d-DPClust$_{zc}$ - simply called d-DPClust in the following. For this algorithm each packet $P_i = (v_i, c_i)$ has a data vector $v_i$ and a cluster number $c_i$. Each router $r$ stores a vector of estimated centroids $\mathcal{Z}_r = (z_r^1, \ldots, z_r^{|\mathcal{C}|})$. When a packet $P_i$ arrives in a router the centroid estimation for cluster $c_i$ is modified according to $z_r^{c_i} = (1-\beta) \cdot z_r^{c_i} + \beta \cdot v_i$, where $\beta$, $0 < \beta < 1$ is a parameter of the algorithm. Thus, $\beta$ determines the share that the data vector $v_i$ has of the new centroid estimation. Then, the new cluster number for packet $P_i$ is determined by using the distances of its data vector $v_i$ to the estimated centroids $z_r^j$, $j = 1, \ldots, |\mathcal{C}|$ that are stored in the router. The packet is assigned to the cluster for which this distance is minimal, i.e. $c_i = \mathrm{argmin}_j ||v_i - z_r^j||$.

In [3] the algorithm d-DPClust was compared with the classical (centralized) $k$-means algorithm with respect to several clustering validity measures (e.g., the Silhouette coefficient [6] and the Dunn index [7]). The comparison was executed for static and dynamic problem instances of the DPC problem. The algorithm d-DPClust was shown to have a good clustering behavior. It was particularly successful with regard to its adaptive behavior in situations with dynamically changing data vectors.

## 3 Computing System and Self-organized Task Allocation

Our model of a computing system is as follows. The system consists of two types of components or members, ordinary members called workers and supporting members called helpers. Each helper has reconfigurable hardware (e.g., a Field Programmable Gate Array (FPGA)) on which it performs the service tasks for the workers. As a model for the reconfigurable hardware we assume that it consists of $q$ slices which can be reconfigured independently from each other. Each slice is always configured so that it works in one of several modes. Different slices can be configured in different modes. A reconfiguration operation has the effect that the mode of some slices is changed. During a reconfiguration operation helper can reconfigure any number of its slices.

In [5] a two helper computing system was analyzed analytically for situations where the request rates for service tasks do not change and the helpers are not reconfigured. Empirically computing systems with many components (helpers and workers) were investigated. Different strategies for the behaviour of the helpers were studied, e.g., when they should accept a service task and when they should reconfigure themselves.

## 4 Decentralized Task Allocation in Networks

In this paper we propose a combination of a decentralized clustering algorithm and a self-organized task allocation system to organize a worker-helper mechanism via the network of a computing system. The network consists of routers and helper nodes. Each helper has reconfigurable hardware in order to execute different types of service tasks efficiently. The switch from one task to another leads to reconfiguration costs, which are higher, the more the tasks differ in their demands for hardware resource. During each time step several service requests are created. To each service request corresponds a service packet that is sent into the network. The resource demands for the corresponding service tasks are specified in the packet. The number of service packets that are created per time step is called the (packet) arrival rate.

The problem is to execute as many service tasks as possible and also to have small reconfiguration costs for the helpers. Service tasks can only be executed by the helpers. The helpers are allowed to reject service tasks. For each service request packet the number of helpers that have been visited by the packet (number of hops) is counted. If a packet is rejected by a helper this counter is increased by one. If the counter of a packet exceeds a threshold value TTL (time to life) the service request packet is dropped. The fraction of dropped packets (in relation to all packets) is called the (packet) drop rate. The main idea is to group the service tasks into clusters which have similar hardware resource demands and to let the helpers specialize by allowing them to execute only the service tasks from one of that clusters. Since all tasks in one cluster are similar with respect to their resource demands and therefore also with respect to the helper configuration that is needed, the reconfiguration costs for the helpers of switching between these tasks are relatively small.

To group the service requests the decentralized clustering algorithm d-DPClust is used. Each service request packet is thus characterized by a vector $(v_i, c_i)$ where $c_i \in \{1, \ldots, n_c\}$ is the cluster number and $v_i$ is a vector that describes the hardware resources needed for the corresponding service task, i.e., the helper configuration that is required to execute the service task. In this paper it is assumed that $v_i$ is a three dimensional vector, i.e. $v_i = (v^1, v^2, 1 - v^1 - v^2) \in [0, 1]^3, v^1 + v^2 \leq 1$. If a service task with data vector $(v^1, v^2, 1 - v^1 - v^2)$ is executed by a helper, then this helper node has to be reconfigured so that the

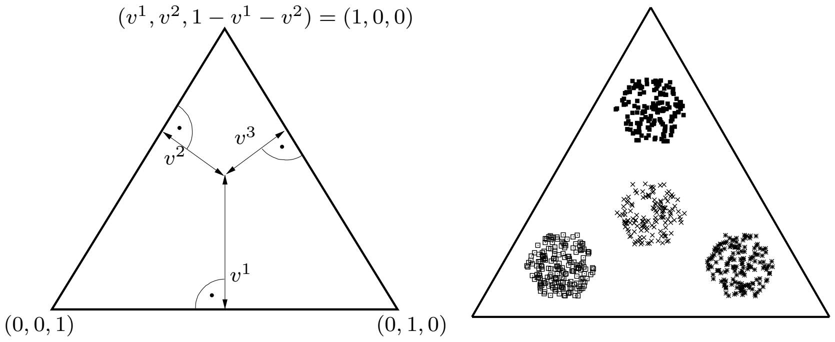


**Fig. 1.** The hardware resource requirements of a service request $(v^1, v^2, v^3)$ with $v^3 := 1 - v^1 - v^2$ are depicted within an equilateral triangle with height 1 (left); hardware resource requirements of service requests of four different classes (right)

fraction $v^1$ (respectively $v^2$ and $1 - v^1 - v^2$) of its slices is configured in mode 1 (respectively mode 2 and mode 3).

Each helper node in the network has an associated cluster number. If the cluster number of a service request packet that is received by a helper is identical to its cluster number, then the helper is reconfigured so that is satisfies the resource demands as specified in the packet. The service task is executed by the helper and the packet is deleted (this is done within one simulation time step). Otherwise, i.e., if the helpers cluster number is different from the packets cluster number, the helper might change its cluster number to the cluster number of the packet. The parameter $p$ determines the probability that this happens. If the helper does not change its cluster number to the cluster number of the arriving service request, then the service packet is rejected and sent to another node in the network. Note, that a service request is also rejected if the helper is already executing another service request at the same simulation time step.

The configuration of a helper (corresponding to the a hardware demand of the actual or last service task that is has executed) can be visualized as a point in an equilateral triangle with height 1 (see left part of Figure 1). Note, that for every point in the triangle the sum of the distances to the right, bottom, and left line is 1. Let the distance from the bottom (respectively left and right) line equal $v^1$ (respectively $v^2$ and $1 - v^1 - v^2$). If a helper is configured according to $(v^1, v^2, 1 - v^1 - v^2)$ and has to be reconfigured according to $(w^1, w^2, 1 - w^1 - w^2)$, then the costs of this reconfiguration are

$$\max(|w^1 - v^1|, |w^2 - v^2|, |(1 - w^2 - w^1) - (1 - v^2 - v^1)|). \qquad (1)$$

Recall, that in contrast to the self-organized task allocation system as presented in Section 3, it is assumed here that a helper that executes a service task $i$ reconfigures itself so that its configuration equals exactly the resource demand that is specified by the vector $v_i$ of the corresponding service packet. Furthermore, it is assumed that the execution of a task always takes one simulation time step.

## 5 Experiments

All test runs were performed in a scenario, where all service requests are from up to four different classes. A snapshot from a typical test scenario is depicted in the right part of Figure 1. Note, that a partitioning of the service request that leads to small costs is not given in advance, as packets have a random cluster identity when they are created. Within each class of service requests the individual requests are chosen as follows. Let $(c_j^1, c_j^2, 1 - c_j^1 - c_j^2)$ be the center of request class $j, j \in \{1, 2, 3, 4\}$. Then for a new service request with configuration request vector $(v_i^1, v_i^2, 1 - v_i^1 - v_i^2)$ the value $v_i^1$ (respectively $v_i^2$) is chosen randomly from the interval $[c_j^1 - 0.1, c_j^1 + 0.1]$ (respectively $[c_j^2 - 0.1, c_j^2 + 0.1]$). Requests for which $1 - v_i^1 - v_i^2$ is not in the interval $[0.9 - c_j^1 - c_j^2, 1.1 - c_j^1 - c_j^2]$ are discarded. The center of request class 1 is $(1/3, 1/3, 1/3)$, the center of classes 2 (resp. 3 and 4) are $(2/3, 1/6, 1/6)$ (respectively $(1/6, 2/3, 1/6)$ and $(1/6, 1/6, 2/3)$). If not stated otherwise in each simulation time step 50 packets with service requests were sent into the network. 50 helpers units and 50 routers were used. The probability $p$ that a helper changes its cluster was set to 0.01 (if not stated otherwise). Parameter $\beta$ which influences the update of the centroid estimation in a router was set to 0.1. If a centroid estimation has not changed by the last 100 packets that arrived at a router, the new centroid estimation is set to the corresponding configuration of the next arriving packet. Each result that is given in the following is averaged over 10 simulation runs, i.e., each pair of cost/drop values is an average calculated from 10 independent simulations. Each simulation was performed over 10000 steps. The reconfiguration costs that are given in the figures are the overall reconfiguration costs that were spent by the helpers when executing the the service requests (comp. Equation 1) divided by the number of all service requests, that have been created during a simulation run.

### 5.1 Number of Clusters

We investigated the drop rate of the service request packets and the reconfiguration costs for test runs with different number of service request classes. Using 1 request class (class 1), 2 request classes (classes 3 and 4), or 4 request classes (all classes) the number $n_c$ of clusters that are used by the decentralized clustering algorithm have been varied with $n_c \in \{1, 2, \ldots, 10\}$ (comp. Figure 2(a), 2(c), and 2(e)). The results are depicted in the left column of Figure 2 when using TTL$\in \{1, 5, 10, 50\}$. There is a clear trade-off between the drop rate and the reconfiguration costs. When using a larger TTL value, the drop rate is reduced significantly. The reduction of the reconfiguration costs for increasing number of clusters $n_c$ depends strongly on the number of request classes. When 2 or 4 request classes are used there is a sharp bend in the corresponding curves, as the algorithm utilizes its adaptability. When $n_c$ is smaller than the number of request classes, then some helpers have to execute service request of more than one class. This leads to relatively high reconfiguration costs as can be seen in Figures 2(c) and 2(e), where packets from 2 or 4 service request classes were put into the network. For example, when 2 request classes and TTL 5 are used, the costs are reduced

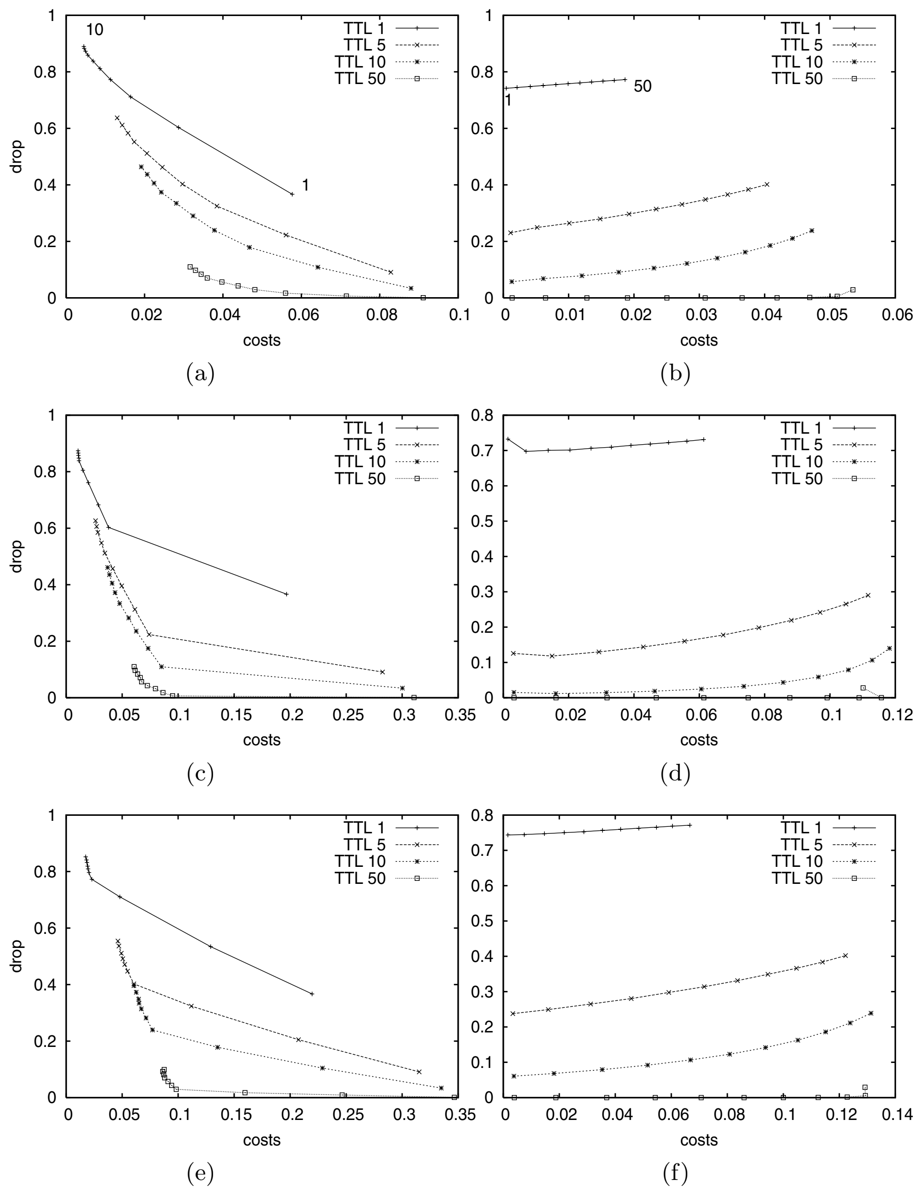


**Fig. 2.** Drop rate/reconfiguration cost trade-off for different scenarios; left column: dots on lines correspond to number of clusters $n_c \in \{1, \ldots, 10\}$ (from left to right); numbers at dots indicate number of clusters used; right column: dots on lines correspond to arrival rates of $\{5, 10, \ldots, 50\}$ (from left to right) packets per simulation step; numbers at dots indicate arrival rate; number of service request classes: 1 (top), 2 (middle), 4 (bottom)

from 0.28 when using $n_c = 1$ to 0.07 when $n_c = 2$ is used. A further increase of $n_c$ (larger than the number of service request classes) reduces the costs only slightly. The small reduction results from the fact that the service requests within one class vary slightly with respect to their resource requirements. Therefore the reconfiguration costs of the helpers can be reduced slightly when the service requests of one class are split into several clusters. The disadvantage is that the packet drop rate increases with a higher number of cluster.

## 5.2 Work Load

In the following we compare simulations where the computing system has different work loads. This was done by using different arrival rates. The arrival rate was set to $\{1, 5, 10, 15, \ldots, 50\}$. The number of clusters for the decentralized clustering algorithm was set to $n_c = 4$ and similar to Subsection 5.1 the number of service request classes was 1,2, or 4. The results are depicted in in the right column of Figure 2.

Obviously, when using a very small (and unrealistic) value of TTL= 1 the drop rate of the packets is very high (always larger than 0.69). But this value is interesting because it shows the average fraction of packets that are not executed by a single helper. The small number of service requests that are executed produce only small reconfiguration costs. When using a higher TTL the drop rate goes down significantly, e.g. for TTL=5 it is less than 0.3 is all cases. The increase in reconfiguration costs is relatively small in this case (less than 0.13 when using 4 service request classes and an arrival rate of 10). When the value of TTL is 50 nearly no packets are dropped in all the investigated scenarios. Also the reconfiguration costs are small in this case (always $< 0.13$).

## 5.3 Changing Cluster Number of Helper Units

A strong influence on the adaptability of the helpers has the parameter $p$, which is the probability that a helper changes its cluster number when an arriving packet has a different cluster number. When $p$ becomes larger the number of rejected packets decreases and the reconfiguration costs increase. Note, that when using $p = 1$ no arriving service request is rejected by a helper due to its cluster identity (only when the helper is executing another service request a packet is rejected). Rejecting a large number of packets leads to an increase of the drop rate. Drop rates and reconfiguration costs were measured when using a cluster changing probability of $p \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$. The results are depicted in Figure 3 for TTL values of 1, 5, 10, and 50. The smaller the TTL values are the stronger is the decrease of the drop rate with increasing $p$. E.g, when using TTL 5 the drop rate is decreased from 0.4 (for $p = 0.01$) to 0.1 (for $p = 1$). But for high values of $p$ the reconfiguration costs become large (they increase from 0.04 to 0.17 for TTL=5 when $p$ increases from 0.01 to 1).

## 5.4 Dynamically Adding and Removing Request Classes

To show the adaptability of our system a dynamic scenario was investigated where the set of service request classes for which there are packets in the network

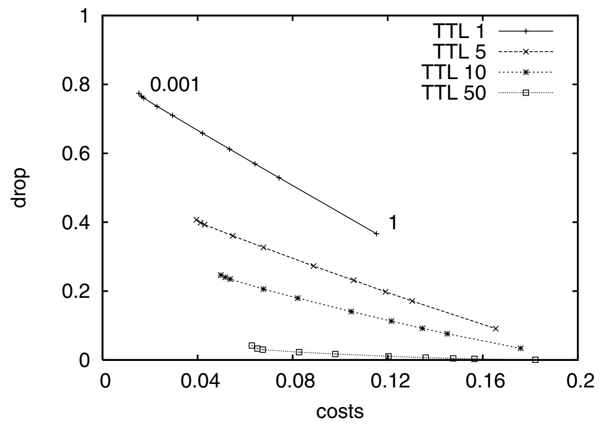


**Fig. 3.** Drop rate/cost trade-off for different probabilities $p$ that a helper changes its cluster when an arriving packet has a different cluster number; $p \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$ (from left to right)

changes. Service classes that are represented by packets in the network are added and deleted during a simulation run. Starting with only one request class (class 1) we successively add classes 2, 3, and 4 every 1000 simulation steps (i.e. packets of the corresponding classes are sent in the network). After that classes 2,3, and 4 were deleted successively every 1000 steps. In Figure 4 the results are depicted for $n_c \in \{1, 2, 4\}$ clusters. When only one cluster is used each additional request class increases the cost significantly, as the helper units have to be reconfigured for different service request classes very often. When using $n_c = 4$ the average reconfiguration costs are much smaller. The additional reconfiguration costs that occur after a new class has been added are due to the fact, that request classes have to be partitioned with less clusters (or are not partitioned at all). This leads to higher intra-class reconfiguration costs. These reconfiguration costs are much smaller than the inter-class reconfiguration costs. Also the drop rate is always very small for $n_c = 4$. This result clearly shows the fast adaptive behavior of the decentralized clustering component based on DPClust.

## 6 Conclusion

We have studied the problem to organize a computing system that consists of worker and helper components which are connected via a network and where the helpers perform service tasks for the workers. The helpers use reconfigurable hardware so that they can execute different service tasks. In order to keep the total reconfiguration costs small the helpers should preferably execute service tasks that need only a small amount of reconfiguration. In order to obtain a decentralized mechanism and to make it suitable for the paradigm of organic computing we have proposed to combine a fully decentralized and dynamic clustering algorithm with a self-organized task allocation system. The clustering algorithm is performed by

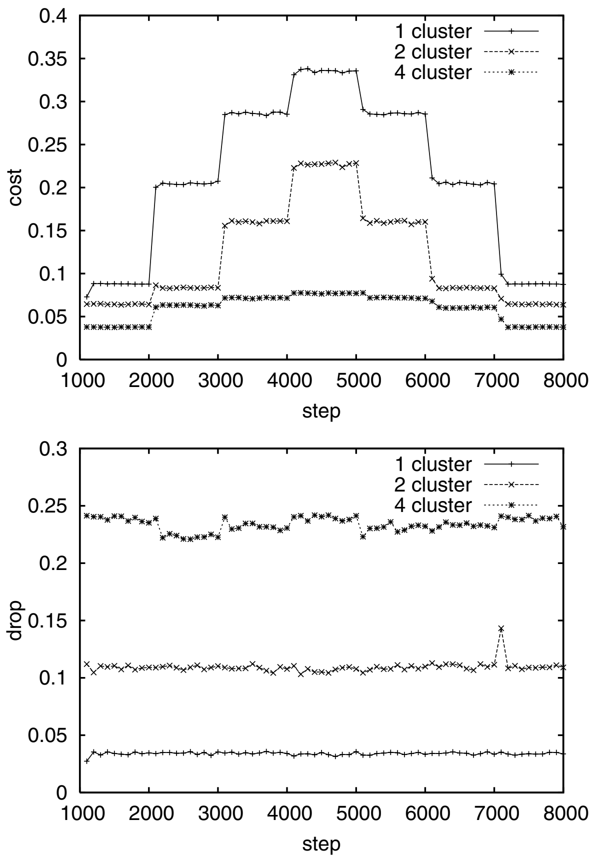


**Fig. 4.** Reconfiguration costs (top) and drop rate of packets (bottom) shown over a simulation run where service request classes are added successively (simulation steps 2000, 3000, and 4000) and then removed (simulation steps 5000, 6000, and 7000); initially (steps 0-999) only one service request class is used; results are given for $n_c \in \{1, 2, 4\}$

the routers in the network and classifies the service requests packets that are sent through the network with respect to their hardware resource requirements (which determine the ideal configuration of the helpers when they execute the service task). Our simulations have shown that the proposed system has a strong adaptive behavior in static and dynamic scenarios and that the decentralized clustering is able to reduce the reconfiguration costs significantly.

## Acknowledgment

This work was supported by the German Research Foundation (DFG) through the project Organisation and Control of Self-Organising Systems in Technical Compounds within SPP 1183.

## References

1. GI: Organic Computing / VDE, ITG, GI - Positionspapier. 2003, `http://www.betriebssysteme.org/Betriebssysteme/FutureTrends/oc-positionspapier.pdf`
2. J.O. Kephart, D.M. Chess: The Vision of Autonomic Computing. IEEE Computer, 36(1): 41-50, 2003.
3. D. Merkle, M. Middendorf and A. Scheidler. Dynamic Decentralized Packet Clustering in Networks. In: Rothlauf, F. et al. (editor): Proc. of the 2nd European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments, LNCS 3449, 574–583, 2005.
4. D. Merkle, M. Middendorf and A. Scheidler. Decentralized Packet Clustering in Router-based Networks. International Journal of Foundations of Computer Science, 16(2): 321-341, 2005.
5. D. Merkle, M. Middendorf and A. Scheidler. Self-Organized Task Allocation for Computing Systems with Reconfigurable Components. Rroc. of the 9th International Workshop on Nature Inspired Distributed Computing (NIDISC'06), 2006, to be published.
6. L. Kaufman, P.J. Rousseuw. Finding Groups in Data: An Introduction to Cluster-Analysis. Wiley, New York, 1990.
7. J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters, J. Cybern., 3(3): 32-57, 1973.

# Self-tuned Refresh Rate in a Swarm Intelligence Path Management System

Poul E. Heegaard[1] and Otto J. Wittner[2]

[1] Telenor R&D[⋆] and Department of Telematics
Norwegian University of Science and Technology, Norway
poulh@item.ntnu.no
[2] Centre for Quantifiable Quality of Service in Communication Systems[⋆⋆]
Norwegian University of Science and Technology, Trondheim, Norway
wittner@q2s.ntnu.no

**Abstract.** CE-ants (Cross Entropy ants) is a distributed, robust and adaptive swarm intelligence system for dealing with path management in communication networks. This paper focuses on strategies for adjusting the overhead generated by the CE-ants as the state of the network changes. The overhead is in terms of number of management packets (ants) generated, and the adjustments are done by controlling the generation rate of ants traversing the network. The self-tuned strategies proposed in this paper detect state changes implicitly by monitoring parameters and ant rates in the management system. Rate adaptation is done both in the network nodes and in the peering points of the virtual paths. The results are promising, and compared to fixed rate strategies the self-tuned strategies show a significant saving (70-85%) in number of packets, and has similar (even slightly better) data packet delay and service availability. The rate adaptation in network nodes provides fast restoration with short path detection times and hence also high service availability. The implicit self-tuned ant rate in the path endpoints improves the convergence time on link state events without flooding the network with management packets in steady state when these are not required.

**Keywords:** Cross Entropy, swarm intelligence, CE-ants, network management, restoration time, service availability.

## 1 Introduction

In the next generation Internet, resource utilisation is likely to become an even more important issue than in today's networks. Paths between source destination pairs should be chosen to ensure an overall good utilisation of the network

[⋆] This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).
[⋆⋆] Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT. http://www.q2s.ntnu.no

resources, and at the same time provide services with high throughput, low loss and low latency. The available spare capacity in the network must be utilised in such a manner that a failure results in a minimum disturbance in traffic flows. To ensure overall system robustness and stay in-line with "Internet philosophy", management systems for path discovery, setup and release, i.e. path management (today realised by combining protocols like MPLS, RSVP, OSPF, BGP etc.), should be truly *distributed* and *adaptive*. Proper path management requires that: a) the set of operational paths should be continuously updated as the traffic load changes, b) new paths should become almost immediately available between communication nodes when established paths are affected by failures, and c) new or repaired network elements should be put into operation without unnecessary delays. Near immediate and robust fault handling advocates distributed local decision-making on how to deal with failures. However, one should be aware that applying truly distributed decision-making typically yields solution which are less fine tuned with respect to optimal resource utilisation.

The combinatorial optimisation aspects of path management are typically NP-hard; see for instance [1]. Nevertheless, considerable knowledge has been acquired for planning paths in networks and insight and practical methods for obtaining such paths by mathematical programming are available. For an overview, see book by Pióro and Medhi [2] and references therein. Several stochastic optimisation techniques have been proposed [3,4,5,6]. Common to these are that they deal with path finding as an optimisation problem where the "solution engine" has a global overview of the problem and that the problem is unchanged until a solution is found. Hence these techniques are applied off-line and provide management only on longer timescales (minutes-hours).

Schoonderwoerd & al. introduced a system that applies multiple agents with a behaviour inspired by ants to solve problems in telecommunication networks [7]. Their system belongs to a group of systems today known as *swarm intelligence* [8] systems, and has been studied further by others, see for instance [9,10,11] and references therein. Self-management by swarm intelligence is a candidate to meet the aforementioned requirements and to overcome some of the drawbacks of the current path and fault management strategies. Heegaard, Wittner, and Helvik [12] describe CE-ants (cross-entropy ants), a swarm intelligent based system for dealing with path management in communication networks. In CE-ants systems, there is a tradeoff between management overhead (number of management packets) and path recovery time after failure events. This paper focuses on bounding the overhead in CE-ants systems. Novel extensions to the self-tuned rate control proposed by the authors in [13] are presented. With limited management overhead, the resulting CE-ants system performs adaptive multi-path load sharing and stochastic routing with fast restoration on link failures.

The CE-ants system is briefly described in Section 2. The self-tuned ant rate control mechanisms are presented in Section 3. In Section 4 the performance of the self-tuned rate control mechanisms are evaluated by simulations where a nationwide communication infrastructure is applied as the network topology. Further work and concluding remarks are given in Section 5.

## 2 Cross Entropy Ants (CEants)

A CE-ants system is a swarm intelligent system originally inspired by the foraging behaviour of ants. The idea is to have a number of simple ant-like mobile agents, denoted just *ants* in this paper, iteratively searching for paths in a network. Having search for and found a path, an ant backtracks and leaves markings, denoted *pheromones,* resembling the chemicals left by real ants during ant trail development. The strength of the pheromones depends on the quality of the path found. Hence, nodes hold distributions of pheromones pointing toward their neighbour nodes. A new ant in its searching phase visiting a node selects the next node to visit stochastically based on the pheromone distribution seen in the visited node. Using such trail marking ants, together with evaporation of pheromone, the overall process converges quickly toward having the majority of the ants follow the trails that tend to be a near optimal path. Due to limited space the foundations for the CE-ants are only outlined in the following paragraphs. See [11] for more details.

In [6] Rubinstein presents an algorithm for iteratively finding optimal solutions to hard combinatorial problems. The algorithm is founded on the recognition of that finding the optimal solution by random selection is an extremely rare event. Rubinstein represents the total allocation of pheromones in a network by a probability matrix $P_t$ where an element $P_{t,ij}$ reflects the normalised intensity of pheromones pointing from node $i$ toward node $j$. An ant's stochastic search for a sample path resembles a Markov Chain selection process based on $P_t$. By importance sampling in multiple iterations Rubinstein alters the transition matrix ($P_t \rightarrow P_{t+1}$) and increases, as mentioned, certain probabilities such that ants eventually find near optimal paths with high probabilities. Cross entropy is applied to ensure efficient alteration of the matrix. To speed up the process further, a performance function weights the path qualities such that high quality paths have greater influence on the alteration of the matrix. A "temperature" parameter $\gamma_t$ in the performance function controls the focus of the overall search towards high quality paths. Focus is tightened gradually, i.e. $\gamma_{t-1} > \gamma_t > \gamma_{t+1}$, to avoid local optimum (cf. simulated annealing [3].)

A distributed and asynchronous version of Rubinstein's CE algorithm, today known as *CE-ants*, is developed and described in details in [14]. On the contrary to Rubinstein's CE algorithm where all ants must produce sample paths before $P_t$ can be updated, in CE-ants an optimised update of $P_t$ can be performed immediately after every new path $\omega_t$ (where $t$ is the $t$'th ant) is found, and a new probability matrix $P_{t+1}$ can be generated. Hence CE-ants may be viewed as an algorithm where search ants evaluate a path found (and re-calculate $\gamma_t$) right after they reach their destination node, and then immediately return to their source node backtracking along the path. During backtracking, pheromones are placed by updating the relevant probabilities in the transition matrix. Due to the compact autoregressive schemas applied in a CE-ants system, the system becomes both computationally efficient, requires limited amounts of memory and is simple to implement.

In [15] elitism is introduced in the CE-ants system to reduce the overhead by reducing the number of ant updates. The *elite CE-ants* system performs significantly better in terms of the number of path traversals required to converge toward a near optimal path. Only a selected set of ants, denoted the *elite set*, backtrack their paths and update pheromones. This reduces the total number of backtracking traversals and pheromone updates. However, it is still an open question how to determine the optimal ant generation rate with respect to minimise overhead and maximise performance. This paper refines, extends and significantly improves two previously rate adaptation strategies introduced in [13]. The rate adaptation is integrated with the *adaptive path strategy* [12] designed for fast restoration and adaptation to both link failures and changes in traffic load.

## 3 Self-tuning Refresh Rate

On most network topology and traffic load changes, the *adaptive path* strategy will almost immediately provide alternatives to the paths that are affected. This is much due to that stochastic routing and exploration ants contribute to establishment of new paths and sporadicly refresh next best paths. The performance of the path management depends on the number of ants sent. There is a tradeoff between path recovery time and overhead in terms of ant messages as well as a tradeoff between overhead and sensitivity to load change and failure frequencies. Higher rates of messages imply faster recovery and higher sensitivity. In [13] the authors conduct a comparative study of different rate adjustment strategies. Results of the study indicate that the most promising approach is a self-tuning rate control that is implicitly adjusted as network topology or traffic load changes. In [13], this implicit rate adjustment is with some success done at the end points of the path. This section presents an improved self-tuning strategy that applies to all nodes in the network. The system model is first introduced, followed by definitions of performance metrics used for evaluation, and finally details of the self-tuning strategy. The self-tuning strategy is designed to react to link failures and restorations, as well as link overloads that imposes (temporary) excessive delay and traffic loss.

### 3.1 System Model

The system considered in this paper is a bidirectional graph $G(v, l)$ where $v$ is the set of nodes and $l$ is the set of links. The objective is to find a virtual connection, $VC_{[s,d]}$, between source node $s$ and destination node $d$, with a minimum *cost*, e.g. minimum delay between $s$ and $d$. A $VC_{[s,d]}$ is constituted by one or more paths, $\omega_{[s,d]} = \{s, \cdots, d\} \in \Omega_{[s,d]}$ where $\Omega_{[s,d]}$ is the set of all feasible paths between $s$ and $d$. The links (edges), $l_{[i,j]}$, in the graph are specified by its end nodes $i$ and $j$. See Figure 1 for an illustration.

Discovery and maintenance of paths for virtual connections are handled by the CE-ants method and is governed by determination of the *cost* of a path. In this paper the link cost includes queueing and processing delay in the node and transmission and propagation delay of the link. The cost of a link is denoted $c(l)$ and

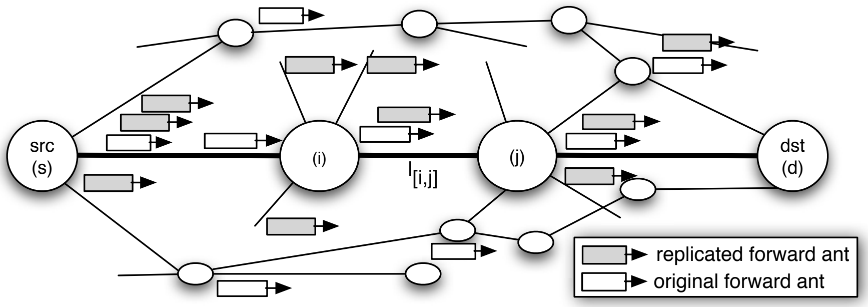


**Fig. 1.** Illustration of self-tuned local ant replication

will vary with traffic load and over time. The cost of a path is additive, $c(\omega) = \sum_{\forall l \in \omega} c(l)$. The cost of a virtual connection is $c(VC) = \min_{\omega \in \Omega} c(\omega)$ when the minimum cost path is used, and $c(VC) = \sum_{\omega \in \Omega} p(\omega) \cdot c(\omega)$ when stochastic routing is applied over the available paths $\omega$ with path probability $p(\omega)$. The near optimal or best paths are referred to as the *preferred paths* of a VC.

The CE-ants system described in this paper is generating management packets, denoted *ants,* at VC source $s$ with rate $\lambda_t$ at time $t$. The suffix $t$ is ignored in the following for notational simplicity. A high rate $\lambda_0$ is applied in the initialisation phase where ants explore the network by doing restricted random walks avoiding revisites of nodes [14]. The short initialisation phase is followed by a convergence phase leading to a steady state where ants are generated at a rate $\lambda_s$ and nearly all traverse the same path. Hence the ant generation rate $\lambda$ varies between $\lambda_0$ and $\lambda_s$, i.e. $\lambda_0 > \lambda > \lambda_s$. The ants contains the destination $d$ and a species identity $id$, $< d, id >$. The identity refers to an ant species with a designated task, in this paper typically finding the minimum cost path from $s$ to $d$. The CE-ants system generates forward and backward ants with rates, $\lambda_f$ and $\lambda_b$, respectively. $\lambda_f = \lambda$ at the source node while $\lambda_b$ is governed by elite selection in the destination node. $\lambda_b \to \lambda_f = \lambda_s$ when the system converges.

The forward and backward rates will change as the state of the links in the network changes. The *link state events* that describe the network dynamics are link failures, link restorations, and link overload. The rates are estimated by discretizing the time axis with granularity $\tau$ and counting the number of ant arrivals in time intervals of size $\tau$. A running average $\hat{\lambda}$ of these rate estimates are generate applying an autoregressive formulation

$$\hat{\lambda}_k = \alpha \hat{\lambda}_{k-1} + (1-\alpha) \cdot \frac{N_k}{\tau}, \qquad k \in \mathbb{P},\, \hat{\lambda}_0 = 0 \tag{1}$$

where $N_k = \|\{t_x | (k-1)\tau \le t_x < k\tau \vee t_x \in \mathbf{T}_a\}\|$ is the number of ant arrivals in time interval $k$. $\mathbf{T}_a$ is the set of ant arrival events, where $a \in \{f, b\}$ if the arrivals are forward $f$ or backward $b$ ants. $\alpha$ is a memory factor that must be set to ensure that the estimators react quickly enough to changes in the traversing behaviour of the ants.

### 3.2 Performance Metrics

In Section 4, the performance of three variants of self-tuned strategies, denoted *global, global-local* and *local*, are compared with the performance of a strategy with fixed rate, denoted *fixed.* Changes in link state during the simulation experiments are defined as *link state events.* The performance is compared by the following metrics:

1. $t_r$ - the *path detection time*, is the time to detect a path in the virtual connection. This is the time it takes from a link state event occurs, $t_e$, to a new path from the source to the destination of a virtual connection is found, i.e. to the first packet from source $s$ arrives at destination $d$. Path detection time indicates the system's ability to sustain a connection.
2. $t_o$ - the *convergence time*, is the time it takes to converge to a new optimal, or at least good, solutions after a link state event. A VC is considered converged when a ratio of $\phi$ [%] of all packets follow the same path. The convergence time indicates the system's ability to reestablish the desired level of service quality for a connection after an link state event.
3. $n$ - the number of ants generated per VC per time unit. The number of ants indicate the amount management overhead generated.
4. $A$ - is steady state *service availability.* The *service* is a virtual connection with guaranteed service level, e.g. packet delay less than $t_{\max}$ [sec.] or packet loss ratio $p_{\max}$ [%] . If the guarantee is violated the service is not available.

### 3.3 Self-tuned Implicit Rate Adaptor in VC Source Node

In [13] the authors introduced implicit rate adaptation of forward ants in the source node of a VC. The forward rate adjustment scheme is based on the relative difference between the current forward ant generation rate $\lambda$ and an estimate of the incoming ant rate $\lambda_b$ (backtracking elite ants). After an adjustment, the ant generation rate is bounded by maximum rate $\lambda_0$ and a minimum rate $\lambda_0 \cdot \varepsilon$ where $\varepsilon \in (0, 1]$:

$$\lambda_k \leftarrow \lambda_0 \cdot max(\varepsilon,\ 1 - \frac{\hat{\lambda}_{b,k}}{\lambda_{k-1}}), \qquad k \in \mathbb{P} \tag{2}$$

The implicit rate adaptor of Eq. (2) is re-applied in this paper. However $\hat{\lambda}_{b,k}$ is now estimated by recording arrivals of backward ants in the arrival set $\mathbf{T}_b$ and using the autoregressive estimator in Eq. (1).

### 3.4 Self-tuned Replication of Ants in Nodes

With the self-tuned implicit rate adaptor from Section 3.3, the detection of a link state event, and the corresponding increase in forward ant rate from the VC source node, will happen some time after the event has occurred. The detection time depends on the network topology, and location and nature of the link state event. In order to reduce the detection time, a local message replication strategy may be added. In [16] a few local replication strategies, ranging from flooding to controlled proliferation, were studied in P2P search strategies on various graphs. Their search criterion is described as a meta-data set. In their controlled proliferation strategy, the amount of overlap between the meta-data set describing the search and what's in the current node determines the number of replica. The more overlap, the more replicas are issued. In the local replication strategy in this paper, the purpose is to broaden the search when instabilities are detected. This means that the number of replicas issued in a given node is not related to the search criterion, i.e. the destination address, but determined by the criticality of the detected link state event.

The local replication strategy suggested in [13] did not respond sufficiently to the link state changes. Therefor, in this paper, a new and novel local replication strategy is suggested. The strategy is applies to all nodes, either as a supplement to the implicit rate adjustment in the VC source node, or as the only rate adaptation. In Section 4 the two variants are denoted *global-local* and *local* respectively.

The basic idea is to detect the events implicitly based only on local information in each node. When a link state event is detected that affects VCs with their preferred path through node $i$, the number of forward ants from node $i$ is decreased or increased. An increase in the forwarding rate is achieved by making copies of the ants that are forwarded from the VC source. In the following it is distinguished between *original forward ants* $< d, id >$ stemming from the VC source node, and *replicated forward ants* $< d, id >^*$ which are replicas generated at one of the intermediate nodes between the source and the destination, see Figure 1 for an illustration. An original forward ant can be replicated, but a replica cannot be further replicated. The path history up to node $i$ of the original ant is copied to the ant replica. Both original and ant replicas are stochastically routed from node $i$ towards the destination according to the current pheromone values in the intermediate node.

As for the self-tuned implicit rate adaptor in the VC source node, the link state events along a preferred path can be detected by monitoring the difference between the forward ant rate $\lambda_f^{(i)}$ and the backward (elite) ant rate $\lambda_b^{(i)}$. If $\lambda_f^{(i)} - \lambda_b^{(i)} \approx 0$ the system has converged and the preferred path is through node $i$. A large difference, $\lambda_f^{(i)} - \lambda_b^{(i)} > 0$, indicates either that the system has not yet converged, or that a link failure has occurred affecting the preferred path through node $i$. Before convergence is reached, a fraction of the ants will not be returned due to the elite selection which rejects ants at the destination with too high cost values, see [15]. This implies that $\lambda_f^{(i)} > \lambda_b^{(i)}$. On a link failure, backtracking (elite) ants will not be able to return, which implies that $\lambda_b^{(i)} \approx 0$. In the converged case, few ants are required, while in the none-converged and failure cases an increase in the forward ant rate is required. The number of replicas is given by the *replication forwarding rate* $\lambda_r$. $\lambda_r$ should be proportional to the difference $\lambda_f^{(i)} - \lambda_b^{(i)}$ to compensate for missing backward ants. Note that in a state of global overload in the network an increased replication rate is not optimal. However by ensuring that ants have low priority, or introducing an upper bound on the ant rate, instabilities are avoided in overload situations by dropping ant packets. Previous studies of the CE-ants system has shown that it is robust to loss of management information and updates [12].

When node $i$ has many out going links, i.e. a high out-degree $\nu_i$, the node may need more replicas to cover the neighbourhood. However, if only one (or a few) link(s) have strong pheromone values, i.e. a high probability of being used, the number of replicas does not need to be in proportion to the number of links since all ants will follow the same (few) link(s) with high probability. In the opposite case, when a node stores a close to uniform distribution of pheromone values, and hence has weak information about which links will be used, more replicas are required to cover the neighbourhood. The *entropy measure*, $E = -\sum_{\forall x} p_x log(p_x)$ originally proposed by Shannon [17], is a suitable means to quantify the node's information about the neighbourhood. Hence in the replication rate estimator of node $i$, the rate difference of forward and backtracking ants is weighted by the out-degree of node $i$ and the normalised entropy of the pheromone based probability distribution $P^{(i)}$ stored in node $i$,

$$\hat{\lambda}^{(i)}_{r,k} = \frac{E^{(i)}_k}{E^{(i)}_{\max}} \cdot \nu_i \cdot (\hat{\lambda}^{(i)}_{f,k} - \hat{\lambda}^{(i)}_{b,k}) = \frac{-\sum_{x=1}^{\nu_i} p^{(i)}_x log(p^{(i)}_x)}{log(\nu_i)} \cdot \nu_i \cdot (\hat{\lambda}^{(i)}_{f,k} - \hat{\lambda}^{(i)}_{b,k}) \quad (3)$$

where $p^{(i)} \in P^{(i)}$.

The $\hat{\lambda}^{(i)}_{f,k}$ and $\hat{\lambda}^{(i)}_{b,k}$ are estimators determined by Eq. (1) using the time epochs in $\mathbf{T}_f$ and $\mathbf{T}_b$, respectively. Only the original forward and backward ants are included in $\mathbf{T}_f$ and $\mathbf{T}_b$. The entropy is scaled by the maximum entropy in node $i$ to produce a factor between 0 and 1. The maximum entropy is when all links have the same pheromone values, i.e. $p^{(i)}_x = 1/\nu_i$, which gives $E^{(i)}_{\max} = -\sum_{x=1}^{\nu_i} 1/\nu_i log(1/\nu_i) = -log(1/\nu_i) = log(\nu_i)$. Note that using the entropy measure only local, node specific, information is required, and no addition tuning parameters are needed.

Eq. (3) is self-adjusting and designed not to impose instabilities. The estimator has the following properties that bounds the replication rate:

1. $\hat{\lambda}^{(i)}_{r,k} \to 0$ as $\hat{\lambda}^{(i)}_{b,k} \to \hat{\lambda}^{(i)}_{f,k}$ : this happens when the path has converged.
2. $\hat{\lambda}^{(i)}_{r,k} \to 0$ as $E^{(i)}_k \to 0$ : this happens when all ants are following the same path, i.e. in node $i$ this implies using the same link.
3. $\max \hat{\lambda}^{(i)}_{r,k} = \nu_i \hat{\lambda}^{(i)}_{f,k}$ : this happens only when node $i$ has not been, or is rarely, visited and no pheromone values exists. The estimation of $\lambda_f$ and $\lambda_b$ is based on observations of original ants only, and in the case where a node has been visited rarely, $\hat{\lambda}^{(i)}_{f,k}$ is close to 0.

## 4 Case Studies of a National-Wide Internet Topology

To study the performance of the dynamic restoration strategies proposed in Section 3, a series of simulation experiments have been conducted on the backbone topology of a Norwegian Internet provider. The topology, illustrated in Figure 2, consists of a core network with 10 core routers in a sparsely meshed topology, a ring based edge network with a total of 46 edge routers, and dual homing access network with 160 access routers. The relative transmission capacities are 1, 1/4 and 1/16 for core, edge and access links, respectively. The avarage number of hops between the access routers is 6.37, see Figure 2 for the hop distribution.

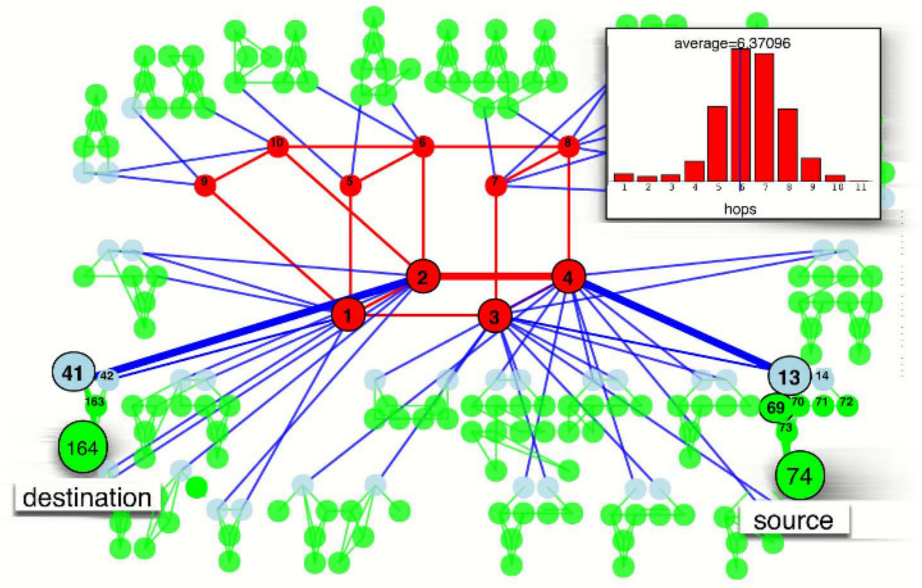
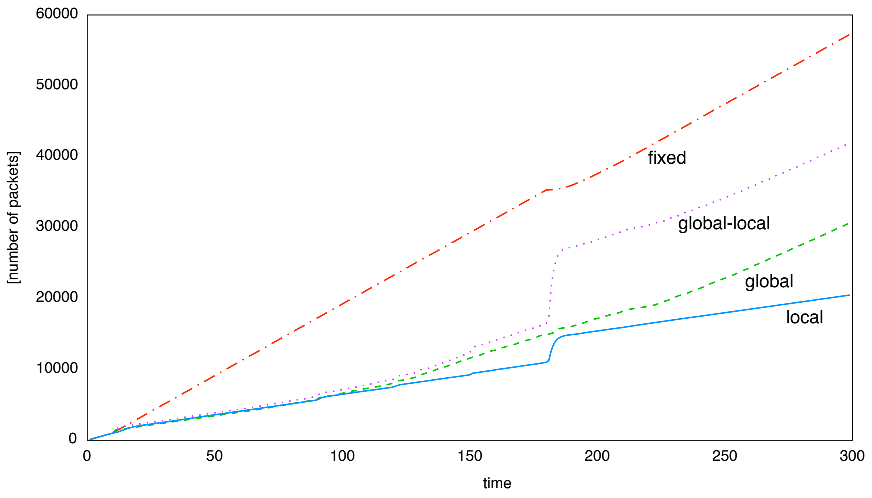
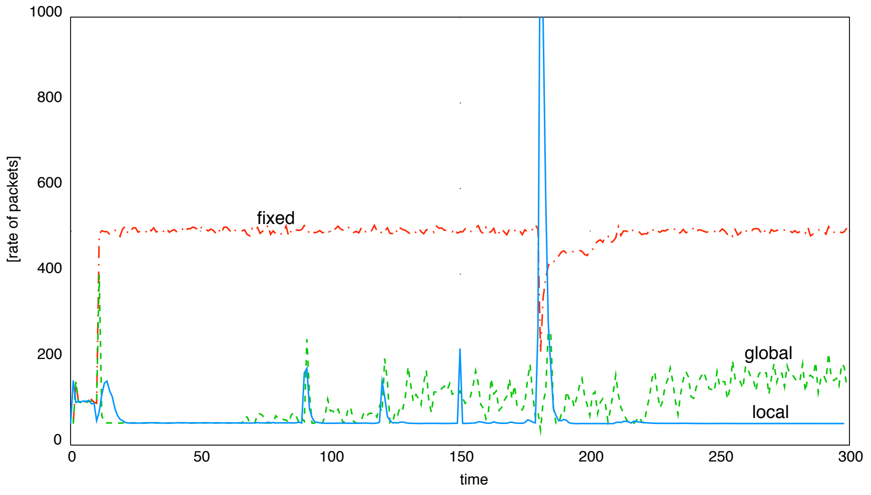


**Fig. 2.** The simulated backbone network. A virtual connection $VC_{[74,164]}$ is established and monitored. The preferred path in stable phase is $\omega^{(1)}_{[74,164]} = \{74, 69, 13, 4, 2, 41, 164\}$. To the upper right, the distribution of the number of hops of all paths between access nodes is given. The average number of hops is 6.37. The preferred path $\omega^{(1)}_{[74,164]}$ of $VC_{[74,164]}$ consists of 6 hops.

The management of a single virtual connection, $VC_{[74,164]}$, and its corresponding paths is studied in details. The paths are exposed to link state events like failures and restoration. Data traffic is routed according to the (multi)paths provided by the management algorithm to improve service availability on link and node failures. Furthermore, by applying multiple (parallel) paths when possible, load distribution is achieved which again reduces delay and provides "soft" proactive protection of the VC. The performance of the data traffic stream between node 74 and 164 is compared under the different restoration strategies. The simulation series runs over a period [0,300] simulated real-time seconds divided in a number of phases where different link state events occur affecting links along the preferred paths of $VC_{[74,164]}$. In [0,10] the network is initialised and explored by the CE-ants system, followed by a stable period [10,90] where all links are operational. In



(a) Accumulative number of ants



(b) Rate of ants

**Fig. 3.** Overhead in terms of forward ants. The local strategy has an overall 86% reduction in number of ants compared to the fixed rate case.

[90,120] $l_{[2,4]}$ has failed and in [120,150] $l_{[1,42]}$ has also failed. $l_{[2,4]}$ is restored in [150,180]. The most critical period is [180,210] where $l_{[2,4]}$, $l_{[1,42]}$ and $l_{[1,3]}$ have simultaneously failed, followed by [210,240] where $l_{[2,4]}$ is restored again.

The simulation experiments compare a fixed rate strategy, denoted *fixed* and having a rate of 500 ants/s, with three self-tuned strategies, one denoted *global* that is only active in the VC source node, one denoted *local* that is active in all

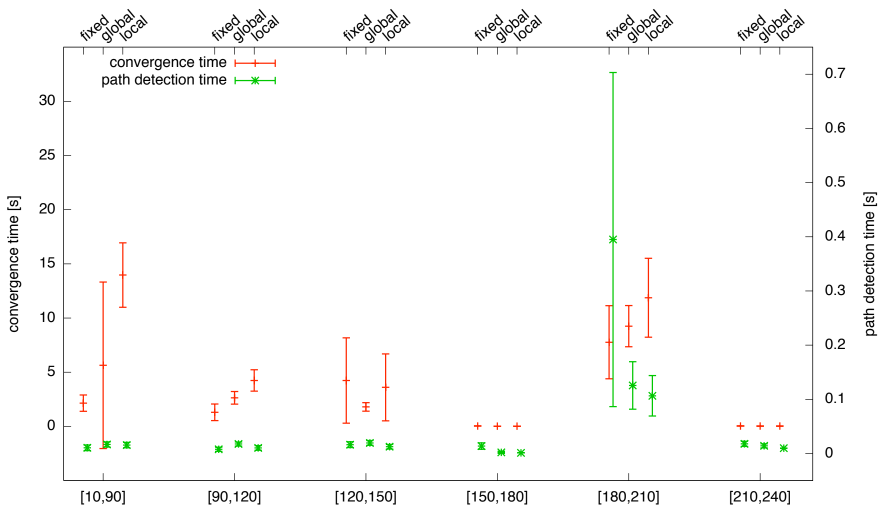


**Fig. 4.** Path detection and convergence times for all schemes

nodes, and finally one denoted *global-local* which is a combination of global and local. For global, local and global-local $\lambda_0$ is equal to the fixed rate case, i.e. 500 ants/s. $\varepsilon$ is set to 0.1 which results in a minimum rate of 50 ants/s. Parameters from Eq. (1) are set to $\tau = 0.4$ [sec.] and $\alpha = 0.8$ (base on a rough parameter study) to ensure a balance between reactivity to rate changes and damping of rate oscillations. Results from simulations of the 4 different strategies are given in Figure 3-6 and are base on 20 simulation replications of each of the strategies. Note that results for global-local are only shown in Figure 3.(a) since this strategy introduced increased overhead without significant improvements in performance compared to the *global* or *local* strategies. Figure 3 shows the accumulated number of ants as well as the rate of ants arriving at the destination node, and hence indicates the amount of overhead generated by the different strategies. In Figure 4 the path detection and convergence times are summarised applying $\phi = 80\%$ as convergence limit. The average and 95%-confidence limits of path detection, $t_r$, and convergence times, $t_o$ are both measured from each link state event in the simulation period. Both Figures 5 and 6 focus on the most critical period [180,210] of the simulation scenario. However, zoom-outs of the whole scenario period are also included. In Figure 5 the average service availability of $VC_{[74,164]}$ is given for $t_{max}$ set to 110% of the delay of the optimal solution in the most critical period [180,210]. Figure 6 shows the data packet performance in terms of delay.

A general observation from the results in Figure 4 is that the path detection time is significantly less than the convergence times (averaged over all $< 0.1\%$). This implies that the service interruptions are very short.

From Figure 3 it is clear that the *global* source rate adaptation and the *local* replication strategies generate significantly fewer ants (management packets),

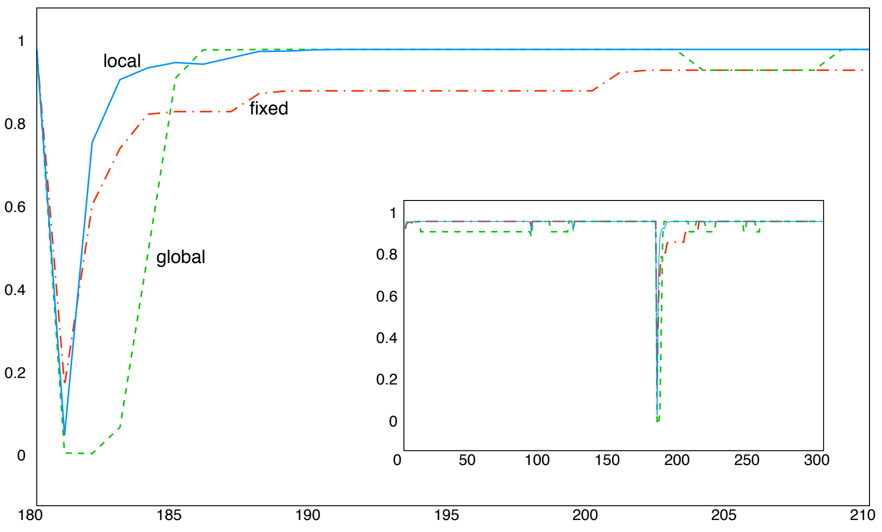


**Fig. 5.** Availability of $VC_{[74,164]}$ where the service level requirement $t_{\max}$ is set to 10% above the delay of the optimal path in period [180,210] when three links are failed

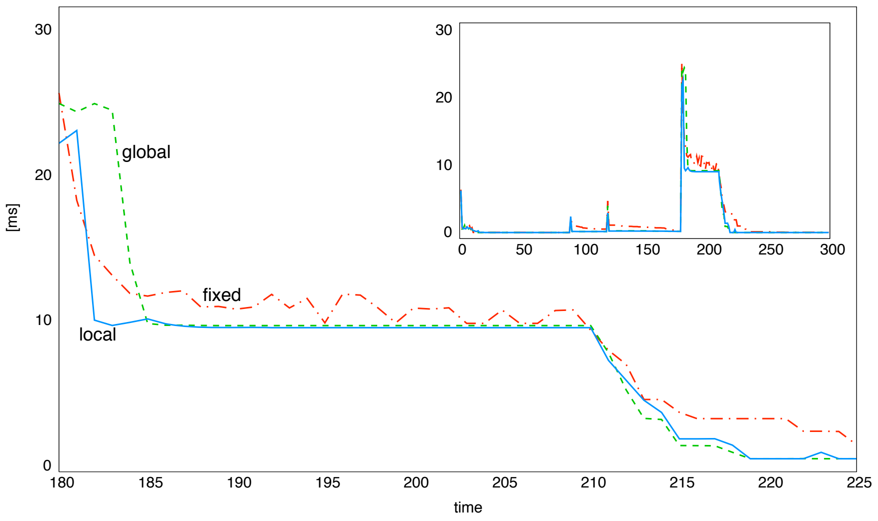


**Fig. 6.** Data packet performance for $VC_{[74,164]}$

in fact 70 - 85 % less than the *fixed* rate strategy. The global-local strategy generates more overhead that global and local. Note that the absolute rates may seem high, especially since they concern signalling traffic for managing a single VC. However, in systems realising CE-ants path management, a single VC will typically carry a number of traffic flows. The overall data rate of the VC will far exceed the ant-signalling rate. Due to the nature of the CE-ants system, piggybacking ant information (typically less than 50 bytes) onto data packets is also an option, and by that a limited number of separate (non-piggybacked) signalling packets will be required.

Figure 6 shows that without reduced overhead, the performance of data traffic for self-tuned strategies in terms of packet delay is similar to, or better than, the fixed rate. This is confirmed by Figure 5 where the same relative performance can be observed for the data service availability. In the critical period [180,210] where a large number of ants are required to find a set of new good paths, the global source rate and the local replication rate are adjusted producing the extra boost of ants required to discover alternative paths. After convergence the rate is self-tuned back to the minimum rate.

Comparing the local replication strategy with the global rate strategy, local replication performs best with respect to path detection time and service availability (Figure 4 and 5). However the global rate strategy performs better with respect to convergence time (Figure 4).

The overall service availability is only slightly distinguishable in the simulated example. Fixed rate has an availability $A = 0.9981$, local replication has $A = 0.9979$, while global has $A = 0.9973$.

## 5 Concluding Remarks

*CE-ants* is a distributed, robust and adaptive swarm intelligence system for dealing with path management in communication networks, and is based on Cross Entropy for stochastic optimisation. In this paper different strategies are studied to control and reduce the overhead in terms of number of management packets (denoted ants) generated in this CE-ants system. Overhead reduction is achieved by controlling the ant generation rate and the number of *elite* ants that update routing information in the network. A series of link state events are simulated and the performance of three self-tuned strategies are compared to a fixed rate approach. The self-tuned strategies involve both ant replication on demand in each nodes and cost sensitive ant rate in VC source and destination. The results show a significant saving (70-85%) in number of packets compared to a fixed rate strategy with the same or better data packet delay and service availability. A purely local replication strategy turns out with very limited overhead to provide fast restoration measured by short path detection times and high service availability.

More case studies should be conducted with different transient periods and combinations of link state events. It is also interesting to study scenarios where differentiated service and dependability requirements lead to differentiated restoration strategies. Further studies of the combination of local replications and implicit source rate adaptation is planned to improve the interplay between the two self-tuned rate adaptation approaches. The scaling, convergence, and bounded properties of the self-tuned replication rate process and the implicit source rate adaptation is also important further work.

Flooding the network under global overload should be avoided, and strategies for piggybacking ant information on data packets and reservation of maximum capacity for ant packets (the ant packets are small) will be considered. It has previously been shown that the CE-ants system is robust to loss of management updates and information. Hence, just dropping ants to avoid contributing to a global overload may be an option.

## References

1. M. O. Ball, *Handbooks in Operation Research and Management Science, Network Models*, vol. 7. North Holland, 1995.
2. M. Pioro and D. Medhi, *Routing, Flow and Capacity Design in Communication and Computer Networks.* ISBN 0125571895, Morgan Kaufmann Publishers, March 2004.
3. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science 220*, pp. 671–680, 1983.
4. F. Glover and M. Laguna, *Tabu Search.* Kluwer Academic, 1997.
5. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1998.
6. R. Y. Rubinstein, "The Cross-Entropy Method for Combinatorial and Continuous Optimization," *Methodology and Computing in Applied Probability*, pp. 127–190, 1999.
7. R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, "Ant-based Load Balancing in Telecommunications Networks," *Adaptive Behavior*, vol. 5, no. 2, pp. 169–207, 1997.
8. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artifical Systems.* Oxford University Press, 1999.
9. G. Di Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, Dec 1998.
10. O. Wittner and B. E. Helvik, "Distributed soft policy enforcement by swarm intelligence; application to load sharing and protection," *Annals of Telecommunications*, vol. 59, pp. 10–24, Jan/Feb 2004.
11. O. Wittner, *Emergent Behavior Based Implements for Distributed Network Management.* PhD thesis, Norwegian University of Science and Technology, NTNU, Department of Telematics, November 2003.
12. P. E. Heegaard, O. Wittner, and B. E. Helvik, "Self-managed virtual path management in dynamic networks," in *Self-* Properties in Complex Information Systems* (O. Babaoglu, M. Jelasity, A. Montresor, A. van Moorsel, and M. van Steen, eds.), Lecture Notes in Computer Science, LNCS 3460 (ISSN 0302-9743), pp. 417–432, Springer-Verlag GmbH, 2005.
13. P. E. Heegaard and O. Wittner, "Restoration performance vs. overhead in a swarm intelligence path management system," in *Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS2006)* (M. Dorigo and L. M. Gambardella, eds.), LNCS, (Brussels, Belgium), Springer, September 4-7 2006.
14. B. E. Helvik and O. Wittner, "Using the Cross Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks," in *Proceedings of 3rd International Workshop on Mobile Agents for Telecommunication Applications*, Springer Verlag, August 14-16 2001.

15. P. E. Heegaard, O. Wittner, V. F. Nicola, and B. E. Helvik, “Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization,” in *Rare Event Simulation and Combinatorial Optimization (RESIM/COP 2004)*, (Budapest, Hungary), September 7-8 2004.
16. N. Ganguly, L. Brusch, and A. Deutsch, “Design and analysis of a bio-inspired search algorithm for peer-to-peer networks,” in *Self-Star Properties in Complex Information Systems* (O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen, eds.), vol. 3460 of *Lecture Notes in Computer Science*, Springer-Verlag, 2005.
17. C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October 1948.

# Cross-Layer Approach to Detect Data Packet Droppers in Mobile Ad-Hoc Networks

Djamel Djenouri[1] and Nadjib Badache[2]

[1] Basic Software Laboratory, CERIST Center of research, Algiers, Algeria
ddjenouri@mail.cerist.dz
[2] LSI, USTHB University, Algiers, Algeria
badache@cerist.dz

**Abstract.** Mobile ad hoc networks (MANETs) are dynamic infrastructureless networks whose routing protocols are fully based on node cooperation, and where each mobile node relies on other intermediate mobile nodes to send packets towards remote ones. Being anxious about its battery shortage, a node participating in the network and using the forwarding service provided by other nodes might behave ***selfishly*** and drop packets originated from others. Such a behavior hugely threatens the QoS (Quality of Service), and particulary the packet forwarding service availability. Another motivation to drop data packets is to launch a DoS (Denial of Service) attack. To do so, a node participates in the routing protocol and includes itself in routes then simply drops data packet it receives to forward. We propose in this paper a novel ***cross-layer*** based approach to detect data packet droppers, that we optimize and decrease its overhead. Contrary to all the current detective solutions, ours is applicable regardless of the power control technique employment.

**Keywords:** mobile ad hoc networks, security, packet forwarding.

## 1 Introduction

The absence of any central infrastructure in MANET imposes new challenges, since services ensured by the central infrastructure must be ensured by mobile devices themselves in this new environment. Particularly, to ensure packets transmission between remote nodes each mobile device (node) acts as a router and devotes its resources to forward packets for others. This consumes its resources, such as its limited battery. In some MANET applications, like in battlefields or rescue operations, all nodes have a common goal and are *cooperative by nature*. However, in many civilian applications such as networks of cars and provision of communication facilities in remote areas, nodes do not belong to a single authority and do not pursue a common goal. Forwarding packets for other nodes is generally not in the direct interest of anyone in such networks, hence there is no good reason to trust nodes and assume that they always cooperate. Indeed, nodes try to preserve their batteries and might misbehave and tend to be *selfish*. A selfish node regarding the packets forwarding process is the one which asks others to forward its own data packets but drops their packets when asked to relay them. Another motivation to drop data packets is rather aggressive; one node could launch a ***DoS attack*** by participating in the routing protocol to include itself in routes and simply dropping packets it receives for forwarding. Whatever the motivation of the dropper is, this dropping threatens the QoS and the service availability in the network. In the rest of this paper we call both the previous malicious and selfish behavior misbehavior on packet forwarding.

Some solutions have been recently proposed to detect such a misbehavior, but almost all these solutions rely on the watchdog technique [1] which suffers from some problems, especially when using the power control technique employed by some new power-aware routing protocols following the watchdog's proposal.

In this paper we deal with this issue and propose a new cross-layer based approach, that consists of a protocol formed by two components. The first component is located in the network layer and is in charge of monitoring nodes' successors forwarding, whereas the second one is located in the MAC layer and is responsible for appending two-hop ACKs (acknowledgments) to the standard MAC ACKs, and for forwarding them. Note that these two-hop ACKs are special ACKs used by our network component. By exploiting the standard MAC ACKs this cross-layer architecture reduces the overhead compared with a standard one layer approach, and gives more robustness. We also propose an optimization for more reducing the overhead. To assess our solution's performance we conduct a simulation study using GloMoSim [2], where we compare our protocol vs. the watchdog.

The remainder of this paper is organized as follows: The next section shows the causes and the effects of the packet dropping misbehavior, and provides an overview the solutions proposed in literature thus far. Section 3 is devoted to the presentation and the analysis of our new approach, followed by the simulation study in the next section. Finally, section 5 concludes the paper and summarizes the future work.

## 2 Related Work

### 2.1 Misbehaving Causes and Effects

Recent studies show that most of one node's energy in MANET is likely to be devoted for packets relaying. For instance, the simulation study in [3] shows that when the average number of hops from a source to a destination is around 5, then almost 80% of a node's transmission energy will be devoted to forward packets for others. This motivates nodes to behave selfishly, and makes them unwilling to relay packets not of direct interest to them. As mentioned before, another motivation for dropping data packets is to launch a DoS attack targeting either the source or the destination of packets. The full reliance on nodes' cooperation makes ad hoc networks hugely vulnerable to this attack.

The packet dropping misbehavior may lead to serious problems when performed by many nodes in the network, such as throughput degradation, latency rise, and network partition that threats the service availability which is one of the security requirements. All these problems affect both well-behaving and misbehaving nodes. Marti et al. [1] have shown by simulation that if 10% to

40% among the network's nodes misbehave on data forwarding, then the average throughput degrades by 16% to 32%. Another study performed by Buttyan and Hubaux [4] has been devoted to investigate the impact of the network size by simulating networks of different sizes (from 100 nodes to 400 nodes) with the same density, and comparing the effect of the same rates of misbehaving nodes on the throughput. The results show that large networks are more vulnerable to this kind of misbehavior.

### 2.2 Current Solutions

The emergent problem of packet dropping misbehavior in MANET has recently received attention amongst researchers, and some solutions have been proposed. These solutions could be classified into two main categories [5]: reactive solutions that aim at detecting the misbehavior when it appears in the network, and preventive solutions which try to inhibit the misbehavior either by motivating nodes to cooperate or by taking measures to prevent packets from being dropped. As far as we know, Marti et al. are the first who dealt with this problem. In [1], they have defined the *watchdog* technique that has been used by almost all the reactive solutions subsequently proposed. This technique is based on the employment of the promiscuous mode. When a node A, which may be either a source or an intermediate node, transmits a packet to B to forward to C, A monitors B's forwarding by promiscuously listening to the carrier and analyzing packets it overhears. A validates B's forwarding if and only if it overhears the packet transmission within a given timeout, otherwise it increments a counter regarding B's dropping. A will accuse B as misbehaving as soon as the counter exceeds a given threshold. In this approach, it is supposed that each transmission can be overheard by all the transmitter's neighbors if no collision takes place, and that the configured threshold is high enough to overcome possible false detections due to channel conditions. The watchdog requires no overhead, and is a relevant solution for detecting misbehaving nodes when the previous assumptions are held.

Nevertheless, subsequent works in the field of the power consumption optimization [6] [7] have proposed to use the power control technique when routing packets. That is, the transmitter does not use a fixed full-power when transmitting data packets to a given receiver, but an adaptable one according to the distance separating the two nodes. Using the watchdog with this power-efficient technique might cause false detections, as illustrated in the following example: Assume B is a well-behaving node that uses controlled powers and the required power from B to C is less than that needed to reach A from B, thereby the packets sent from B to C will not be overheard at A. Consequently, node A will not be able to validate any B's forwarding and may accuse wrongly B as misbehaving. In addition to its failure when employing the power control technique, the watchdog cannot detect the misbehavior in some other cases such as [1]:

1. Partial dropping: node B can circumvent the watchdog by dropping packets at a lower rate than the watchdog's configured minimum misbehavior threshold. Remember that A accuses B as a misbehavior when A remarks that the number of packets dropped by B exceeds a given threshold

2. Receiver collision: after a collision at node C, B could skip retransmitting the packet without being detected by A
3. False misbehavior report: a node may falsely report other innocent nodes in its neighborhood as misbehaving to avoid getting packets to forward through them
4. Insufficient transmission power: B can control its transmission power to circumvent the watchdog. If A is closer to B than C, then B could attempt to save its energy by adjusting its transmission power such that the power is strong enough to be overheard by the monitoring node (A), but less than the required transmission power for reaching the true recipient (C). Note that performing this way is power-efficient for B.
5. Cooperated misbehavior: B and C could collude to cause mischief. In this case, when B forwards a packet to C it does not report to A when C drops the packet. C does the same thing when used as a predecessor of B in some route. This kind of misbehavior is very hard to detect, and is out of the scope of our proposal.

Marti et al. also propose the path rather, which helps to route packets around misbehaving nodes detected by the watchdog. However, no punishment has been defined, and no mechanism allowing nodes to exchange their experience regarding the misbehaving nodes knowledge has been fixed by the authors. More recent proposals have dealt with these issues.

In [8], Yang et al. describe a unified network layer solution to protect both routing and data forwarding in the context of AODV. Michiardi and Molva [9] suggest a generic reputation-based mechanism, namely CORE, that can easily be integrated with any network function. CONFIDANT is another interesting reputation-based solution, proposed by Buchegger and Le-Boudec [10]. Still, all these detective solutions rely on the watchdog technique in their monitor component.

Buttyan and Hubaux [3] propose, model, and analyze an efficient preventive economic-based approach stimulating nodes to cooperate. The authors introduce what they call *virtual currency* or *nuglets*, along with mechanisms for charging service usage. The main idea of this technique is that nodes which use a service must pay for it (in nuglets) to the provider nodes. This concepts is used and generalized to *credit* by Zhong et al. in SPRITE [11], where they propose an improved solution compared with Nuglets. However, SPRITE introduces a centralized point that is not realistic in the ad hoc context. Other stimulating preventive approaches are based on the game theory, such as [12]. These preventive solutions motivate nodes to cooperate, but do not aim at detecting the misbehaving nodes contrary to the previous ones.

In [13], Papadimitratos and Haas present the SMTP protocol. It is a hybrid solution that mitigates the misbehavior effects (packets lost) by dispersing packets, and detects the misbehavior by employing end-to-end feedbacks. This kind of feedbacks allows the detection of routes containing misbehaving nodes, but fails to detect these nodes. Conti et al. [14] [15] propose another interesting end-to-end feedbacks based solution, that uses a cross-layer interaction between the network and the transport layers to decrease the overhead. To overcome the detection problem of end-to-end feedbacks, Kargl et al [16] propose *iterative probing*, that detects links containing misbehaving nodes but fails to detect the appropriate nodes. To find the appropriate node on a link after an iterative probing authors propose the so called *unambiguous probing*, which is based on the watchdog thus suffers from its problems. In this paper we propose a detective solution to mitigate some watchdog's problems, notably failures related to the problems 2 and 4 presented before, and false detections when employing the power control technique.

## 3 Novel Approach

### 3.1 Cross-Layer Two-Hop ACK

Our proposal aims at mitigating the watchdog's problems, especially those related to the power control technique use, with reasonable overhead. To reduce the cost (the overhead) of our solution we use a cross-layer based approach and exploit the MAC layer feedbacks. Our monitoring protocol is composed of two parts; the first one is located at the network layer, whereas the second one is located at the MAC layer which is generally more tamper resistant.

Like in the watchdog, each node monitors the next forwarding of each packet it transmits, and a source routing protocol is assumed to be used. To explain our monitoring process we first deal with one forwarding, and we suppose in the following (like in the previous section) that node A sends packets to B and monitors the forwarding to C. This process can be easily extended to all the hops along the route. We use a new kind of feedbacks we call *two-hop ACK*, an ACK that travels an intermediate node (two wireless links) [17]. Node C acknowledges packets it receives by sending A via B a two-hop ACK. To reduce the overhead, the two-hop ACK transmission from C to B is intergraded within the ordinary MAC ACK.

To prevent B from falsifying two-hop ACKs we suggest to use the following asymmetric cryptography based strategy: Node A generates a random number and encrypts it with C's PK (Public Key), then appends it to the packet's header. When C receives the packet, it gets the number back, decrypts it using its SK (Secret Keys), encrypts it using A's PK, and finally puts it in a two-hop ACK which is sent back to A via B. Instead of sending this ACK in a separate packet, C uses the ordinary MAC ACK it sends back to B upon the reception of the data packet and piggybacks the two-hop ACK to it. Node B, however, could not delay its acknowledgment of the data packet reception from A, hence forwards this ACK to A in a separate packet. When A receives the ACK it decrypts the random number and checks if it matches with the one it has generated, in order to validate B's forwarding regarding the appropriate packet. If B does not relay the packet, A will not receive a valid[1] two-hop ACK and will be able to detect this dropping after a timeout. This detection results in the increasing of a counter on the packets dropped at B, and like in the watchdog, A accuses B as misbehaving when this counter exceeds a configured threshold.

[1] We assume that the encryption algorithm and SKs are robust enough.

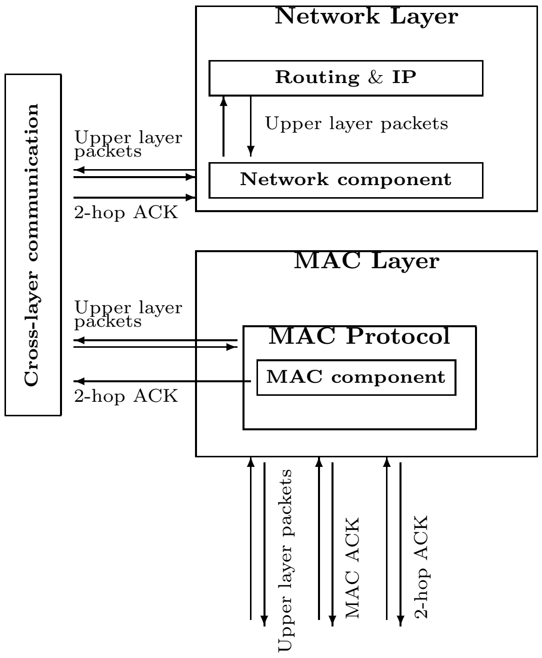


**Fig. 1.** Cross-layer Architecture

Our approach needs a public key distribution mechanism which is out of the scope of our purpose. However, a mechanism like the chain of trust [18] can be used. Note that the same keys could be employed for other security purposes at other layers, and are not inevitably peculiar to our solution.

Figure 1 illustrates our solution's architecture, and the type of packets exchanged between the different components and protocols. Note that the MAC component is only in charge of initiating two-hop ACK, forwarding them, and passing them up to the network module, while the other monitoring functions (requesting two-hop ACKs, validating the forwarding, counting the number o packets dropped,..etc) are the responsibility of the upper component. In spite of the cross-layer design that reduces the overhead, the major drawback of this first solution is still the overhead it engenders as a two-hop ACK is required for each data packet on each couple of hops regardless the nodes' behavior, which is costly. In the following we propose an optimization to more reduce this overhead.

## 3.2 Random Two-Hop ACK

To decrease this cost we suggest to *randomize* the ACK ask, i.e. node A does not ask C an ACK for each packet but upon sending a packet to forward it *randomly* decides whether it asks an ACK or not with a probability $p$, then conceals this decision in the packet. A simple way to conceal the decision is to exploit the random number. For instance, when the node decides to ask an ACK it selects an even number, and an odd number when it decides not to ask the ACK. This *random* selection strategy prevents the monitored node from deducing which packets contain ACK requests. Note that getting such information allows a misbehaving to drop packets with no requests without being detected. When the node decides not to ask an ACK it directly validates the forwarding, whereas when it decides to ask an ACK it waits for it during a timeout like in the ordinary two-hop ACK.The probability $p$ is continuously updated as follows: It is set to 1 (the initial value representing zero-trust) when a timeout exceeds without receiving the requested ACK, and to $P_{trust}$ when the requested ACK is received. This way more trust is given to well-behaving nodes, and the ACK requesting is enforced after a lack of one ACK, which allows to achieve all by the same performance in misbehaving detections (true positives) like the ordinary two-hop ACK, as we will see later.

One possible obvious further optimization to improve the accuracy in detections of the random two-hop ACK monitoring is that node C acknowledges the number of packets transmitted from the last ACK ask, instead of acknowledging merely one packet. This way A would not directly validate packets with no ACK request, but waits for the next requesting. To do this the number of packets acknowledged needs to be carried and *encrypted* in each two-hop ACK, which increases a little bit the overhead. However, we do not investigate this optimization, since the basic random two-hop ACK converges rapidly to the ordinary two-hop ACK (as it will be illustrated in the next section).

## 3.3 Analysis

Unlike the current detective solutions based on the promiscuous mode monitoring (the watchdog), ours relies on the random two-hop ACK. The monitoring node (A) validates the monitored node's (B) forwarding when it receives an ACK from the successor of this latter (C). This process can be generalized along the path for each consequent two hops till the destination, and efficient encryption/decryption operations have been added in order to authenticate the two-hop ACKs and secure the solution against spoofing attacks.

Getting rid of the promiscuous mode based monitoring makes our solution independent of transmission powers, and resolves the watchdog false detection problem related to the employment of the power-control technique. Moreover, our solution resolves the problem 2 of the watchdog (section 2). When a collision appears at C, B should retransmit the packet, otherwise A would not validate its forwarding. This because B's forwarding will not be validated at A until C really receives the packet and sends back the two-hop ACK, unlike the watchdog where the validation is only related to B's first transmission. Our solution also solves the problem 4 of the watchdog. Remember that when A is closer to B than C, then B could save its energy and makes the transmission power strong enough to be overheard by A but less than the one required to reach C. This problem is eliminated in our solution, since B's forwarding validation at A is not just related to B's transmission but to C's reception. Furthermore, the two-hop ACK we use allows to detect the *appropriate* misbehaving node, unlike the end-to-end ACKs [13] and the iterative probing [16].

Our solution has been designed using a cross-layer approach and has been divided into two components; one in the MAC layer and the other in the network layer. This cross-layer approach allows the integration of the two-hop ACK into the ordinary MAC ACK, which reduces the communication overhead. In the rest of this analysis we assume that there is no packet loss. Later in our simulation study we will make more investigations of more realistic scenarios with mobility and collusion. If we assume the average path length is H hops, the worst case communication complexity (when all nodes well behaves) of our first solution is: $O((H-1))$ two-hop ACK transmissions, which is identical to the end-to-end ACK employment and iterative probing [16]. If we used a standard one layer approach then two separate transmissions would be required for each hop, hence the overhead would be $O(2 \times (H-1))$. This communication complexity also represents the number of additional steps to execute the protocol (without considering the forwarding steps) for all the previous protocols, except for the iterative probing [16] in which the communication complexity is $O((H-1) + \log(H-1))$ when a misbehavior occurs; $(H-1)$ for waiting for end-to-end ACKs (used in this approach) and $\log(H-1)$ for probing, and it is $O((H-1))$ when no misbehavior takes place.

Although our first protocol is a bit faster in detection than iterative probing[2] with the same communication overhead complexity, its communication overhead remains high, since an ACK is required for each data packet on each couple of hops regardless the nodes' behavior. Our randomization of the two-hop ACK asking strategy reduces more the overhead, especially when nodes well behave as we will see later. The worst case communication complexity of this solution is $O(p_{trust} \times (H-1))$. The accurate value of the communication overhead (for both the ordinary and random two-hop ACK versions) depends on the nodes' behavior. Suppose that the monitored node misbehaves (drops the packet) with a probability (expectation) $\theta$. Thereby, the reduction factor ($RF$) provided by the random approach optimization (the overhead of the ordinary two-hop ACK/ the overhead of the random version) can be approximated by:

$$RF \approx \frac{1-\theta(1-P_{trust})}{P_{trust}} \tag{1}$$

The steps for computing the overhead of both versions and thus RF are removed due to space limitation, they can be found in our technical report [19].

In the following we discuss the factor for $P_{trust} = 1/4, 1/2, 3/4$.

- when $P_{trust} = 1/4$, $RF \approx 4 - 3\theta$
- when $P_{trust} = 1/2$, $RF \approx 2 - \theta$
- and finally, when $P_{trust} = 3/4$, $RF \approx \frac{4-\theta}{3}$

Now, we discuss the efficiency (accuracy in detection) of the random two-hop ACK vs. the ordinary two-hop ACK. The behavior of the node for each packet

[2] It is faster since it involves $O(H-1)$ steps instead of $O(H-1) + \log(H-1)$.

follows a Bernoulli distribution with a parameter $\theta$ (the expectation which is the probability of dropping). Monitoring n packets could be considered as the repetition of the previous operation (monitoring one packet) $n$ times. Therefore, the number of packets dropped ($pdr$) for n packets is a random variable that is the sum of n random variables which follows a Bernoulli distribution with parameter $\theta$, thus follows a Binomial distribution with expectation: $E(pdr) = \theta \times n$.

Theoretically, the ordinary two-hop ACK detects all this number of packets (when the assumption of no packet loss is held). The purpose now is to assess the number of packets dropped and detected (pd) by the random two-hop ACK, i.e. $E(pd)$, then we will investigate the detection ratio $DR = E(pd)/E(pdr)$. This ratio shows how the random version is close to the ordinary two-hop ACK in detections. The probability of requesting an ACK *of the random two-hop ACK algorithm* is continuously updated, it differs from one operation (monitoring one packet) to another according to the result of the previous operation and the previous behavior. We denote the *algorithm's* probability of requesting an ACK for a packet i (the value of p set by the algorithm for the packet i, which is a random variable) by $P_i$. Consequently, the real probability (in the execution) of asking an ACK for packet $i$ would be expressed by $E(P_i)$. $P_i$ is fixed to 1 if in the previous operation the packet was dropped and detected, that is with the probability[3] $\theta E(P_{i-1})$, otherwise it is fixed to $P_{trust}$, i.e. with probability $1 - \theta E(P_{i-1})$. Therefore, the mathematical expectation of $P_i$ could be expressed by: $E(P_i) = 1 \times \theta E(P_{i-1}) + P_{trust} \times (1 - \theta E(P_{i-1}))$. Hence:
$E(P_i) = P_{trust} + \theta(1 - P_{trust})E(P_{i-1})$.
The number of packets detected by the random strategy ($pd$) also follows a Binomial distribution, since it is the results of repeating a Bernoulli operation $n$ times with parameter $\theta P_i$, but the only difference from the continuous requesting is that in this latter strategy ($P_i$) is not constant. We have:

$$E(pd) = \sum_{i=1}^{n} \theta E(P_i) = \theta \sum_{i=1}^{n} E(P_i) \tag{2}$$

Note that $P_1 = 1$.

**Lemma 1.** $\forall i \geq 1$,
$$E(P_i) = \theta^{i-1}(1 - P_{trust})^i + P_{trust} \sum_{j=0}^{i-1} \theta^j (1 - P_{trust})^j$$

Using this lemma, formula 2 could be developed into:

$$E(pd) = \frac{\theta P_{trust}}{1-\theta(1-P_{trust})} n + \theta(1 - P_{trust}) \frac{1-\theta^n(1-P_{trust})^n}{1-\theta(1-P_{trust})} \left(1 - \frac{\theta P_{trust}}{1-\theta(1-P_{trust})}\right) \tag{3}$$

The steps of simplification, as well as the proof of lemma 1 are available in [19].

[3] The probability of detection is the probability of asking an ACK in the $(i-1)^{th}$ operation. The events dropping the $i^{th}$*packet* and requesting ACK for the $(i-1)^{th}$ packet are independent.

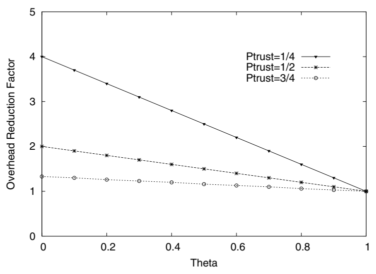


**Fig. 2.** Overhead Reduction Factor

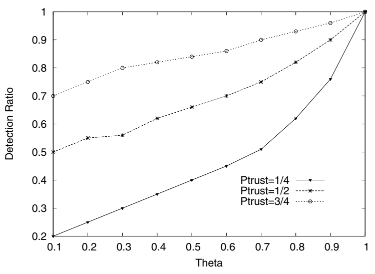


**Fig. 3.** Detection Ratio

This probability depends on many parameters, we will investigate it as well as the detection ratio (DR) vs. some usual values of $P_{trust}$.
For $P_{trust} = 1/4$, $E(pd) \approx \frac{\theta}{4-3\theta}n$, $DR \approx \frac{1}{4-3\theta}$
for $P_{trust} = 1/2$, $E(pd) \approx \frac{\theta}{2-\theta}n$, $DR \approx \frac{1}{2-\theta}$
and finally, for $P_{trust} = 3/4$: $E(pd) \approx \frac{3\theta}{4-\theta}n$, $DR \approx \frac{3}{4-\theta}$

Figures 2 and 3 illustrates respectively the approximated reduction and detection ratios according to $\theta$. We remarque that $P_{trust} = 0.5$ strikes a balance between efficiency (detection ratio) and cost (reduction factor). It decreases the complexity overhead as much as half (when nodes well-behave), while keeping the detection ratio good enough (always $\geq 0.5$). Contrary to $P_{trust} = 0.25$ that has too low values of detection ratio for low and average misbehaving, and to $P_{trust} = 0.75$ that has too low values of reduction factor. Thus, we fix $P_{trust} = 0.5$ later in our simulation study.

As illustrated, authentication of the two-hop ACK packet is ensured by employing encreption/decreption operations on a random number, generated by the monitor and piggybacked to the monitored packet. These operations have minor impact, since they are applied merely on the random number and not on the whole packet holding it. Note that we avoided the use of digital signatures in order to avoid useless packet's hash computation.

## 4 Simulation Results

To evaluate the performance of the proposed protocol we have driven a GloMoSim based [2] simulation study, that we present hereafter. We have simulated a network of 50 nodes located in an area of $1500 \times 1000m^2$, where they move following the random way-point model [20] with an average speed of 1m/s for 900 seconds (the simulation time). To generate traffic we have used three CBR sessions between three pairs of remote nodes, each consists of continually sending a 512 bytes data packet each second. On each hop, every data packet is transmitted using a controlled power according to the distance between the transmitter and the receiver. We compare two versions of our protocol, 2HopACK and Random 2HopACK, as well as the watchdog (WD), with regard to the true positive rate, the false positive rate, and the number of two-hop ACKs (which represents the overhead). We measured these metrics vs. the misbehaving nodes rate, which represents the rate of nodes that misbehave and drop packets they are asked to relay. Each point of the plots presented hereafter has been obtained by averaging five measurements with different seeds. Note that like the watchdog we implemented our protocol with DSR for this simulation. However, it can be implemented with any source routing protocol. Also note that WD requires no kind of ACKs, so the last metric (number of two-hop ACK) concerns merely our protocol's versions. In this study we empirically fixed the tolerance threshold[4] to 100 packets, we plane to make more investigations on optimal values and strategies in our future work.

## 4.1 True Positive Rate

The true positive rate (TPR) represents the efficiency on packet droppers detection. It is the average rate of true detections computed as follows:

$TPR = \sum_{i=0, m_i \neq 0}^{n} \frac{td_i/m_i}{k}$ $td_i$: is the true detections of node i, i.e. the number of misbehaving nodes monitored by node i that are detected.
$m_i$: the number of misbehaving nodes monitored by node i.
$n$: the number of nodes.
$k$: number of nodes that have monitored misbehaving nodes (whose $m_i \neq 0$).

We remark in figure 4 that except for low misbehaving rate (10%) TwoHopACK has the best detection rate above 0.5, and that RandomTwoHopAck has relatively lower TPR values but very close to those of TwoHopACK. We can also see that both protocols outperform WD. The increase of the TPR rate with misbehaving rate can be argued by: In low misbehaving rates the rate of nodes monitored but not judged (for which numbers of packets monitored did not exceed the threshold) is important thus they are not detected because of the experience lack, but as the misbehaving rate increases this rate decreases and monitors get enough samples to make judgments. Defining optimally the tolerance threshold could improve the TPR, especially for low misbehaving rate. This issue will be investigated in our future work.

## 4.2 False Positive Rates

This metric, we denote by FPR, will show how our protocol mitigates false detections of packet dropping due to the power control use.

It is the average rate of false detections giving by the following formula.

$$FPR = \sum_{i=0, m'_i \neq 0}^{n} \frac{fd_i/m'_i}{k'}$$

[4] The number of packets detected dropped upon which the node is accused.

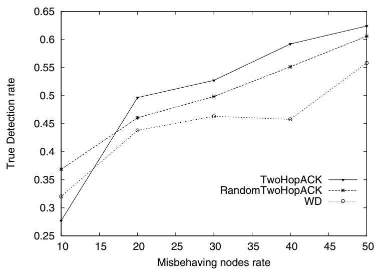


**Fig. 4.** True positives

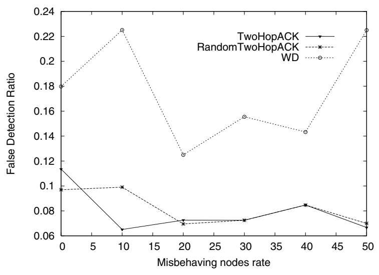


**Fig. 5.** False positives

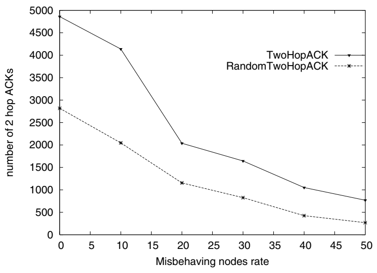


**Fig. 6.** Number of two-hop ACK packets

where:
$fd_i$: is the false detections of node i, viz the number of well-behaving nodes monitored by node i that are wrongly detected.
$m'_i$: the number of well-behaving nodes monitored by node i.
$n$: the number of nodes.
$k'$: number of nodes that have monitored well-behaving nodes (whose $m'_i \neq 0$).

As illustrated in figure 5 both versions of our protocol have low FPR, contrary to WD which causes too high FPR. The FPR of our protocol is basically caused by collisions and nodes mobility, whereas the big difference between our protocol and WD is due to false detections engendered by the power control technique use. Still, optimal definition of the tolerance threshold could minimize the FPR due to channel and mobility conditions.

## 4.3 Overhead

The overhead of our protocol is the number of two-hop ACK packets, depicted in figure 6. Note that WD is excluded here since it requires no communication overhead for monitoring. RandomTwoHopACK reduces considerably the overhead particularly for low and average misbehaving rates. This improvement is due to the efficient technique of randomizing the ACK requests.

## 5 Conclusion and Future Work

In this paper we have proposed a novel cross-layer based solution, aiming at detecting the misbehaving nodes that do not correctly cooperate for forwarding data packets. Instead of using the end-to-end ACK [13] or iterative probing [16] our solution is based on the two-hop ACK, which allows to detect the misbehaving node and not just the route containing such a node or just the appropriate link. Unlike the watchdog, our solution is applicable regardless the power control technique employment. Moreover, it allows the misbehaving nodes detection in some cases where the watchdog fails (in cases 2 and 4 presented in section 2). Our solution is composed of two components, one located at the MAC layer and the other at the network layer. This cross-layer design decreases the communication overhead as much as half compared with a standard network layer implementation of the proposed technique. In spite of this improvement, the first solution requires an ACK for every data packet on each hop, which is still costly. To reduce this cost we have suggested to randomize the two-hop ACK requesting, with probabilities continuously update in such way to give more trust to well behaving node and zero-trust to any node observed dropping a packet. The analysis and simulation results show that the random two-hop ACK is all but as efficient as the ordinary two-hop ACK in high true and low false detections, while hugely reducing the overhead, and that both versions clearly outperform the watchdog especially on false detections. Simulation results also show that there are always possibility of false detections due to collisions and nodes mobility, thus the tolerance threshold is of high importance.

As perspective we plane to provide a rigorous definition to this tolerance threshold that should be well configured to overcome dropping caused by channels and mobility conditions. We also plan to complete the proposal by defining actions to take when a node is accused as a misbehaving, and by proposing a mechanism allowing nodes to exchange their knowledge regarding nodes that misbehave. Monitoring routing control packets, especially those broadcasted with which two-hop ACK is impractical, is also in our agenda.

## References

1. Marti, S., Giuli, T., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: The 6th ACM Conference on Mobile Computing and Networking, MOBICOM 2000, Boston, MA, USA (2000) 255–65
2. Zeng, X., Bagrodia, R., Gerla, M.: Glomosim: A library for the parallel simulation of large-scale wireless networks. In: The 12th Workshop on Parallel and distributed Simulation. PADS'98, Banff, Alberta, Canada (1998) 154–161
3. Buttyan, L., Hubaux, J.: Stimulating cooperation in self-organizing mobile ad hoc networks. ACM/Kluwer Mobile Networks and Applications **8** (2003)
4. L.Buttyan, Hubaux, J.: Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. Technical Report DSC/2001/001, Swiss Federal Institute of Technology, Lausanne, Switzerland (2001)
5. Djenouri, D., Khalladi, L., Badache, N.: Security issues in mobile ad hoc and sensor networks. IEEE Communications Surveys and Tutorials **7** (2005) 2–29

6. Doshi, S., Brown, T.: Minimum energy routing schemes for a wireless ad hoc network. In: The 21st IEEE Annual Joint Conference on Computer Communications and Networking(INFOCOM'02), New York, USA (2002)
7. Djenouri, D., Badache, N.: New power-aware routing for mobile ad hoc networks. The International Journal of Ad Hoc and Ubiquitous Computing (Inderscience) **1** (2006) 126–136
8. Yang, H., Meng, X., Lu, S.: Self-organized network layer security in mobile ad hoc networks. In: ACM MOBICOM Wireless Security Workshop (WiSe'02), Georgia, Atlanta, USA (2002)
9. Michiardi, P., Molva, R.: Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In: Communication and Multimedia Security 2002 Conference, Portoroz, Slovenia (2002)
10. Buchegger, S., Le-Boudec, J.Y.: A robust reputation system for p2p and mobile ad-hoc networks. In: Second Workshop on the Economics of Peer-to-Peer Systems, Harvard university, Cambridge, MA, USA (2004)
11. Zhong, S., Chen, J., Yang, Y.R.: Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In: The 22st IEEE Annual Joint Conference on Computer Communications and Networking(INFOCOM'03), San Francisco, CA, USA (2003)
12. Srinivasan, V., Nuggehalli, P., F.Chiasserini, C., R.Rao, R.: Cooperation in wireless ad hoc networks. In: The 22st IEEE Annual Joint Conference on Computer Communications and Networking(INFOCOM'03), San Francisco, California, USA (2003)
13. Papadimitratos, P., Haas, Z.J.: Secure data transmission in mobile ad hoc networks. In: ACM MOBICOM Wireless Security Workshop (WiSe'03), San Diego, California, USA. (2003)
14. Conti, M., Gregori, E., Maselli, G.: Towards reliable forwarding for ad hoc networks. In: Personal Wireless Communications (PWC 03). Number 2775 in LNCS, Venice, Italy, Springer-Verlag GmbH (2003) 169–174
15. Conti, M., Gregori, E., Maselli, G.: Improving the performability of data transfer in mobile ad hoc networks. In: the Second IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'05), Santa Clara, CA, USA (2005)
16. Kargl, F., Klenk, A., Weber, M., Schlott, S.: Advanced detection of selfish or malicious nodes in ad hoc networks. In: 1st European Workshop on Security in Ad-Hoc and Sensor Networks, ESAS'04, Heidelberg, Germany (2004)
17. Djenouri, D., Badache, N.: New approach for selfish nodes detection in mobile ad hoc networks. In: The first IEEE/Creat-net Workshop on Integration of Security and Quality of Service (SecQoS'05), Athens, Greece (2005)
18. Capkun, S., Buttyan, L., Hubaux, J.P.: Self-organized public-key management for mobile ad hoc networks. IEEE Transactions on Mobile Computing **2** (2003) 52–64
19. Djenouri, D., Badache, N.: Cross-layer approach to detect data packet droppers in mobile ad-hoc networks: extended version. Technical Report LSI-TR-0606, USTHB University, Algiers, Algeria (2006)
20. Maltz, J.B.D., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: The fourth Annual ACM/IEEE International Conference On Mobile Computing And Networking (MobiCom'98), Dallas, TX, USA (1998) 85–97

# On-Demand Distributed Energy-Aware Routing with Limited Route Length*

Cheolgi Kim[1], Kisoo Chang[2], and Joongsoo Ma[1]

[1] Information and Communications University, Daejeon 305-732, Korea
[2] Samsung Advanced Institute of Technology, Suwon 449-712, Korea

**Abstract.** In ad hoc networks, energy preservation on communication is crucial because most of nodes are battery-powered. On the other hand, communication delay is an important factor of real-time communications. Since energy-aware routes commonly have long route lengths, which commonly result long communication delays, a trade-off has to be made on route setup between energy preservation and communication delay. Moreover, long route length also incurs high packet drop rate, which causes low reliability and high retransmission cost. We propose on-demand energy-aware route search algorithms with limited route length (ODEAR-LRL). ODEAR-LRL exploits timer-based RREQ flooding method. Even though timer based RREQ flooding does not guarantee the optimality, our evaluation shows that our on-demand algorithms fairly approximate to the optimal cost.

## 1 Introduction

In ad hoc networks, power consumption is a critical issue because most of the participating nodes are supposed to be battery-powered. Even though the energy optimization algorithms in centralized networks have been widely investigated, most of them cannot be directly adapted to ad hoc networks. In centralized networks, base stations are exploited for energy saving of subscriber stations in many aspects, while ad hoc networks hardly have a device which is able to sacrifice itself for other nodes' energy preservation. Thus, the collaborating methods have been investigated for energy saving in ad hoc networks.

Among energy saving methods in ad hoc networks, a transmission power control is one of the mostly considered approaches. In the method, transmission power is supposed to be controlled to the minimum power level such that the expected receiving power be a given bit error rate (BER), while IEEE 802.11 specifications declares that a transmission power is fixed to a constant value limited by FCC regulation. The receiving power is given by

$$P_R = \frac{\zeta P_T}{d^K} \tag{1}$$

* This research was supported by the SAIT (Samsung Advanced Institute of Technology), Korea, and 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 Assessment).

where $\zeta$ is a shadowing coefficient, $P_T$ is transmission power level, $d$ is the distance between the nodes and $K$ is a path loss coefficient, usually larger than two [1]. Given BER, transmission power is a function of the distance between the source and the destination. If a distance between them decreases to a half, the transmission power commonly decreases to a value less than a quarter.

Because of the exponent $K$, which is usually larger than two, the triangular inequality cannot be applied to the total transmission power of a route. Thus, the total transmission power can be reduced by including appropriate intermediate relay nodes in the route. In this article, we call the routing schemes exploiting this property as *energy-aware routing.* There have been a lot of such energy-aware routing algorithms proposed [2, 3, 4, 5, 6, 7]. Notice that energy-aware routes commonly have long route lengths (or hop counts) because of many relay nodes to reduce the total transmission power. The long route length incurs two problems as follows:

**High end-to-end packet loss rate.** End-to-end packet delivery rate is lower with longer route length since packet error rate per hop is commonly fixed when transmission power is controlled. It not only reduces the quality of service but also increases retransmission rate in a reliable connection. Moreover, it results poor actual energy performance due to the retransmissions or lost packets.

**Long end-to-end delay.** End-to-end delay becomes long because of a number of relays in a energy-aware route. It is critical especially in real-time applications, such as voice over IP (VoIP). In VOIP as well as Multimedia applications, the packet becomes useless if the delivery of a packet exceeds the deadline.

To address them, we propose On-Demand Energy-Aware Routing with Limited Route Length (*ODEAR-LRL*), which is the distributed approximation of optimal centralized algorithm. Our evaluation shows that the approximations perform fairly close to the optimal.

## 2 Related Work

Energy-aware route can be effective only if appropriate power control MAC is supported. Power control MAC must be carefully designed as long as RTS/CTS is maintained for the hidden terminal problem. S. Agarwal et al. designed power-control MAC on 802.11 and evaluated their proposal in group mobility model [8]. E.-S. Jung et al. showed that the nodes that cannot correctly listen to RTS or CTS may interfere the communication with power-control MAC, and proposed their scheme to resolve it [9]. A. Muqattash et al. proposed POWMAC, which controls transmission power for not only energy saving but also capacity enhancement, by a new handshaking scheme RTS/CTS/DTS [10].

The energy gain only with power-control MAC is not high enough. The synergy can be gained from energy-aware routing each link distance of which is relatively short such that the power control is effective for energy saving. The energy-aware routing proposals to minimize the transmission power consumption have been proposed, as well, as follows. How to estimate per-link transmission power is one of the most concerned issues. L. De Nardis et al. proposed to use UWB ranging scheme [5] while S.-H. Lee et al. proposed received signal strength[7]. C. Gentile et al. proposed kinetic minimum-power routing which estimates the projected locations of mobile nodes for power control [11]. As the complexity of energy-aware routing is high, some approaches have exploited simplicity of cluster in power control [2, 6]. Energy-aware routing emulation scheme also has been proposed by putting relay nodes in the middle of existing route in MAC layer by J. Gomez et al. [3].

In the perspective of the whole network, not only total energy must be minimize, but also energy consumption must be decentralized to many nodes to extend the network life time as nodes in the middle of network or on the bottle neck area may consumes their battery more rapidly. S.-J. Baek et al. proposed multi-path routing scheme for energy balancing [12]. The hot communication links can be diffused by communicating through multiple routes between a single pair of source and destination. To prefer the nodes with more remained battery life as candidate nodes of routes, M. Maleki et al. proposed a routing scheme with RREQ piggybacked with each node's battery information [13]. I. Oh et al. proposed similar scheme without battery information piggybacking but delayed RREQ broadcasting for low battery nodes [14]. Some work fully exploits some nodes powered by AC inlets to minimize the battery consumption by battery powered nodes [15, 16].

S. Banerjee et al. argued analytically that the route with many relay nodes do not always perform better with the proper metric including the packet error recovery efforts in [17]. They showed there is optimal number of relay nodes for *end-to-end retransmission model.* A few relay nodes do not exploit the potential reduction in the transmission energy, while large number of relay nodes cause the overhead of retransmissions to dominate the total energy budget.

## 3 Problem Statement

The goal of ODEAR-LRL is to minimize the total transmission power cost not to exceed a given route length. In this paper, We consider two transmission energy models of [17] in our proposals: *HHR* (hop-by-hop retransmission model) and *EER* (end-to-end retransmission model).

HHR model assumes that per-link ARQ is intensively used to minimize per-link packet loss so that per-link packet delivery probability is near one. The total transmission energy consumed in HHR model is given by

$$E_{\mathrm{HHR}}(X) = \sum_{i=1}^{n} E(x_{i-1}, x_i) = \alpha \sum_{i=1}^{n} d^K_{x_{i-1},x_i} \tag{2}$$

where $x_i$ is $i$-th node in the route, $d_{i,j}$ is distance between node $i$ and $j$, and $\alpha$ is transmission energy coefficient.

In EER model, packets are lost in the middle of end-to-end delivery because per-link packet drop rate is not negligible. Thus, end-to-end retransmission is needed to recover the packet loss. The total transmission energy consumed in EER model is approximately given by

$$E_{\mathrm{EER}}(X) = \frac{\alpha \sum_{i=1}^{N} d_{x_{i-1},x_i}^{K}}{(1-p_l)^N}. \tag{3}$$

where $p_l$ is per-link packet loss rate.

The reason why our approaches limit the number of hops is two folds: (1) According to S. Banerjee's work [17], there is optimal route length, which minimizes the total transmission energy regarding retransmission cost in EER model. J. Bicket et al. insisted that 45% of per-link packet loss rate is reasonable in wireless mesh networks [18]. If per-link ARQ (Automatic Repeat reQuest) mechanism is not intensively used, retransmission cost will increase with route length, nullifying the energy saved by many relay nodes. (2) Moreover, some real-time applications need to limit the end-to-end delay for the quality of service. Even though the end-to-end delay is not represented as a function of route length, it is highly correlated with the route length. Therefore, the delay requirement can be fairly achieved by the limit of the route length. We describe and formalize the problems in the following subsections.

**Optimal Route Length Regarding Retransmission Cost.** In EER model, retransmission cost increases in super-linear order with respect to route length where total transmission energy decreases in near inverse proportion. Thus, from a point of route length, the total transmission energy regarding retransmission cost starts to increase, and that point will be optimal route length.

In [17], optimal route length can be obtained by

$$N_{\mathrm{opt}} = \frac{K-1}{-\log(1-p_l)}. \tag{4}$$

Note that the optimal route length depends only on $K$ and $p_l$, not on $D$. Thus, the nodes can easily derive the optimal route length with $K$ and $p_l$, which can be obtained from the communication environment and link status. With known optimal route length, our algorithm achieves minimum total transmission energy route in EER model.

**Multimedia Application Case.** Even in HHR model, our algorithm can be used to enhance QoS properties of multimedia applications. Most of multimedia applications have delay constraints due to the real-time and interactive properties. We suppose that the end-to-end delay is proportional to the route length, and some measured data in mesh network also shows that the communication delay is highly correlated with route length [18]. However, S.-T. Sheu et al. have insisted that longer route length does not directly mean longer delay [19]. The reason is two folds: (1) longer link will have lower SINR with a fixed transmission power MAC (2) and the link with longer distance will contend with more nodes due to the larger interference area. The first fold can be ignored in our work because we assume that nodes control their transmission powers. The second one can be significant in the high contention environment. However, Most of ad-hoc network protocols have QoS-aware MACs, which provide higher priority to the time-critical applications. For example, applications with high priority can have shorter inter frame spacing time, or have reserved slot for communications. Thus, we suppose that the one-hop delay would hardly depends on the link distance for the multimedia application, when route length inherently affects the communication delay. With this assumption, we can easily derive the maximum route length in terms of the time constraints of a certain application.

## 4 On-Demand Distributed Energy-Aware Route Search Algorithm with Limited Route Length

**Lemma 1.** *The minimum energy route based with route length limited to $h$ between node $u$ and $v$, $\bar{R}_h(u,v)$ on HHR model is given by*

$$\bar{R}_h(u,v) = [\bar{R}_{h-1}(u,w); v] \tag{5}$$

*such that $w$ minimizes $E_{HHR}([\bar{R}_{h-1}(u,w); v])$ and $(w,v) \in E$*

*Proof.* Let us represent $\bar{R}_h(u,v)$ as $[\hat{R}(u,w); v]$. Then, the second last node of the route is $w$. For a contradiction, we assume that $\hat{R}(u,w) \neq \bar{R}_{h-1}(u,w)$. Now, we have

$$E_{\text{HHR}}(\bar{R}_{h-1}(u,w)) > E_{\text{typ}}(\hat{R}(u,w)). \tag{6}$$

However, $\bar{R}_{h-1}(u,w)$ is the minimum energy route from $u$ to $w$ with limited route length $(h-1)$. As $\hat{R}(u,w)$ must have a route length at most $h-1$, Eq. 6 conflicts with the definition of $\bar{R}_{h-1}(u,w)$. □

**Lemma 2.** *The minimum energy route based on the retransmission-aware energy model with route length limited to $h$ between node $u$ and $v$, $R_h(u,v)$ is given by*

$$R_h(u,v) = [R'; v] \text{ where } R' \in \bigcup_{(w,v)\in E \text{ and } 0\le i<h} R_i(u,w) \tag{7}$$

*such that $R'$ minimizes the total energy cost $E_{ret}([R'; v])$.*

*Proof.* Assume that $R_h(u,v) = [R'; v]$ and $R' \notin \bigcup_{(w,v)\in E \text{ and } 0\le i<h} R_i(u,w)$. Moreover, suppose that $j$ is the route length of $R'$ and $R'$ is represented as $[u; \cdots; w]$, so let $w$ be the destination of $R'$. The assumption can be simplified as $R' \neq R_j(u,w)$ with the definition of $j$ and $w$. The total energy consumption based on retransmission-aware energy model of $R'$ and $R_j(u,w')$ is given by

$$E_{\text{EER}}([R'; v]) = \frac{E_{\text{ret}}(R')}{1-p_l} + \frac{\alpha d_{w',v}^K}{(1-p_l)^j}$$

$$E_{\text{EER}}([R_j(u,w'); v]) = \frac{E_{\text{ret}}(R_j(u,w'))}{1-p_l} + \frac{\alpha d_{w',v}^K}{(1-p_l)^j}$$

respectively. $E_{\mathrm{EER}}([R';v])$ must be smaller than $E_{\mathrm{EER}}([R_j(u,w');v])$ by the assumption. However, it conflicts with the fact that $E_{\mathrm{EER}}(R') > E_{\mathrm{ret}}(R_j(u,w'))$ by the definition of $R_j(\cdot)$. □

Lemma 1 and 2 shows that energy-aware route with limited route length can be calculated from the vectors of energy-aware routes with limited route lengths and those of their energy values of neighbor nodes. Similar idea has been used in typical energy-aware routing.

In [7], each intermediate node collects the energy and route information from RREQ broadcasted by neighbor nodes for minimum consumed power routing (MCPR). After collecting *'enough'* information, it broadcasts its own RREQ with the optimal intermediate route calculated and its energy cost. The broadcasting wave reaches to the destination, and the destination selects the optimal one. The dilemma is here, 'Which node will broadcast RREQ earlier and which will later?' Each node collects the route information from the neighbor nodes, but reversely, it is a neighbor of other nodes. If Node A has listened to the RREQ of Node B before Node A's RREQ, Node B would not have listened to the RREQ of Node A before Node B's RREQ. The node of early broadcasting cannot listen to the information from the node of later broadcasting before RREQ. So the order of RREQ broadcasting concludes the order of the nodes in a resulted route.

MCPR uses a heuristic of 'the node with less expected power first.' Each node predicts the total energy from the source node to itself by the RREQ receiving time and the receiving power. The RREQ transmission time is decided by expected total energy. With smaller expected total energy, the RREQ transmission time becomes earlier. On RREQ transmission, the node piggybacks the predicted energy and route on its own RREQ packet that it broadcasts [7].

Another heuristic to decide the order, which we can use, is 'the closer node from the source node first.' Since the route is from source to destination a link with a reverse direction would be rarely helpful. To realize the heuristic the RREQ transmission time must be decided with predicted distance from the source node not with the predicted total transmission power. It can be easily transformed from power to distance by Eq. 1. Other than expected total power, ODEAR-LRL protocols adopt timer-based broadcasting to serialize the RREQ transmissions in an order of distances from the sorce node. However, the information maintained in ODEAR-LRL is different from that in MCPR. While MCPR maintains single scalar of energy cost of a minimum energy route, ODEAR-LRL maintains a vector of energy costs and routes with respect to the route length based on Lemman 1 and 2. Each node's operations on RREQ reception and transmission are described at the end of this section. *RECEIVE_RREQ* is a procedure on RREQ reception. *tx_node* is a node that has transmitted the RREQ, *rret_id* is RREQ identifier, *cost* is the energy cost from the *tx_node* to the receiver, $e\_v[0 \cdots h]$ is the energy cost vector from the source to the *tx_node* and $r\_v[0 \cdots h]$ is the vector of routes from source to the *tx_node* with respect to the route length. $E(\cdot)$ would be a cost function of either HHR model or EER model. TRANSMIT_RREQ is a procedure on RREQ transmission except at the destination node. The destination node is supposed to reply with RREP on TRANSMIT_RREQ because it does not need to rebroadcast the RREQ again.

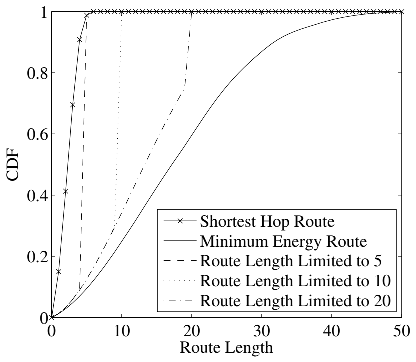
```
**procedure** $RECEIVE\_RREQ(tx\_node, rreq\_id, cost, e\_v[0 \cdots h], r\_v[0 \cdots h])$
    Update RREQ tx timer

    **if not**$(e\_vector(rreq\_id)$ exists) **then**
        **create** $e\_vector(rreq\_id)$ and
        $e\_vector(rreq\_id)[0 \cdots h] \leftarrow \infty$
        $r\_vector(rreq\_id)[0 \cdots h] \leftarrow \varnothing$
    **end**

    **for** $i = 1 \cdots h$
        $total\_energy \leftarrow E(e\_v[i-1], cost, i)$
        **if** $total\_energy < e\_vector(rreq\_id)[i])$ **then**
            $e\_vector(rreq\_id)[i] \leftarrow total\_energy$
            $r\_vector(rreq\_id)[i] \leftarrow [r\_v[i-1]; tx\_node]$
        **end**
    **end**
**procedure** $TRANSMIT\_RREQ(rreq\_id)$
    **if** this node is the destination of $rreq\_id$ **then**
        Transmit RREP with the route $r\_vector(rreq\_id)[h]$
    **elseif** this node is the source of $rreq\_id$ **then**
        Transmit RREQ with $e\_v[0 \cdots h] \leftarrow 0$ and $r\_v[0 \cdots h] = itself$
    **else**
        Transmit RREQ with $e\_v \leftarrow e\_vector$ and $r\_v \leftarrow r\_vertor$
    **end**
```

## 5 Performance Evaluation

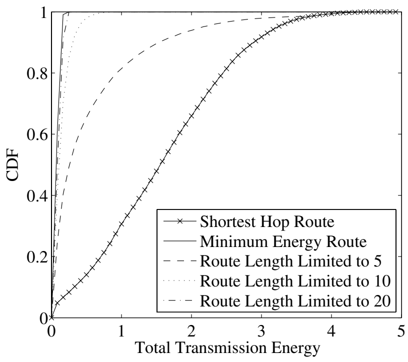
We simulated the ODEAR-LRL on HHR model. Our simulator was developed on MATLAB 7.1. 200 nodes were deployed in 800 m $\times$ 800 m 2-D square. The communication range of each node is assumed to be 200 m. Mobility and fading effects other than path-loss were not considered. The routes were assumed to be built based on the location information of the participating nodes and all possible pair of source and destination nodes were equally measured.

As shown in Fig. 1 (a), The minimum energy route lengths grow up to 48 hops in the simulation environment. However, EAR-LRL are shown to have strictly limited route. Our approach is expected to have good delay bound property for multimedia applications.

Fig. 1 (b) presents the total energy consumed per route in communication with typical energy model when $\alpha = 1$ and $K = 4$. Each possible route is assumed to be used once for normalization. As shown, the route with limited route length of 20 is near optimal, even though its maximum route length is less than a half of that of minimum energy route. Moreover, the consumed energy with limited route length of 10 is also competent with significantly shorter route length.



(a) The cdf of route length



(b) The cdf of energy consumed

**Fig. 1.** The cdf (cumulative distribution function) of route length and energy consumed in routing with typical energy model when $\alpha = 1$ and $K = 4$ in terms of the route setup algorithms

We simulated ODEAR-LRL assuming log-normal fading. Fig. 2 shows the average energy inefficiency per route compared with the optimal energy aware route with limited route length due to the inaccuracy of ODEAR-LRL. We simulated on 10 different topologies. All possible routes are equally measured. When RREQ is received, the predicted distance and received power is not accurately the function of distance because of the log-normal fading. However, communication energy cost is not assumed to be affected by log-normal fading by the weak law of large number because the communication quantity is supposed to be sufficiently large.

There are two reasons of the inaccuracy of ODEAR-LRL: inherent limitation of on-demand approach and the effect of fading. ODEAR-LRL has inherent inefficiency because of imperfect information of the neighbors. Each node can collect the route information of the nodes that are nearer to the source node by predicted distance. If farther nodes have more efficient route information, the selected route would be less efficient than the optimal. Moreover, fading also affects the performance. If a fading variance is larger, the predicted distance from the RREQ as well as the estimated costs of the routes are more inaccurate. The figure shows that the effect of fading is much larger than the effect of RREQ ordering. When standard deviation of RREQ is 10 dB and the route length limit is 10 (the worst case in our simulation), the average inefficiency of routes becomes about 0.5 compared with the optimal EAR-LRL routes even while the inefficiency is less than 5% without fading. Considering the path loss coefficient $K = 4$, which needs 16 times larger power for doubled distance, it can be regarded as reasonably good efficiency, we interpret.

The ratio of failure to make routes by ODEAR-LRL is about 0.015 with route length limited to 5 and zero with route length limited to 10 even though the feasible routes exist. It rarely varies by the log-normal fading. The routing failure ratio seems to be fairly low even though the resulted routes are not the optimal ones.

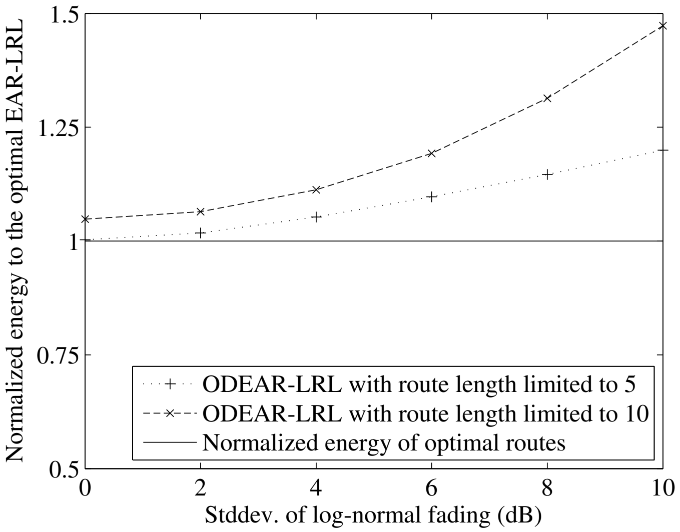


**Fig. 2.** The normalized total energy consumed in routes by ODEAR-LRL compared to the optimal EAR-LRL with respect to the standard deviation of log-normal fading

## 6 Concluding Remarks

We proposed the on-demand distributed approximation of EAR-LRL protocols (ODEAR-LRL). It can be used for multimedia applications on ad hoc networks, which need small energy consumption and low latency together. Moreover, it can be used to build a energy-aware route based on EER model, as well. The performance results show that ODEAR-LRL makes routes with fairly competitive energy costs and good delay characteristics. ODEAR-LRL protocols generate near optimal routes without fading. Even though the resulted routes become more inefficient with large fading effect, their performance is still up to 1.5 compared with the optimal ones with 10dB standard deviation log-normal fading.

In future, we plan to investigate the effect of mobility. We hopefully wish that the energy estimation on communication would help the accuracy of the energy cost estimation. Route vector caching is also considered for accuracy and route setup efficiency.

## References

[1] ElBatt, T.A., Krishnamurthy, S.V., Connors, D., Dao, S.: Power Management for Throughput Enhancement in Wireless Ad-Hoc Networks. In: Proc. of IEEE ICC 2000. (2000)

[2] Kwon, T.J., Gerla, M.: Clustering with Power Control. In: Proc. of the 18th IEEE MILCOM. (1999)

[3] Gomez, J., Campbell, A.T., Naghshineh, M., Bisdikian, C.: Conserving Transmission Power in Wireless Ad Hoc Networks. In: Proc. of the 9th IEEE ICNP. (2001)

[4] Toh, C.K.: Maximum Battery Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks. IEEE communications Magazine (2001)

[5] De Nardis, L., Giancola, G., Di Benedetto, M.G.: A power-efficient routing metric for UWB wireless mobile networks. In: Proc. of IEEE 58th VTC 2003-fall. Volume 5. (2003) 3105–3109

[6] Gentile, C., Haerri, J., Dyck, R.E.V.: Kinetic Minimum-Power Routing and Clustering in Mobile Ad-Hoc Networks. In: Proc. of IEEE 56th VTC 2002-fall. Volume 3. (2002) 1328–1832

[7] Lee, S.H., Choi, E., Cho, D.H.: Timer-based broadcasting for power-aware routing in power-controlled wireless ad hoc networks. IEEE Communications Letters **9** (2005) 222–224

[8] Agarwal, S., Krishnamurthy, S.V., Katz, R.H., Dao, S.K.: Distributed Power Control in Ad-hoc Wireless Networks. In: Proc. of the 12th IEEE Int'l Symp. on PIMRC. (2001)

[9] Jung, E.S., Vaidya, N.H.: A Power Control MAC Protocol for Ad Hoc Networks. Wireless Networks **11** (2005)

[10] Muqattash, A., Krunz, M.: A Single-Channel Solution for Transmission Power Control in Wireless Ad Hoc Networks. In: Proc. of the 5th ACM Int'l Symp. on MobiHoc. (2004)

[11] Gentile, C., Van Dyck, R.E.: Kinetic Spanning Trees for Minimum-Power Routing in MANETS. In: Proc. of IEEE 55th VTC 2002-spring. Volume 3. (2002)

[12] Baek, S.J., de Veciana, G.: Spatial energy balancing in large-scale wireless multihop networks. In: Proceedings of IEEE INFOCOM 2005. (2005)

[13] Maleki, M., Dantu, K., Pedram, M.: Power-aware Soure Routing Protocol for Mobile Ad Hoc Networks. In: Proceedings of the 2002 Int'l Symp. on Lowpower Electronics and Design. (2002)

[14] Garcia, J.E., Kallel, A., Kyamakya, K., Jobman, K., Cano, J.C., Manzoni, P.: A Novel DSR-based Energy-efficient Routing Algorithm for Mobile Ad-hoc Networks. In: Proceedings of IEEE VTC 2003-Fall. (2003)

[15] Lindgren, A., Schelén, O.: Infrastructured ad hoc networks. In: Proceedings of the 2002 Int'l Conf. on Parallel Processing Workshop. (2002)

[16] Oh, I., Choi, J.: A Pseudo Base Station based Algorithm for Power Aware Routing in UWB Ad Hoc Networks. In: Proceedings of ICCS 2004. (2004)

[17] Banerjee, S., Misra, A.: Minimum Energy Paths for Reliable Communication in Multi-hop Wireless Networks. In: Proc. of the 3rd ACM Int'l Symp. on MobiHoc. (2002)

[18] Bicket, J., Aguayo, D., Biswas, S., Morris, R.: Architecture and Evaluation of an Unplanned 802.11b Mesh Network. In: Proceedings of ACM MobiCom '05. (2005)

[19] Sheu, S.T., Chen, J.: A novel delay-oriented shortest path routing protocol for mobile ad hoc networks. In: Proc. of IEEE ICC 2001. (2001)

# Automatic Data Locality Optimization Through Self-optimization

Rainer Buchty, Jie Tao, and Wolfgang Karl

Universität Karlsruhe (TH), Institut für Technische Informatik, 76128 Karlsruhe, Germany
{buchty, tao, karl}@ira.uka.de

**Abstract.** Data locality optimization in parallel systems is a non-trivial task. This task is typically done by the programmer: based upon an exhaustive analysis of an application's run-time behavior, data access and distribution is re-modeled manually. Once the system, application, or just the input data set changes this effort has to be repeated.

Ideally, this task can be automated which requires introduction of Self-X qualities into the system. We developed an architecture concept for self-organizing parallel computer systems. This architecture is based on two main principles which are flexible monitoring to instantiate self-awareness, and adaptive components for all aspects of self-configuration. It is completed by a self-awareness mechanism, the autonomic planning. These Self-X properties pervade all system layers.

Based on this architecture concept, we implemented an autonomic data locality optimization system. With the achieved results presented in this paper we successfully demonstrated suitability and applicability of the architecture concept and were able to highlight the benefits of autonomic data locality optimization.

## 1 Introduction and Motivation

Past prognoses always saw Moore's Law as the ultimate barrier for future system development. However, in modern systems the complexity has risen to an amount where not physical constraints but the complexity itself turns into a problem. While it is possible to build such complex systems, maintenance and especially application optimization are increasingly difficult to handle by humans.

Because of growing system complexity combined with increasing usability and reliability requirements, such tasks should be fulfilled automatically, i.e. the systems with self-organizing characteristics and capabilities are mandatory. In general, such systems require introspection to acquire system-wide state-information to tune for desired performance, energy consumption, reliability, security, or other metrics. In a typical self-organizing system, a monitor probe will acquire state, and then a steering component will adapt the system – either dynamically at runtime, or statically as in profile directed feedback techniques for future executions.

This requires flexible and powerful monitoring resources on all system layers. It is vital to avoid restriction to some few monitoring points as present in current systems: by extending monitoring to all system layers, it is possible to exploit emergence effects contributing to improved self-awareness.

Based on this data, a more suitable (i.e. optimized) system configuration can be determined. This requires presence of adaptive components on all system layers which then supply necessary reconfiguration capabilities. Amount of reconfiguration is determined by analysis of monitoring data and evaluation against given rules or problem specifications, so-called objective functions.

Objective functions describe the wanted, optimal system behavior. Because of their often contradictory nature, weighing these objective functions is a non-trivial task, and therefore different approaches and strategies exist to achieve what is considered the "optimal" solution. This weighing process is based on application type, field of use, and additional system conditions. Further complexity is added from the fact that objective functions are not necessarily one-dimensional but can also be multi-dimensional, aggregate functions.

We developed an architecture concept, ASoCS, which specifically supports design and programming of self-optimizing parallel and distributed computer systems. As a case study for using this architecture concept, we implemented an automatic data locality optimization based on the ASoCS concept. Data locality optimization is a well-known problem within the field of parallel programming: based upon an exhaustive analysis of an application's run-time behavior, data access and distribution is remodeled manually. Once the system, application, or just the input data set changes this effort has to be repeated. With our exemplary implementation we were able to demonstrate suitability and applicability of this concept and highlight the benefits of autonomic data locality optimization.

This paper is organized as follows: in Section 2 we present related approaches regarding the introduction of autonomous features, so-called Self-X capabilities such as Self-Awareness or Self-Optimization, for system and application design. We then introduce the ASoCS framework in Section 3 followed by an exhaustive discussion of our exemplary implementation of an autonomic data locality optimizer based on ASoCS principles in Section 4. The paper concludes with Section 5.

## 2 Related Work

Data locality optimization requires sophisticated system introspection by using various monitoring techniques. That way data access patterns and frequencies can be detected, which allow better use of local memory and cache resources.

The ASoCS concept was specifically designed to aid such introspection by a powerful monitoring concept explained in Section 3. Monitoring is able to not only gather data from various system layers, but also capable of processing, interpretation, and evaluation against given objective functions already in place without necessarily requiring a central post-processing instance.

In adaptive systems, this data serves to compute a possible, more suitable system status resulting in a system reconfiguration affecting all system layers. This optimization process ideally runs automatically without human interaction, i.e. the system turns into what is typically called an *autonomic system*.

Certain aspects of the above problems have been targeted in current and previous research activities and industrial initiatives. Within this section activities involving automatic optimization are discussed.

**Initiatives Targeting Autonomic Computing.** IBM's *Autonomic Computing* (AC) initiative [15] targets automated system administration without need for human interaction. The initiative covers performance, energy efficiency, reliability, and security [34,35]. To achieve this goal, a system is partitioned into several components: *Tivoli Monitoring* [17] monitors the most important system resources. Collected data is evaluated against an optimizing objective function; this is done by the *Business Workload Manager* (BWLM).

Within their AC initiative, IBM already implemented self-repairing and query-optimizing components for their DB2 relational database. The query optimizer is part of *IBM's Learning Optimizer* (LEO) and was developed for the *Smart Managing and Resource Tuning* (SMART) database technology project. Additional servers, based on the *eLiza* project [16], are able to automatically manage computing and memory resources. Another part of this project is *Enterprise Workload Management* (eWLM), which among other features will enable performance self-optimization.

Similar initiatives exist, such as Sun Microsystems' *Network Virtualization Strategy* (N1) for dynamic allocation of network resources [28], or Hewlett-Packard's *Planetary Computing* project [4]. The latter is an architecture enabling supercomputing centers to automatically reconfigure software infrastructure, and assign needed memory and server resources.

In addition to the already mentioned industrial projects, numerous academic projects exist such as UC Texas' *EDGE architecture* [9] or the adaptive software from Michigan State University [10].

Common to all approaches is their limited scope resulting from the addressed scenario, automated system administration. Only certain applications, like database optimization or data storage, are addressed. Furthermore those approaches are usually targeting higher system levels and leave out low-level optimization on hardware level.

**Feedback Systems.** Feedback loops are e.g. used in real-time systems for industrial and automotive control: input data provided by sensors is collected and evaluated. Based on evaluation result, appropriate control takes place. Such feedback loops are prerequisites for self-organizing systems.

Current research interest is typically put on adjusting computing power and energy efficiency. Here, for instance, IPC count is used as a measure to adjust issue width and number of functional units [7]. Another approach is using miss ratio, IPC count, and jump ratio for detection of execution phases with intent to reconfigure caches and TLBs to optimally match phase requirements [3].

Another project, described in [23], targets the memory subsystem by measuring how much time elapses between consecutive accesses to an energy-aware memory. Based hereon, appropriate power mode and most energy-saving page allocation are determined.

Similar to monitoring systems, again a multitude of solutions exists, each targeting a special area. A generic and flexible architecture could be used as an universal framework for all types of feedback systems. In addition, feedback systems as described above greatly benefit from a system-wide monitoring infrastructure providing supplemental data.

**Monitoring.** Feedback systems as introduced in the previous section require techniques to gather and process system parameters to be able to create a certain sense of self-awareness. These parameters can be collected on various system levels such as lowest hardware level, driver level, OS level, or application level.

On lowest hardware level, performance counters offer some rudimentary monitoring support. They are typically used to profile an application and investigate possible application optimizations such as enhanced data layout in memory to improve cache use. Modern processor architectures offer so-called event counter registers [1,8,19,20,30,18,29]. Number and use of these registers are dependent on the individual architecture: counter registers are either bound to certain events or can be more or less freely assigned [29,19]. The majority of existing tools is based on these counter registers.

Counter-based methods typically suffer from basic limitations [37], and do not allow differentiation between events being triggered by speculative and non-speculative execution. False counts from speculation are addressed with the precise event-based sampling (PEBS) of Intel's Pentium 4 architecture [20,36]. Similar, but less complex methods are implemented in the IBM's Power architecture [30,18].

For architectures without such hardware support, monitoring can be achieved using plain software. For example, `gprof` [11] uses compiler-generated function prologues to log information such as called address and number of calls. To collect statistical information and enable mapping of execution times to functions, `gprof` periodically parses the program counter.

It is also possible to embed monitoring routines on driver level as demonstrated by the Myrinet-based Shrimp Cluster [24]. A combined hardware/software method was used on the SMiLE monitor for SCI networks [14].

Monitoring APIs as described in [32], [2], or [25,26], decouple monitoring devices from post-processing software by offering an abstract programming interface rather than directly accessing the monitoring hardware.

The drawbacks of the above examples can be subsumed as follows: not only are existing monitoring infrastructures in hardware inherently fixed, they are typically very application-specific. In addition, no standardized, generic API exists to work with monitoring infrastructures. So far, several approaches for monitoring APIs exist, However, these are typically bound to certain application such as e.g. monitoring parallel systems [25,26,27].

An approach into this direction was taken within the APART project [13] and the associated EP-Cache project [12]. These, however, only target monitoring and analysis in parallel and distributed systems. With respect to self-organizing systems, still no uniform, application-independent, and standardized monitoring API exists including reporting existing monitoring resources, permitting access to these resources, and – regarding self-organization – enabling reconfiguration.

## 3 The ASoCS Architecture Concept

The data locality optimization (DLO) implementation closely follows the ASoCS architecture concept. This concept was previously presented in [6], therefore in this section we give a quick overview over our architecture as required to understand how the exemplary DLO implementation makes use of the concept.

## 3.1 Architecture Details

Integral part of the proposed architecture concept is a novel monitoring infrastructure. As previously explained, monitoring must take place on all system layers. It furthermore must be flexible to adhere to demands of the planning stage which computes necessary system reconfiguration according to monitoring data. To achieve this, monitoring will not be a monolithic part of respective system layers. Instead, it will be split into monitoring capsules and monitoring modules.

Monitoring capsules are embedded into all system layers and provide appropriate interfacing required to dock or plug monitoring modules into the respective layer. Monitoring modules represent the monitoring functionality consisting of the sensory part (data pickup) and defined pre-processing capabilities. Splitting the monitor resources into capsule (interface) and module (functionality) enables exchange of monitoring modules, thus the monitoring infrastructure itself can be reconfigured depending on the planning stage's needs. Monitoring modules will be stored in a repository from where they can be retrieved and docked into the appropriate monitoring capsule.

A dedicated API will serve as an abstraction layer and enable access to monitoring capsules and adaptive components by other system services such as the adaptive planning stage. This stage decides which monitoring modules are loaded into their respective capsules.

Data collected by the monitoring infrastructure will be buffered in a local performance repository. This data can be merged with other local data from additional system nodes resulting in a global performance manager enabling scalability of the entire performance repository system. A dedicated query interface provides access to all local performance repositories and the monitoring infrastructure.

This query interface is used by the adaptive planning stage which evaluates collected data against objective functions to determine appropriate system reconfiguration. This evaluation is based on certain metrics quantizing system parameters and objective functions; when adapting this architecture to distinct application, it is necessary to investigate and evaluate existing metrics for use in adaptive planning and to eventually develop novel metrics aiding in the evaluation process where needed.

The overall architecture concept basically enhances and refines a common control loop scheme: a system is split into a 5-tier hierarchical scheme: the bottom layer is formed by the system hardware, with the (Real-time) Operating System (OS) including hardware drivers on top. The OS is assisted by libraries which in term are required by the compiler to finally create the desired application. Depending on its type, an application might influence only some or all levels. On each hierarchy level monitoring capsules exist. These capsules can be loaded with monitoring modules stored in a monitoring repository. Data collected and preprocessed by monitoring modules are stored in a local performance repository.

The architecture concept explicitly addresses parallel and distributed systems: local repositories of all system nodes can be retrieved by a global repository manager (GRM). This GRM is virtually centered between the local repositories of all nodes. Repositorys are accessed by the adaptive planning stage (APS) through a dedicated query interface. APS then evaluates this data and determines required reconfiguration. To enable such reconfiguration, each hierarchy level contains adaptive components which are instrumented by APS.

In Section 2 several projects and initiatives were presented which more or less exhaustively target such closed control loop systems. So far, none of them addresses a uniform architecture for self-organizing or organic architectures, but rather focuses on certain applications or system aspects. Contrary to these, our concept is application- and system-independent, and addresses all system layers rather than specific system parameters. It is furthermore applicable to single nodes as well as parallel and distributed systems.

When applying the architecture concept, methods for data evaluation and computation of system reconfiguration based on monitoring data, objective functions, and additional data such as amount of possible reconfiguration must be investigated and developed with respect to the given application scenario. This also includes evaluation of existing metrics and eventually developing of novel metrics to be able to quantize requirements and reconfiguration efforts as required by the target application scenario.

## 4 Prototypical Implementation with Data Locality as Initial Objective Function

In order to evaluate the ASoCS concept, we built a prototypical architecture and developed several components for a feedback loop with respect to data locality optimization on NUMA architectures. The reason for choosing NUMA locality as the initial objective function lies in the fact that we have been doing research work in the area of shared memory programming on top of NUMA architectures [33].

A typical NUMA (Non-Uniform Memory Access) machine is comprised of several commodity PCs or workstations connected through modern interconnection technologies. On such a machine, the main memory is distributed over the system, but globally organized into a shared virtual memory accessible from all processor nodes. Due to the different property of local and remote memory accesses, however, references targeting a remote memory can take up to two orders of magnitude longer than local accesses. As a consequence, unoptimized applications often suffer from poor data locality and the resulting high memory access latencies. As the first step towards self-organizing computer systems, we tackle this locality issue on NUMA machines.

### 4.1 Structure of the Feedback Loop

Figure 1 depicts how we map this problem onto the general framework of ASoCS. First of all, such data locality optimization needs a set of system parameters such as memory access distribution and remote access characteristics. we assume Monitoring Capsules (MC) integrated in the NUMA network interface capable of observing all remote memory transaction. On top of these MCs, we use low-level APIs for preprocessing the original monitoring data. For each MC, a Data Buffer is maintained for storing the local monitoring information. From there, data is aggregated and combined into the Global Database. This work is done by the OMIS/OCM [40] monitoring interface. Besides,

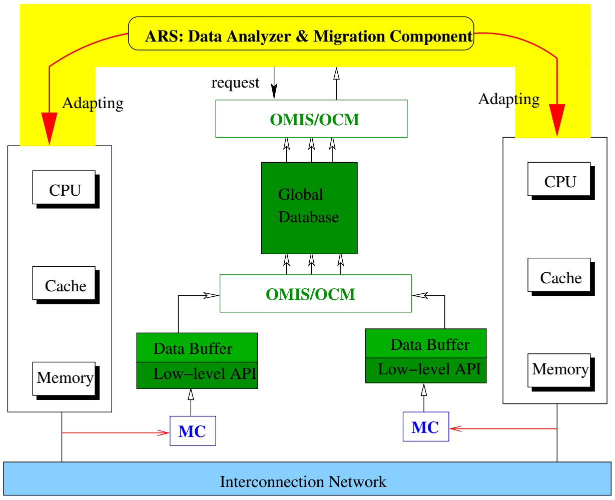


**Fig. 1.** Memory Locality Adapting on NUMA Systems

OMIS/OCM also provides a Query Interface that delivers the memory locality information to higher levels. Finally, an Adaptive Planning and an Adaptive Component are needed for self-tuning. The former is implemented with a Data Analyzer that automatically analyzes the runtime interconnection traffic and detects access hot spots, while the latter is achieved with a Migration Component that transparently modifies the data layout via moving data to its dominating node.

**Monitoring Capsule:** The design of this component is based on the SMiLE hardware monitor [22] which has been developed for observing the interconnection traffic on the SMiLE (Shared Memory in a LAN Environment) cluster connected via the Scalable Coherent Interface (SCI), a low-latency, high-bandwidth interconnection technology. For acquiring comprehensive data about the inter-node communication, we designed two analysis modules for monitoring the memory transaction: static and dynamic. The former allows to explicitly program the hardware for event triggering and action processing on special memory regions of interest, while the latter is based on histogram-driven monitoring, in which all memory transactions through the local interconnect bus are monitored in order to provide fine-grain monitoring statistics across the complete application's working set.

A hardware Monitoring Capsule is also designed to hold these analysis modules, which can be dynamically loaded into the capsule at the runtime. The capsule implements two interfaces: a link interface and a PCI interface. The former is used to snoop a local bus, which connects a single node to the actual interconnection fabric, and extract information from the transactions delivered over this bus, while the latter, an interface between the PCI bus and the monitor, offers direct access to the host node enabling the users or system software to configure the hardware monitor and read the gathered monitoring data.

**Low-level API and the Data Buffer:** In order to avoid delivering fully detailed monitoring data which is not essential for further use, we implemented a library of functions for data processing. This PAPI-like [5] standard API contains a set of routines capable of generating statistical information in the form of e.g. memory access histograms that show the number of accesses from all processor nodes to the whole working set at different granularity. These histograms can be used to detect communication bottlenecks where required data is mainly acquired from a remote processor node. For locally storing this monitoring data a Data Buffer is maintained on each processor. The buffer is organized as a histogram chain in order to enable fast searching of the needed information.

**Data Aggregation and Querying Interface:** We use the OMIS/OCM [39] monitoring system to combine monitoring data from all nodes. OMIS (On-line Monitoring Interface Specification) is a specification of an interface between a programmable on-line monitoring system for distributed computing and the tools that reside on top of it. It offers two interfaces: one for the interaction with different tools and the other for the interaction with the program and runtime system layers. OCM is an OMIS Compliant Monitoring system adhering to OMIS. It has been implemented for a series of loosely coupled environments including clusters and NoWs and has initially been designed for message passing tools. It is structured into a core and several extensions. For this work we extended OCM for providing services with respect to the access of data delivered by the Monitoring Capsules. Further, we extended OCM with a high-level Query Interface that provides the memory locality information to higher levels.

**Adaptive Planning and Component:** The remainder components include a Data Analyzer (Adaptive Planning) and a Migration Component (Adaptive Component). The former is used to analyze the monitoring data and determine whether to move a data page to another processor node. Based on the Querying Interface, the Data Analyzer is capable of accessing the memory access histograms created by the low-level API. It then compares the number of accesses to a data page from all processor nodes. If the accesses performed by a remote node exceed a predefined threshold, it is decided to move this page to the remote node. This decision is then delivered to the Migration Component, which uses system calls to move the data from the original location to the node that more requires it. This kind of adapting is performed periodically either at specified time or by synchronization points, and is held during the whole execution of the applications.

A critical issue with this approach is the migration algorithm used to make the migration decision. Commonly used page migration mechanisms are based on competitive algorithms, which migrate a page if the difference between the number of local references and the number of remote references concerning one node exceeds a predefined threshold. A similar one, called U-Mig, is also proposed within this work. As the local accesses can not be acquired by the monitors, the migration decision is based on the references performed by all remote nodes on the page under consideration. If the difference between the number of the remote accesses from the dominant node and the average remote accesses performed on the page exceeds a threshold, it is decided to move the page to the dominant remote node.

Using this algorithm, however, a correct decision can be made only after a large amount of references have been issued, resulting in late migrations and thereby a loss of performance. Therefore, We implemented several novel migration algorithms, which base their analysis on memory references to multiple shared pages, in a way that the accesses to a set of pages are combined using a weighted distribution.

An example is the so-called *W-Mig* scheme that uses the number of relative references to decide the location of a page. The number of relative memory accesses to page $P$ from node $N$ is calculated as the sum of weighted references from the same node to the pages spatially neighboring page $P$, using the following formula:

$$R_{PN} = \sum_{i=0}^{n} W_i C_i$$

In this formula, $W_i$ is a weight representing the importance of the i*th* page to page $P$ and $C_i$ is the number of references to page $i$, while $n$ is the number of pages located on node $N$. The weight is assigned according to the distance of a page to page $P$, whereby a closer page is assigned with a higher weight due to the spatial locality of memory accesses. Besides that, the neighborhood is restricted to the pages located on the same node of page $P$, since only these pages see the same remote nodes and hence their monitoring information includes the accesses from all $P$'s remote nodes. For any page located on another node, while a remote node seen by $P$ is a local node, no access information from this node can be acquired.

**Table 1.** Description of benchmark applications

| Application | Description | Working set size |
|---|---|---|
| FT | Fast Fourier Transformations | 64×64×64 |
| LU | LU-decomposition for dense matrices | 32×32×32 |
| MG | Multigrid solver | 32×32×32 |
| CG | Grid computation and communication | 1400 |
| RADIX | Integer radix sort | 262144 keys |
| OCEAN | Simulation of large scale ocean movements | 130×130 |
| WATER | Evaluation of water molecule systems | 343 molecules |
| SOR | Successive Over Relaxation | 1024 × 1024 |
| Gauss | Gaussian elimination | 512 × 512 |

To determine the location of a page, the numbers of relative references from all remote nodes are compared. If the difference between the number of relative accesses from the dominant node and the average relative accesses exceeds a threshold, it is decided to move the page to the dominant remote node. The advantage of this algorithm comes from the fact that theoretically spatially neighboring pages have similar access behavior due to the spatial locality of memory accesses. This means that if a node predominately accesses a page, it is also likely to access its neighboring pages in the same way. Therefore, the behavior of neighboring pages can be used to determine the location of this page. The benefit is that, based on the higher number of accesses, a migration decision can be made earlier.

**Summary:** Overall, we implemented a closed feedback loop for adapting the data distribution on NUMA systems. At the same time, we also established the basis framework of the proposed architecture. Most components within this framework can be applied to build other feedback loops with slight extension. For example, we are currently working on an adaptation disk for improving the cache performance. Monitoring data is acquired from performance counters; the established databases and Query Interface are directly applied; the existing Adaptive Planning component is slightly extended; and a new Adaptive Component is under development.

## 4.2 Experimental Results

The prototypical implementation of the proposed architecture has been verified with standard applications. In this subsection, we discuss the achieved results.

**Experimental Setup:** Since the hardware Monitor Capsules are not yet available, we created a simulation environment based on SIMT [38]. SIMT is a multiprocessor simulator modeling the parallel execution of shared memory applications on NUMA machines. As it aims at research work on the memory system, SIMT contains mainly mechanisms for simulating the complete memory hierarchy in detail. This includes a flexible cache simulator which models caches of arbitrary levels and various cache coherence protocols, a DSM simulator which models the management of distributed shared memories and a set of data allocation schemes, and a network mechanism modeling the interconnection traffic. For this experiment, we extended SIMT to model the Data Analyzer and the Migration Component. We also implemented the Monitoring Capsule within SIMT, including all interfaces for dealing with memory references provided by SIMT and configuration information from the user.

**Benchmark Applications:** The established prototype was evaluated with several OpenMP applications (FT, LU, MG, CG) from the NAS parallel benchmark suite [21], a few codes (RADIX, OCEAN, WATER) from the SPLASH-2 benchmark suite [41], and two self-coded kernels (SOR, GAUSS). A short description and the used working set size of these applications are shown in Table 1.

**Adaptation Effect:** First, we compared the parallel execution time with three versions: transparent (original), manual optimized, and self-tuned. The optimized version is achieved by manually specifying data allocations in the source code, based on the access pattern of applications presented by an existing data visualizer. This visualizer [31] presents the access pattern of applications in understandable graphical views, providing guidance towards an improved memory locality.

Figure 2 shows the experimental results on a 32-node NUMA system with a local access latency of 150 CPU cycles and a remote access latency of 1500 cycles. Overall, both optimized and self-tuned versions perform better than the transparent execution, with the manual optimization generally better than self-adaptation. The best performance was achieved with the manually optimized version of SOR (a small code used for iteratively solving partial differential equations), where a performance gain of factor 1.89 has been observed. This can be explained by the fact that manual optimization introduces an initial correct data layout and results in no runtime overhead. However, WATER is an exclusion, with which self-tuning outperforms manual optimization. This

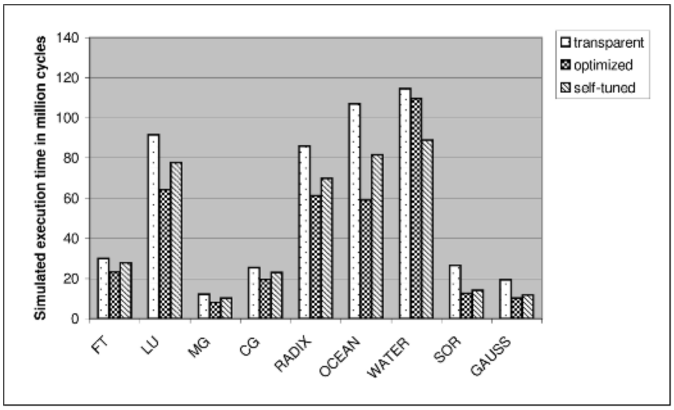


**Fig. 2.** Simulated execution time (in million CPU cycles) of tested applications in different versions

is caused by the dynamic access pattern of WATER, which renders that static optimization, which places data on fixed nodes, does not suit for the changing access behavior where data is alternately accessed by several processors.

**Comparison of Migration Algorithms:** In order to evaluate the novel migration schemes, we simulated all applications with different migration algorithms enabled. Besides U-Mig and W-Mig described in the previous section, we also simulated an L-Mig algorithm in order to examine the impact of information about local references and to evaluate the other schemes. L-Mig assumes the awareness of local accesses which can be acquired by the simulation system. In this case, the number of the dominant accesses will not be compared with the average accesses as it is the case of U-Mig, but with the local references.

The result is summarized in Figure 3. The y-axis gives the improvement of each migration version to the transparent version. This is calculated via dividing the execution time with transparent version by the execution time with a kind of migration enabled.

Examining U-Mig and W-Mig it can be observed that, as expected, W-Mig performs better in case of CG, OCEAN, WATER, and GAUSS. For others, both algorithms behave similarly. The gain in performance with W-Mig is caused by more migrations which were shown by the number of migrations provided by the simulation system. It was also found that these migrations are performed in the earlier phase of the program's execution. Programs thereby benefit from the local references that would be remote if no migration was performed, despite the overhead introduced by the migrations.

Comparing U-Mig and W-Mig with L-Mig, it can be noted that the distance between the results of migration with or without local access information is insignificant. In some cases, like for RADIX, W-Mig is even better. According to the migration behavior shown by the simulator, both U-Mig and W-Mig rarely migrate a page mainly accessed by the local node to a remote node, even though the information about local references is not available. Hence, they introduce comparable performance to those algorithms which have knowledge about local accesses.

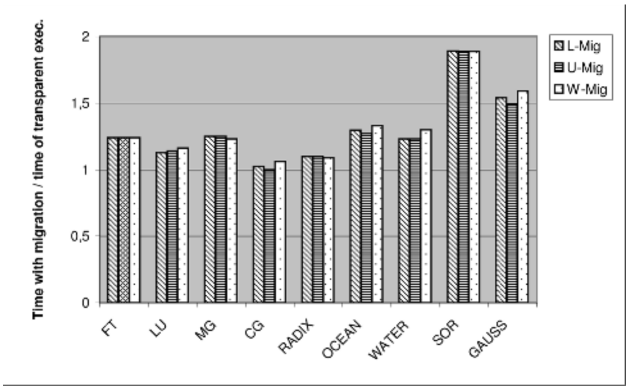


**Fig. 3.** Comparison between different migration algorithms

**Threshold Adapting:** As mentioned, we use a threshold to determine whether a page should be migrated. However, it is difficult to choose an adequate threshold for all applications. Depending on the access pattern of individual application, this threshold can be large or small in order to make correct migrations without causing performance loss. In this case, we deploy an adaptive threshold which can be dynamically adjusted according to the runtime execution behavior: if excessive migrations are performed, the threshold is lowered; if excessive remote accesses are observed, the threshold is increased.

In order to evaluate this approach, we tested the execution behavior of several applications with different thresholds. For comparison, we selected other three constant thresholds which are individually defined as 1.5, 2, and 3 factors of the average references performed on a page. Figure 4 presents the experimental results and illustrates the simulated execution time versus the threshold.

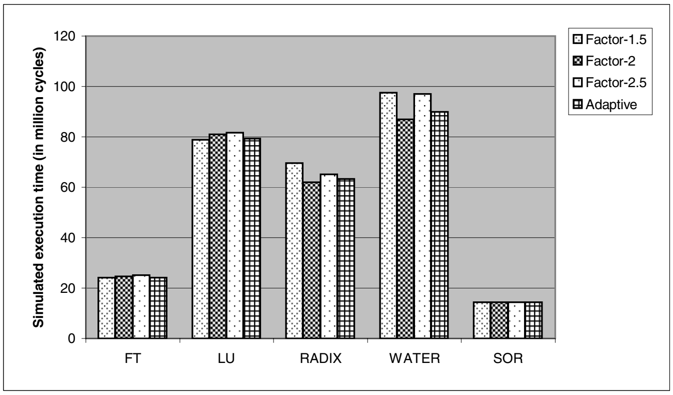


**Fig. 4.** Execution time of applications with adaptation using different thresholds.

Examining the constant thresholds, it can be seen that applications behave differently, with FT and LU presenting a better performance by factor 1.5, WATER and RADIX a better performance by factor 2, and SOR showing no significant change with varying factors. This result means that no constant threshold is optimal with respect to various applications.

For the adaptive threshold, however, it can be seen that all applications show a good performance, either the same with or only slightly worse than the best performance acquired by the optimal threshold factor for an individual application. This again proves the necessity of self-adaptation.

## 5 Conclusion

In this paper we presented an autonomic method of data locality optimization in parallel and distributed systems. This implementation was modeled upon our ASoCS architecture concept, proving the general applicability of that concept.

The need for automated optimization was discussed in the Section 1 where it was shown that future systems must employ self-optimization capabilities to overcome current limitations resulting from increased system complexity.

Section 2 presented an overview over related work addressing autonomic systems and how these are used to achieve self-optimization in their specific field of use. It was furthermore noted, that – different to our ASoCS concept – these existing concepts typically either address only certain aspects of autonomic computing or are inherently limited to dedicated applications and application scenarios.

In Section 3 we presented an overview over the ASoCS concept. The applicability of this concept was demonstrated by a prototypical implementation targeting data locality on NUMA systems described in 4. It was shown, how the locality optimizer was modeled after the ASoCS concept, and simulation results for various benchmark applications were presented to compare static (manual) optimization and adaptive (automatic) optimization.

The presented results are promising and show, although manual optimization is able to achieve better results in some cases, that on average adaptive optimization is already on par with manual optimization. With the help of presented and additional results we expect to further improve the adaptive optimization process.

## References

1. AMD. AMD Athlon processor, x86 Code Optimization Guide. 2002.
2. J.M. Anderson, L.M. Berc, J. Dean, S. Ghemawat, M.R. Henzinger, S.-T.A. Leung, R.L. Sites, M.T. Vandevoorde, C.A. Waldspurger, and W.E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct 1997.
3. R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the International Symposium on Microarchitecture*, Dec 2000.
4. Jamie Beckett. Scaling IT for the Planet: Creating the worldwide computing utility. `http://www.hpl.hp.com/news/2001/oct-dec/planetary.html`.

5. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, Fall 2000.
6. Rainer Buchty, Georg Acher, Jürgen Jeitner, Wolfgang Karl, Jie Tao, and Carsten Trinitis. ASoCS: An Architecture Concept for Self-optimizing Parallel and Distributed Computer Systems. *PARS Newsletter*, 22:108–117, Dec 2005. ISSN 0177-0454.
7. E. Chi, M. Salem, I. Bahar, and R. Weiss. Combining software and hardware monitoring for improved power and performance tuning. In *Boston Area Architecture Workshop (BARC) 2003*, Jun 2003.
8. Compaq Computer. Alpha 21264 Microprocessor Hardware Reference Manual.
9. Doug Burger et al. Scaling to the End of Silicon with EDGE Architectures. In *IEEE Computer*, pages 44–55, Jul 2004.
10. Philip K. McKinley et al. Composing Adaptive Software. In *IEEE Computer*, pages 55–65, Jul 2004.
11. J. Fenlason and R. Stallman. GNU gprof: The GNU Profiler. 1997.
12. Michael Gerndt and Wolfgang Karl. EP-Cache. February 2005. `http://wwwbode.cs.tum.edu/ gerndt/home/Research/EP-Cache/EPcache.htm`.
13. APART IST Working Group. Automatic Performance Analysis: Real Tools. 2004. `http://www.kfa-juelich.de/zam/RD/coop/apart/`.
14. Robert Hockauf, Wolfgang Karl, Markus Leberecht, Michael Oberhuber, and Michael Wagner. Exploiting Spatial and Temporal Locality of Accesses: A New Hardware-Based Monitoring Approach for DSM Systems. In David Pritchard and Jeff Reeve, editors, *Euro-Par'98 Parallel Processing, 4th International Euro-Par Conference, Southampton, UK, September 1-4, 1998 Proceedings*, volume 1470 of *Lecture Notes in Computer Science*, Berlin, September 1998. Springer Verlag.
15. Paul Horn. IBM's Perspective on the State of Information Technology. `http://www.research.ibm.com/autonomic/manifesto`.
16. IBM. Autonomic Computing Web Page. `http://www.ibm.com/servers/autonomic/`.
17. IBM. Tivoli Software Web Page. `http://www.ibm.com/software/tivoli/`.
18. IBM. PowerPC 740/PowerPC 750 RISC Microprocessor User's Manual. 1999.
19. Intel. Intel Itanium Architecture Software Developer's Manual. 2000.
20. Intel. Intel Architecture Software Developer's Manual Volume 3: System programming Guide. 2002.
21. H. Jin, M. Frumkin, and J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. Technical Report NAS-99-011, NASA Ames Research Center, October 1999.
22. W. Karl, M. Leberecht, and M. Oberhuber. SCI Monitoring Hardware and Software: Supporting Performance Evaluation and Debugging. In *SCI Scalable Coherent Interface Architecture and Software for High-Performance Compute Clusters*, volume 1734 of Lecture Notes in Computer Science, chapter 24, pages 417–432. Springer-Verlag, Berlin, 1999.
23. A.R. Lebeck, X. Fan, H Zeng, and C.S. Ellis. Power-aware page allocation. In *Proceedings of ASPLOS IX*, Nov 2000.
24. C. Liao, M. Martonosi, and D.W. Clark. Performance monitoring in a myrinet-connected shrimp cluster. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (Sigmetrics'98)*, Aug 1998.
25. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. *OMIS – On-line Monitoring Interface Specification (Version 2.0)*. Shaker Verlag, Aachen, Germany, 1997. ISBN 3-8265-3035-7.

26. Thomas Ludwig and Roland Wismüller. OMIS 2.0 — A Universal Interface for Monitoring Systems. In M. Bubak, J. Dongarra, and J. Wasniewski, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1332 of *Lecture Notes in Computer Science*, pages 267–276, November 1997.
27. Thomas Ludwig, Roland Wismüller, and Arndt Bode. Interoperable Tools based on OMIS. In *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98*, page 155, Welches, OR, USA, August 1998. ACM Press. `http://www.acm.org/pubs/citations/proceedings/metrics/281035/p155-ludwig/`.
28. Scott McNealy, Greg Papadopoulos, and Jonathan Schwartz. N1: Revolutionary IT Architecture for Business. `http://wwws.sun.com/software/solutions/n1`.
29. Sun Microsystems. Ultra-SPARC IIi User's Manual. 1997.
30. Motorola. MPC7450 RISC Microprocessor Familiy User's Manual. 2001.
31. T. Mu, J. Tao, M. Schulz, and S. A. McKee. Interactive Locality Optimization on NUMA Architectures. In *Proceedings of the ACM Symposium on Software Visualization*, San Diego, USA, June 2003.
32. P.J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Depense HPCMP User Group Conference*, Jun 1999.
33. M. Schulz, J. Tao, C. Trinitis, and W. Karl. SMiLE: An Integrated, Multi-paradigm Software Infrastructure for SCI-based Clusters. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 247–254, Berlin, Germany, May 2002.
34. InstallShield Software. Web Page. `http://www.installshield.com/`.
35. ZeroG Software. Web Page. `http://www.zerog.com/`.
36. B. Sprunt. Pentium 4 performance-monitoring features. In *IEEE Micro*, pages 72–82, Jul/Aug 2002.
37. B. Sprunt. The basics of performance-monitoring hardware. In *IEEE Micro*, pages 64–71, Jul/Aug 2002.
38. Jie Tao, Martin Schulz, and Wolfgang Karl. A Simulation Tool for Evaluating Shared Memory Systems. In *Proceedings of the 36th Annual Simulation Symposium*, Hyatt Orlando, Florida, April 2003. To be appear.
39. R. Wismüller. Interoperability Support in the Distributed Monitoring System OCM. In *Proceedings 3rd International Conference on Parallel Processing and Applied Mathematics - PPAM'99*, pages 77–91, Kazimierz Dolny, Poland, September 1999.
40. R. Wismuller, J. Trinitis, and T. Ludwig. OCM - A Monitoring System for Interoperable Tools. In *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98*, page 149, Welches, OR, USA, Aug 1998. ACM Press.
41. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

# A Bio-inspired Approach for Self-protecting an Organic Middleware with Artificial Antibodies

Andreas Pietzowski, Benjamin Satzger, Wolfgang Trumler, and Theo Ungerer

Institute of Computer Science – University of Augsburg
86159 Augsburg, Germany
{pietzowski, satzger, trumler, ungerer}@informatik.uni-augsburg.de

**Abstract.** Our human body is well protected by antibodies from our biological immune system. This protection system matured over millions of years and has proven its functionality. In our research we are going to transfer some techniques of a biological immune system to a computer based environment. Our goal is to design a self-protecting middleware which is not vulnerable to malicious events. First off this paper proposes an artificial immune system and evaluates optimal parameter settings. This shows the correlation between the size of a system and the length of the receptors used within antibodies for an efficient detection. Our tests showed that the recognition rate of unknown malicious objects can reach up to 99%. Further on we describe the integration of the immune system into our organic middleware OC$\mu$ and afterwards we propose techniques to minimize the memory space needed for storing the antibodies and to speedup the time needed for detecting malicious messages. We obtained a space minimization by 30% and gained a speedup of 30 with execution time optimization.

## 1 Introduction

To secure computer systems, current protection applications (e.g. virus scanners) have to be aware of signatures from viruses or worms that occurred previously. Due to that restriction they cannot recognize or handle brand new intruders that are not already stored in a database. Computer immunology opens up new ways and methods to recognize new intrusions like our biological immune system does [2] and fits well into the research fields of autonomic [12] and organic computing [13] that explore self-organization techniques to achieve computing systems that are self-configuring, self-optimizing, self-healing, and self-protecting. In our organic ubiquitous middleware research [16] we investigate self-protection techniques to cope with intentionally or unintentionally malicious peers or services. The biologically inspired technique of computer immunology extracts ideas from our human immune system to develop an artificial counterpart [4]. Thus the following approach is a first protection step which can detect intrusions within a system with high message activity in a fast way. The main goal is to develop a whole self-protecting system like our biological immune system which is permissive to good-natured middleware services and messages but can detect appearing malicious events. This immune system is implemented in our organic middleware to evaluate its self-protection behavior in a real environment.

The next section confronts the biological with the artificial immune system and shows the different preferences and their effectiveness. We continue with optimizing the system as well for memory requirements as also for execution time minimization. Subsequently we discuss the implementation of the self protecting system and its components in the organic middleware. In the end we mention related researches and end up with conclusions and future work.

## 2 Transition from Biological to Artificial Immune System

Lets first have a look at the functionality of the biological immune system which is present in every animal and human being. The basic requirement of such an immune system is to distinguish between harmless objects called *selfs* and harmful objects or antigens called *non-selfs* [9]. Matching works due to different protein patterns on the surface of the unknown objects and the antibodies. In our body we have an instance named *thymus* which is aware of all selfs known in the body. Antibodies are continuously created with random protein surfaces and get singled out with *negative selection.* This means if such a cell is activated in the thymus it will be destroyed [6] otherwise it will be released to the body to protect it against a specific type of intruder. This biological immune system is extremely distributed besides the centralized work of the thymus [15].

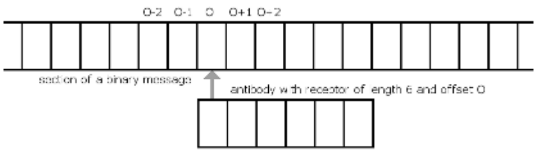

O-2 O-1 O O+1 O-2
section of a binary message
antibody with receptor of length 6 and offset O

**Fig. 1.** Example how an antibody docks to a message and starts comparison

In a computer-based immune system the working domain concerns bit strings instead of proteins. The goal in an artificial immune system is also to distinguish between foreign non-self messages and messages which are tolerated in the system because they belong to self. It also should be distributed that the middleware does not end up unprotected if one central detection node in the system gets disabled or unreachable. To avoid this we abided by the highly distributed biological paradigm. Thus we developed antibodies which are represented by a short bit string of length $R$ which embodies the receptor. Additionally the antibodies have a specific offset where they start their comparison to the messages (see figure 1). Instead of waiting for all newly created antibodies until they perambulated the thymus we preferred to generate them in a structured way. This works because the middleware is aware of all its self messages and thus it can create all receptor patterns not used by any self message at a specific offset.

## 3 The Design of the Artificial Immune System

With the approach described in the previous section we have different values which have to be considered or adjusted to optimize the recognition of non-selfs in a decentralized system:

– The length of the self messages
– The receptor length of the antibodies
– The amount of different offsets among all antibodies where they start the comparison
– The choice and amount of antibodies distributed in the system and located at a node in the system

To evaluate the immune system approach we implemented the artificial immune system into our middleware OC$\mu$ [16]. We also separated the key components from the system and built a simulation environment for faster evaluations. This simulation environment provides the possibility to generate a specific amount of selfs and antibodies with a given or calculated receptor length.

### 3.1 The Length of the Messages

Instead of grappling around with messages of different and maybe infinite length we transformed all the messages appearing in the system to a unified length by using a hash algorithm. Now the optimal length of the resulting hash value has to be determined and the most appropriate hash algorithm has to be chosen.

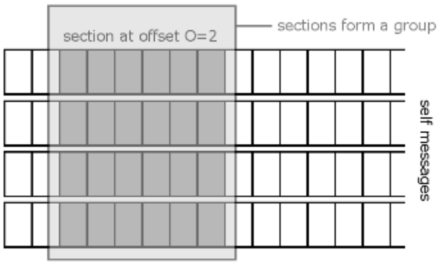


**Fig. 2.** Sections within some hashed self messages when a receptor length of six bits at offset 2 is used

In our system we have $S$ self messages. Antibodies always compare themself to the hashed incoming messages at $R$ contiguous bits, similar to [9], but always starting at a specific offset $O$ (see figure 1) and always comparing the whole receptor.

We regard antibody-specific sections within the set of all self messages, where a section is defined as the part an antibody should compare to, i.e. the $R$ contiguous bits of the antibody starting at its specific offset $O$. The sections of all hashed self messages with the same offset and length form a group (see figure 2). All the hashed self messages can be grouped in $P$ groups - namely submessages of length $R$ starting at position $O$ from the hashed self messages. We only consider one of these groups because every group behaves identical. Thus one group consists of $S$ messages of random bit sets of length $R$ and the maximum amount of antibodies which can be created in this group is limited by the usage $U$ of different bit sets in a group.

Because the system is aware of all the harmless self messages it is able to analyze the binary distribution within the different groups. The following pseudo algorithm shows how the possible antibodies can be created for a specific offset:

**Algorithm 1.** Generating antibody receptors for a specific offset group

```
R = 1;
pattern = 0; // binary representation of the pattern
loop
  while $pattern < 2^R$ do
    if !groupMatches(pattern) then
      possibleAntibody(pattern); // negative selection
    end if
    pattern++;
  end while
  pattern=0;
  R++;
end loop
```

The algorithm produces all possible antibodies starting with the smallest receptor length of one bit. Because there is no upper bound for the receptor length the loop can be exited when enough receptors were created.

### 3.2 The Optimal Length of the Receptors

For determining the optimal length of the receptors we set up different simulator configurations with different amounts of self messages and tested the recognition of antibodies with different receptor lengths. The performance of the different types of antibodies is shown in figure 3 where always the maximum amount of antibodies were created in every test environment.

This experiment demonstrates that it is not efficient to choose antibodies with a fixed or even a random length for the receptors. Instead it shows that this value has always to be adjusted due to the amount of self messages known in the system. To achieve this we have to look a bit closer to some conditions. We consider a binary alphabet and hashed messages of length $L$. We use antibodies with a receptor length of $R$ bits and those receptors can be compared to the hashed messages at most at $P$ different starting positions or offsets:

$$P = 1 + L - R, \qquad R \leq L \tag{1}$$

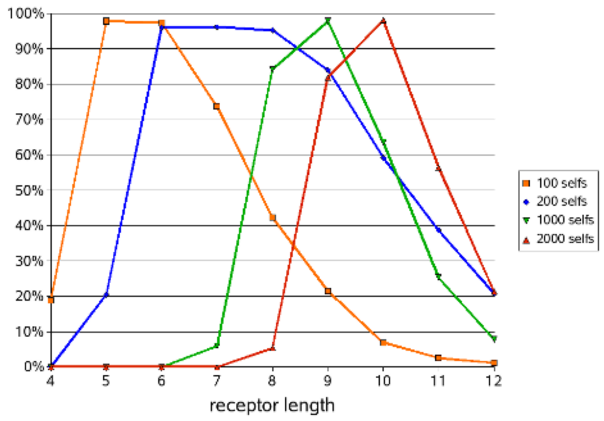


**Fig. 3.** Recognition probability with different amount of selfs and receptor lengths

If every antibody has its fixed offset where it compares itself with the hashed message we end up in $A_{all}$ different antibodies which match all possible selfs and all non-selfs:

$$A_{all} = 2^R \cdot P \tag{2}$$

Lets assume that we have $S$ different self messages in our system which should not be recognized by the immune system. In the worst case we get an amount of $A_{self}$ antibodies which should not leave the thymus because they match at least one self:

$$A_{self} = S \cdot P \tag{3}$$

This results in at least $A_{eff}$ effective antibodies:

$$A_{eff} = A_{all} - A_{self} = 2^R \cdot P - S \cdot P, \qquad R \leq L \tag{4}$$

The optimal length of the receptor can be determined when considering that the amount of antibodies should be equal or smaller than the amount of selfs.

$$A_{eff} \leq S \tag{5}$$

$$2^R \leq S \cdot \frac{1+P}{P} \tag{6}$$

$$R \lesssim \lfloor log_2(S) \rfloor \tag{7}$$

This shows that the best length for the receptors should always be smaller than the binary logarithm of the amount of known self messages[1] but at least long enough to get a group usage below 100%. The group usage is the percentage value which embodies how many different bit patterns appear in that group with respect to all possible bit patterns.

[1] As the receptor length has to be whole-numbered we always use the next lower integer as result of the binary logarithm.

Because the system is aware of all the harmless self messages it is able to analyze the binary distribution within the different groups. Thus a better length can be calculated because the mathematical calculation always assumes the mean probability which does not always match with the reality. We introduced a threshold for the group usage to calculate a good length for the receptors. Different tests showed that the recognition of non-selfs is best (with also low memory usage) if using a receptor length at which the group usage is slightly below 95%. In any other way there are too many antibodies to generate or not enough for a sufficient recognition rate.

## 3.3 The Amount of Offsets

In our research we tested different kinds of offsets which were used to match the antibodies with a hashed message to detect malicious objects. The easiest way is to define only a single offset and generate antibodies with receptors of length $R$ which compare to the hashed message only at this specific offset. But this is exactly the same as if the hashed messages would have the same size than the receptors and the whole incoming messages always compares to the antibodies. This would result in a very bad recognition of non-selfs as the usage of this group corresponds to the probability for not recognizing the non-selfs.

In contrast in another test environment we designed the antibodies to use all possible offsets. This results in fully overlapping groups and thus less hashed messages can be generated that do not match any antibody. And as expected our tests showed that the more we overlapped the groups the better the recognition of non-self messages worked out. the best recognition was observed when using all possible offsets (from offset 0 to $L - R$) and always the same amount of antibodies in each offset group.

## 3.4 The Right Choice of Antibodies

When not generating all possible antibodies, it is important how many antibodies to create and at which offsets. Because the system is aware of all selfs and thus it is aware of all used receptor patters in all groups we can generate the antibodies in a structured way. We tested three different methods for the creation of antibodies:

– Generate antibodies with fewest group usage first
– Generate antibodies with highest group usage first
– Generate an equal amount of antibodies in every group

In our tests we set up a system containing 1000 self messages. Thus we generated different amounts of antibodies with the three methods mentioned above. The result of recognizing malicious messages is shown in figure 4. First we generated antibodies out of those groups which have the fewest usage but this resulted in a bad recognition of non-selfs – at least when generating only 1000 antibodies. A similar bad behavior was observed when choosing the second method where those antibodies were generated first which had the offsets of those groups with the highest usage of bit patterns. Apparently, the best method – also when generating only a few antibodies – is to generate an equal amount of antibodies in every group.

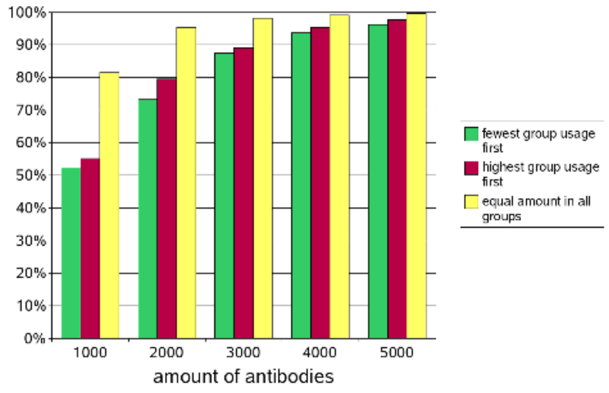


**Fig. 4.** Recognition with different methods of generating antibodies

This experiment also showed that the recognition of non-selfs leads up to 99.3% when using 5000 antibodies of a receptor length of nine bits and equal distributed offsets among all antibodies in an environment of 1000 self messages.

## 4 Optimizations

### 4.1 Optimizing Memory Requirements by Merging Antibodies

At a first thought one would suggest to store the self messages instead of the antibodies and compare all incoming messages with the self set. This is correct when the system has only a few self messages but this research examines systems with high activity and thus many self messages. When looking closer to this issue the usage of antibodies should be preferred. In the system setup mentioned in section 3.4 it would need $1000 \cdot 128bit = 15.6kb$ to store all known selfs. Compared to the amount of 5000 antibodies with a receptor length of nine bits and 128 different offsets this results in only $5000 \cdot (9bit + 7bit) = 9.77kb$ memory to store these antibodies and still reach a recognition probability of 99,3%. The seven bits result from the storage of the specific offset which can take values from 0 to $128 - R$. Another benefit of choosing the antibodies is that only short receptors have to be compared instead of the whole hashed messages of 128 bits in length.

We propose a technique for memory requirement minimization which does not advance the recognition probability but minimizes the amount of antibodies to be stored in the system. Two antibodies with the same offset and with an identical receptor can be merged to one and the same antibody which is trivial.

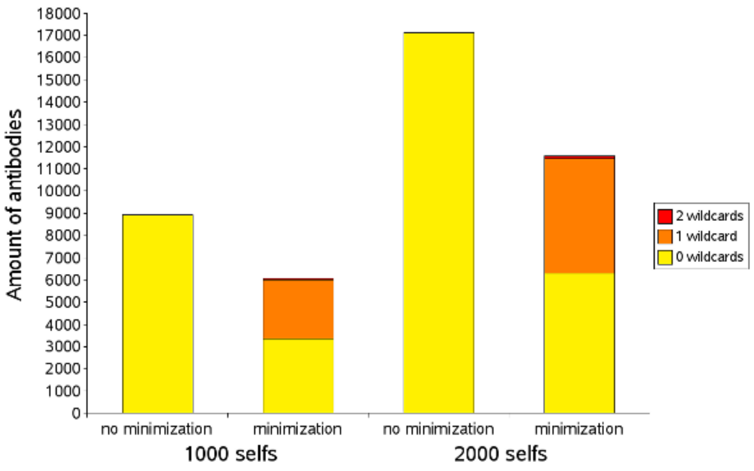


**Fig. 5.** Mimizing the amount of antibodies by merging at specific bits on the receptors

But also antibodies with the same offset and different receptors can be merged to one antibody if they differ at only one position on the receptors. In that case a new antibody can be created with a *wildcard* at this specific position which replaces the two antibodies. At compare time this wildcard position does not have to be compared because it treats both possible values – 0 and 1 – as a valid bit yielding even in a small speedup. Additionally antibodies with wildcards can be merged together when two receptors have the same wildcard positions and differ at exactly one further position.

In our tests we were able to eliminate about 30% of the antibodies by merging them together. We evaluated the optimization with different amounts of randomly generated antibodies and figure 5 shows how many antibodies could be merged to antibodies with one and two wildcard positions when using a setup of 1000 and 2000 antibodies. Unfortunately in our test cases we were never able to produce antibodies with three or more wildcards because this is very difficult when the group usage of selfs exceeds some specific value.

## 4.2 Optimizing Execution Time

In a human body the comparison of antibodies against appearing objects takes place in a three-dimensional environment and antibodies check themselves randomly all the time. Due to the one-dimensional architecture of computer memory we have to cope with some limitations in that aspect [2]. In our test cases we stored all antibodies in an array and thus the comparison against incoming messages takes some time - more precisely if we have $A$ antibodies with a receptor length $R$ we have to cope with a time complexity of $O(A \cdot R)$ to compare an incoming message to all antibodies. That is not very efficient if we have a lot of different antibodies. Comparison would slow down the communication between applications.

To achieve a more efficient comparison a new storage mechanism has to be devised for the antibodies. Many antibodies correspond to other antibodies in specific parts of their receptor. So the idea grew to store the receptors in a binary tree. Thus the comparison can start at the root of the tree and follow the way down according to the binary pattern. If there is a complete way down to the bottom of the tree one antibody stored in that tree matched the incoming pattern (see figure 6).

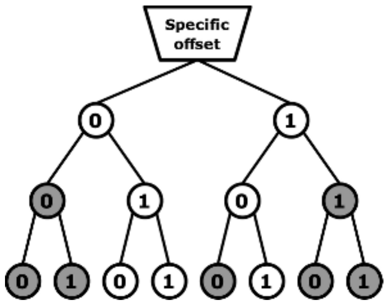


**Fig. 6.** Representation of a receptor tree that contains the receptors 010, 011 and 101. White nodes represent enabled values at specific positions, gray nodes are values that are not available in any receptor.

This approach implements a separate perfect binary tree (without a root node) of depth $R$ for each offset known in the system. A tree is analogous to the receptors of the antibodies at one specific offset if read from top to bottom and only traverses enabled nodes (enabled nodes are white colored in figure 6). Because every tree has to be perfect and every node consists of one boolean value a memory usage of $(2^{R+1} - 2)$ bits is necessary for every tree with an offset in $P$. To compare incoming messages against the trees each tree has to be passed according to the incoming pattern at the corresponding offset. If there is no enabled path from the root to the bottom of the tree the next tree comparison takes place which embodies the antibodies at another offset. If one tree matches at one specific path the message is treated as malicious. This comparison results in a time complexity of $O(P \cdot R)$ and because in general $P \ll A$ this approach is much faster than comparing all antibodies with a complexity of $O(A \cdot R)$. Further more $P$ and $R$ are fixed for a group of trees this results in $O(1)$ for comparing incoming messages to all stored antibodies. In our tests the tree comparison resulted in a 30 times faster comparison than using the linear comparison algorithm.

## 5 Integration in the Organic Middleware OC$\mu$

The Organic Middleware OC$\mu$ (formerly known as AMUN) [16] is a message-based middleware based on the peer-to-peer system JXTA (see figure 7). To realize the self-x properties from organic computing the middleware is extended by an organic manager and interfaces which add monitoring capabilities. Because of that manager the system is categorized as an organic middleware. Different services are started among the nodes of the system depending on a given configuration. Those services are not fixed to a node but can be relocated automatically by the middleware for self-optimizing the whole system if necessary. All services contain an unique ID and communicate through messages. All messages which are sent by a node always preambulate the event dispatcher where different message monitors can be placed either at the incoming or the outgoing interface. Message monitors can operate on incoming or outgoing messages without any control by the services (i.e. in this case a message monitor checks the messages if they are self or non-self). The goal of this work is to integrate the artificial immune system to prevent the system from malicious intrusive services or messages. The intrusion detection system in OC$\mu$ currently consists of three components: The *thymus*, the *immune instance*, and the *intrusion detection monitor*.

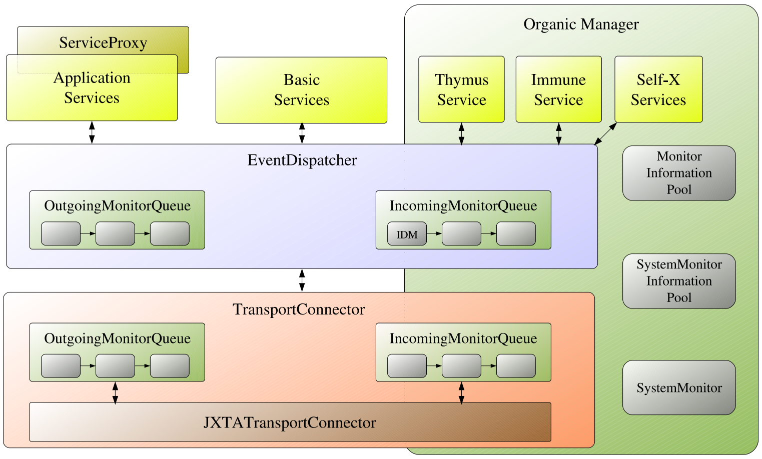


**Fig. 7.** The architecture of the organic middleware OC$\mu$

## 5.1 The Thymus

The thymus is realized as a service in the middleware and is the cornerstone of the whole immune system. It has the ability to ask its local services for all message types known from the configuration and also to ask remote thymuses about the message types used on that node. This implicates that a thymus service also has the functionality to spread the known message types to other nodes as they are requested be remote thymuses. Another task of a thymus service is the proper generation of antibodies with respect to the systemwide known message types and to suggest a suitable receptor length of the antibodies to achieve an optimal recognition rate of intrusive message types. Generating antibodies is very time consuming. Thus the most thymuses in the middleware just send all their local message types to some dedicated thymus services which generate all necessary antibodies. Therefore every node in the middleware should run an instance of the thymus service - at least for sending the known selfs.

### 5.2 The Immune Instance

The component called immune service embodies the instance which decides if some intruding message should be accepted or treated as non-self. Therefore this module is also realized as a middleware service. It is able to receive antibodies which are generated and spread by the dedicated thymuses around the middleware. Those antibodies are internally stored in an array for comparison to incoming message. In the next section we will show how to speed up the comparison when using a binary tree for storage.

### 5.3 The Intrusion Detection Monitor

The last component in this architecture is the intrusion detection monitor (IDM) which is placed in front of the incoming message monitor queue within the event dispatcher (see figure 7). Every incoming message is checked automatically for its validity. To realize this the monitor sends the message type and the ID of the sending service to the immune service. This service decides if the message either belongs to self and should be trusted or classifies it as non-self because it may be harmful to the middleware and its functionality. As for now the intrusion detection monitor only checks for malicious messages and can block them but it does not defend the intrusion yet. Tracking back a malicious message to its originator results in detection of a harmful service.

### 5.4 Efficiency of the Artificial Immune System

There are two issues which have to be considered for evaluating the efficiency. On the one hand there is the time needed for generating the antibodies and on the other hand the time needed to check an incoming message. Generating antibodies is time consuming anyway and this work is only done on dedicated nodes. Due to that fact the job of the thymus will not be measured. Every incoming message on a specific node has to be categorized as good or bad. This check might also be time consuming and may affect the response time of the system. Due to that issue we measured the delay which appears when sending all messages through the monitor. When the detection monitor is enabled and with the optimizations described in the previous section, a node in the middleware only needed about 0.3% to 1.3% more time to process any messages than the normal message handling in OC$\mu$ without any checks. These values were measured with 5000 selfs, about 15,000 antibodies and 10,000 randomly generated messages in a running OC$\mu$ system.

## 6 Related Work

Non-immunological detection systems are mostly signature based or at least they use the information gained from previously detected anomalies. For example current virus scanners contain signatures from known viruses and scan files for a match with those signatures. But there are also approaches which can detect unknown or new intrusions because of not comparing to known patterns but the other way round – using the negative or clonal selection.

Hofmeyr and Forrest developed an architecture for an artificial immune system to detect intrusions in a network [9]. They also modeled their immune system on the biological immune system described in section 2. The antibodies and the messages in their system consist of bit strings with a fixed length and the system compares the messages appearing in the network to all created antibodies. If $r$ contiguous bits are identical the message is treated as non-self otherwise the messages is accepted by the immune system. The number $r$ has to be adjusted in their system to tune the recognition probability and can have values from one bit to the entire length of the messages [10]. Our approach follows a similar way to design the antibodies but shows the basic correlation between the amount of self-messages and the length of the receptors needed to get an optimal recognition probability in any domain.

A further approach was made at the Swiss Federal Institute of Technology in Lausanne where an artificial immune system was proposed which can detect misbehavior in mobile ad-hoc networks because the dynamic source routing in those networks can be used by malicious nodes to vulnerate the system. They generated random antibodies using negative and clonal selection. With clonal selection antibodies generate copies of themselves with similar but not strictly identical receptors [1]. Our immune system aims at another application domain, i.e. to protect our organic middleware from intrusive services. Plus, in our artificial immune system we are aware of all selfs used in the system and thus we have the ability to know what belongs to self and non-self.

## 7 Conclusions and Future Work

The tests showed that the recognition rate was very good (up to 99.3% in some test cases) when choosing a suitable receptor length and an appropriate amount of antibodies. We also looked closer at the speed of detecting selfs and non-selfs and found a way to speed up the comparison enormously by using a binary tree.

The organic middleware [16] is a message oriented middleware based on a peer-to-peer system. To realize the self-x properties from organic computing paradigms the middleware is extended by an organic manager and interfaces which add monitoring capabilities. As for now we implemented the automatic generation and distribution of the antibodies in the middleware environment and we also proposed some improvements and optimizations which lead to lower memory usage and a faster detection.

In a next step we integrated a reaction system around the detection mechanism at an early stage. Thus the middleware will be able to deactivate malicious services or shut down infected nodes in the network to prevent other nodes from the intrusion. This leads to a so-called self-healing mechanism which recovers the system after a successful defeating of malicious events.

## References

1. Jean-Yves Le Boudec and Slaviša Sarafijanović. An Artificial Immune System Approach to Misbehavior Detection in Mobile Ad-hoc Networks. In *In Proceedings of Bio-ADIT – The First International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pages 96–111, Lausanne, Switzerland, January 2004.
2. Mark Burgess. Computer Immunology. In *Twelfth Systems Administration Conference (LISA '98)*, Boston, Massachusetts, December 1998.
3. Robert M. Corless, David J. Jeffrey, and Donald E. Knuth. A Sequence of Series for the Lambert W Function. In *International Symposium on Symbolic and Algebraic Computation*, pages 197–204, Maui, Hawaii, USA, 1997. ACM.
4. L. N. de Castro and J. Timmis. Artificial Immune Systems: A Novel Paradigm to Pattern Recognition. In J. M. Corchado, L. Alonso, and C. Fyfe, editors, *Artificial Neural Networks in Pattern Recognition*, pages 67–84, SOCO-2002, University of Paisley, UK, 2002.
5. Leandro N. de Castro and Fernando J. von Zuben. *Biologically Inspired Computing.* Idea Group Publishing, 2005.
6. Patrik D'haeseleer. An Immunological Approach to Change Detection: Theoretical Results. In *9th IEEE Computer Security Foundations Workshop*, Dromquinna Manor, County Kerry, Ireland, June 1996. IEEE.
7. John M. Hall and Deborah A. Frincke. An Architecture for Intrusion Detection Modeled After the Human Immune System. In *Proceedings of the International Conference on Computer, Communication and Control Technologies*, volume 6, pages 75–78, 2003.
8. Emma Hart and Jonathan Timmis. Application Areas of AIS: The Past, The Present and The Future. In *4th International Conference on Artificial Immune Systems (ICARIS 2005*, volume LNCS, pages 483–497. Springer-Verlag, 2005.
9. Steven A. Hofmeyr and Stephanie Forrest. Architecture for an Artificial Immune System. In *Evolutionary Computation 8*, number 4, pages 45–68, Massachusetts Institute of Technology, 2000.
10. Steven Andrew Hofmeyr. *An Immunological Model of Distributed Detection and Its Application to Computer Security.* PhD thesis, University of New Mexico, May 1999.
11. Zhou Ji and Dipankar Dasgupta. Estimating the Detector Coverage in a Negative Selection Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 281–288, Washington DC, June 2005. ACM.
12. Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer Society*, pages 41–50, January 2003.
13. Christian Müller-Schloer. Organic Computing Initiative. Published as PDF on `http://www.informatik.uni-augsburg.de/lehrstuehle/sik/research/organiccomputing/download/OC-english.pdf`, April 2004.

14. Ronald Rivest. The MD5 Message-Digest Algorithm. Technical Report Request for Comments: 1321, Internet Engineering Task Force (IETF), April 1992.
15. Anil Somayaji, Steven Hofmeyr, and Stefanie Forrest. Principles of a Computer Immune System. In *New Security Paradigms Workshop*, pages 75–82, Cumbria, UK, 1997. ACM.
16. Wolfgang Trumler, Faruk Bagci, Jan Petzold, and Theo Ungerer. AMUN - autonomic middleware for ubiquitous environments applied to the smart doorplate. *Advanced Engineering Informatics*, (19):243–252, April 2005.

# Autonomic Management of Edge Servers

Mikael Desertot[1, 2], Clement Escoffier[1], Philippe Lalanda[1], and Didier Donsez[1]

[1] Laboratoire LSR-IMAG, 220 rue de la Chimie,
Domaine Universitaire, BP 53
F-38041 Grenoble, Cedex 9, France
{mikael.desertot, clement.escoffier philippe.lalanda, didier.donsez}@imag.fr
http://www-adele.imag.fr
[2] Bull SA
1, rue de Provence - BP 208
F-38432 Echirolles Cedex - France

**Abstract.** Delivering innovative Internet services raises numerous business and technical challenges for providers. It actually requires building and managing complex, distributed architectures in order to reach the quality of service that is needed. In this paper, we argue that using edge computing in the domain of Internet services has a number of advantages. However, this approach relies on complex and hard to administrate environments. We believe that autonomic computing techniques constitute a key element for the dynamic management of edge servers. In the paper, we present an autonomic manager that meets the market needs and that has been tested in collaboration with Bull SA.

## 1 Introduction

The Internet has become a major means to share information and, more recently, to provide e-services to individuals and companies. A broad range of industries is actually depending more and more on the Web to sus pport their business processes [5][13]. Many think it represents the next wave of e-business, and when looking at the resources that companies are investing on this matter, we see that the stakes are substantial.

Unfortunately, providing services meeting the market needs is not easily achievable because services have to meet stringent requirements regarding performance, security and availability, which are nonnegotiable features. Due to the ever-growing number of users and the heterogeneity of rendering devices (laptops, PDAs, mobile phones), handle these requirements is a more and more challenging task for service providers. It actually requires building and managing complex, distributed architectures, in order to reach the needed quality of service (QoS).

The most common approach to meet these requirements is to deploy clusters on the service providers' sites. By replicating the infrastructure and by using load-balancing policies, service providers are able to ensure reasonable response times and high availability. However, this approach has at least three main drawbacks. The first one is that the expensive resources put together to make up a cluster are not used all the time. There are fully exploited only when the web site faces load peaks or flash crowds. Most of the time, a large part of the cluster is inactive. The second issue is the management complexity of such architectures. It requires a domain expert to manage load balancing and to monitor the cluster execution. Indeed, increasing the number of machines also means increasing the number of possible failures. The third problem is that clusters do not provide solution to networks congestion.

An alternative approach is to use the concept of edge computing. The purpose of this recent paradigm is to move computing resources away from centralized servers (or clusters of servers) to physical nodes close to the clients. Edge computing is gaining wider acceptance for a number of reasons such as bandwidth savings, optimal resource usage, or security and QoS improvement. At the same time, running applications at the edge is becoming possible in many fields. In the domain investigated in this paper, the concept of edge computing can be implemented by service providers through the loan of Internet Service Providers (ISP) resources. ISPs usually have their own networked infrastructure to provide their customers with robust, secure and dedicated link to the Internet. Part of this infrastructure, that is not fully used, can be loaned to service providers in order to run services (or parts of services) on machines close to their clients. The closest machines, which are on the logical edge of the ISP network, are called Edge Servers.

Applying edge computing in the domain of Internet services has a number of advantages as explained below. However, a major problem remains. Similarly to clusters, the Edge Computing approach relies on complex and hard to administrate environments. Sophisticated monitoring actions have to be frequently undertaken in order to maintain good QoS and to provide new features, regularly required by the clients.

In this paper, we argue that autonomic computing [1] techniques are a key element for the dynamic management of edge servers. Autonomic Computing is already used for clusters management, but it is possible to enlarge its use to fit Edge Computing requirements. We present an autonomic manager that meets the market needs and that has been tested in collaboration with Bull SA (see www.bull.com) through a "Video on Demand" application. The paper is organized as follows. Next section shows how the concept of edge computing can be applied to Internet services. Section 3 presents the architecture of our system. Section 4 focuses on the autonomic manager for edge computing. Section 5 presents case studies and some simulation.

## 2 Edge Computing and Internet Services

As previously mentioned, edge computing can be seen as a new business model for ISP and Content Delivery Networks (CDN). According to this model, resources may be allocated on-demand to service providers in order to face load peaks and flash crowds. These resources are distributed over the ISP/CDN backbones and are generally numerous low-cost blade servers. Those servers are preferably close to the end users to improve response time and to alleviate the ISP/CDN backbone.

Edge computing has a number of advantages [4][13]. First, it significantly reduces the amount of data to be transmitted over a network (Internet in our case) and the average distance to reach the user. This reduces transmission costs, latency and, therefore, improves QoS. This is especially true for multimedia–based services dealing with video or audio streams. Edge computing also improves security; data entering edge servers can be immediately verified and encrypted. Thus, since data go through firewalls sooner, viruses can be detected earlier.

Edge computing has been used primarily for caching purposes in multimedia–based services (see Akamaï www.akamai.com). It is, for instance, particularly suited to the video on-demand domain when blockbusters come out. In this case, provider servers are not able to deliver the required number of streams in good conditions. Caching movies close to consumers allows the delivery of high quality streams. Here, the edge server can also be used to adapt the data streams to the client rendering devices, which once again significantly saves bandwidth.

We experiment the use of edge computing in the context of Internet services. This part of the work is done in collaboration with the ObjectWeb consortium (www.objectweb.org) and the Bull company which provide the industrial use cases. More precisely, our purpose is to dynamically migrate part of the code implementing the services on edge servers when needed (load peaks, stringent real-time requirements, etc.). We also investigate the domain of networked multiplayer game. In this domain, it is impossible for a shared master server to face all the players' requests. Dynamically delegating part of the game management to servers close to users helps providing the necessary computing power and decreases the response time (critical for this domain).

Our approach is to implement services as a set of loosely coupled software components (preferably according to a three-tier architecture), that can be executed on the providers servers or on the edge servers as well. This requires having compatible runtime environments on both sides. In our project, we are using J2EE servers: the open source JOnAS server (jonas.objectweb.org) on the provider side and a standard, light-weight JOnAS version, that we have developed, on the edge servers. This light-weight server can be seen as an extensible core easy to start, configure and stop on demand (see [2] [3] for details). It dynamically provides the technical functions needed to run a specified set of components, nothing more (recall that edge servers are rented, shared machines). This work is however not in the scope of this paper.

To manage the components distribution we use an autonomic approach. The most efficient distribution actually evolves over time and depends on many complex parameters. Managing the distribution is not doable by a human operator. The purpose of the autonomic manager is therefore to dynamically and automatically place the components implementing a service on the best suited servers. The efficiency of the approach depends on the time needed to take decision and to install components. The decision software and the associated infrastructure have then to be particularly flexible and reactive. The decision software has also to determine where and how components should be migrated. Once a migration decision has been taken, it is necessary to determine which components can be migrated (some may not be replicable, or it could be not acceptable to migrate them for security or privacy purposes). Furthermore, components may depend on specific technical services for their execution. These services have to be deployed and configured on the host servers.

## 3 Global Architecture

The overall architecture is based on the generic architecture proposed by IBM [1]. As illustrated in figure 1, this architecture defines three main entities: the environment to be managed, a set of "touch points" representing access points on the environment (sensors and effectors) and an autonomic manager. The architecture dynamics is implemented through a control loop: information is captured about the environment, treated by the autonomic manager to decide which actions are to be executed on the environment.

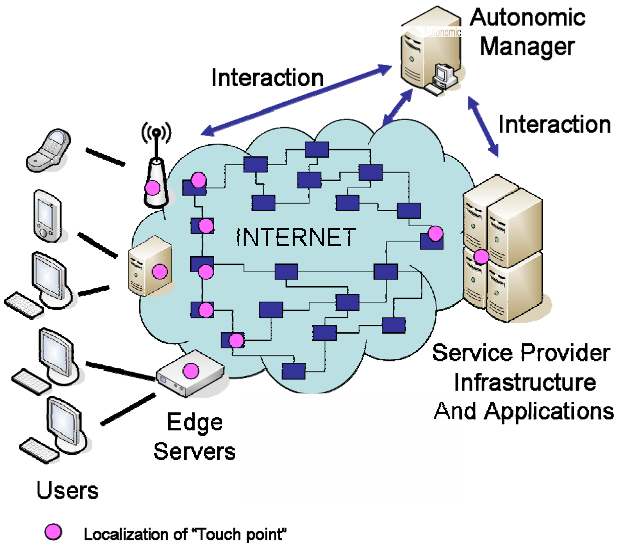

Autonomic Manager
Interaction
Interaction
INTERNET
Service Provider Infrastructure And Applications
Edge Servers
Users
Localization of "Touch point"

**Fig. 1.** Global architecture

In our system, the managed environment is the technical infrastructure of the service providers and of the ISP, including the edge servers and the networks. It should be noted that the number of entities to monitor is potentially huge. The autonomous manager should be able to deal with thousand of users and tens of edge servers. In addition, the number of users and the availability of edge servers are always evolving. Edge servers, which belong to ISP, can appear and disappear at runtime.

As previously said, the way components are distributed on edge and providers servers varies over time. It depends on the following parameters that have to be permanently monitored:

- the edge servers availability and their capacity to host additional computations without decreasing their performance,
- the renting cost of the edge servers,
- the current and incoming load of the provider servers,
- the required QoS,
- the migration (and synchronisation) cost.

In the current implementation, information data are periodically required by the autonomic manager, e.g. sent regularly by the different servers.

As illustrated in figure 1, the autonomic manager is dedicated to a single service provider (we do not tackle the case where an autonomic manager is shared and multi-operated). The autonomic manager can be on the same network as the provider servers or can be distant. Therefore it does not use the application bandwidth to retrieve environmental information and so the provider network is not impacted.

## 4 The Autonomic Manager

We have also based our work on the generic architecture defined by IBM. As illustrated on figure 2, an autonomic manager is made of five components:

– a monitor collects (or receives) information on the managed environment and builds periodical activity reports,
– an analyzer elaborates a representation of the environment's state based on the activity reports,
– a planner uses the environment model to determine and schedule actions on the environment if necessary,
– an executor is responsible for the correct realization of the scheduled actions,
– a knowledge base contains the information collected on the environment.

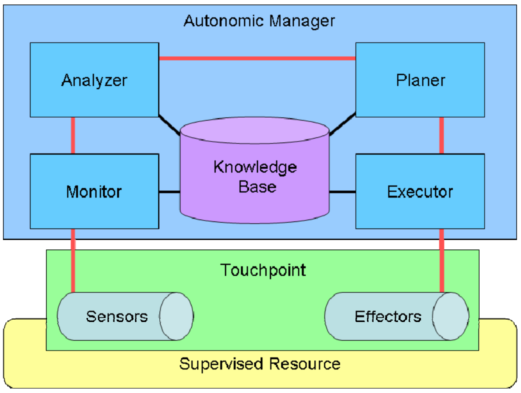


**Fig. 2.** A generic autonomic manager

We have implemented the five components of this generic architecture. As we will see in the next sections, we had to face three important issues. The first one was to feed the Monitor with data reflecting the environment in an efficient way, providing as much data as possible without being too intrusive. The second challenge was to appropriately model the data and the applications. The last issue was to find means to reason efficiently on a potentially large amount of data.

The Autonomic Manager has been implemented in Java. The communication between the five components is bidirectional and relies on method invocations (the

Autonomic Manager is not distributed). Using Java has simplified the implementation and the connections to the J2EE environment. Its dynamic loading capabilities were very interesting with respect to the management and evolution of the environment.

## 4.1 The Monitor

As previously said, the purpose of the Monitor is to collect information and synthesize it in activity reports. Information is actually collected by probes distributed on the different J2EE servers (provider or edge servers) and on the network in order to evaluate its load (and to detect possible congestions).

Communication between the probes and the autonomic manager relies on Joram, an open source implementation of the JMS standard (Java Messaging Service) available on ObjectWeb. In our context, asynchronous message-based communication actually provides reliability and flexibility to the collecting process at a reasonable cost.

Data collected by the probes are regularly sent to the autonomic manager through messages, under various topics. Depending on the polling frequency, which can be changed at run time, data may be raw or more elaborated. Finally, it has to be noted that the autonomic manager can directly ask the probes for information, if needed.

The activity reports built by the Monitor contain the data themselves and additional information including a timestamp, the location of the probe, the invocation process, etc.

## 4.2 The Knowledge Base

The knowledge base contains the data used by the analyzer to make decisions. In our case, it contains the following information:

- the current status of the monitored environment (and historical data about it),
- a J2EE compliant description of the software architecture implementing the services,
- the current distribution of the software components,
- a set of management policies, i.e. the active strategies that are applied by the decision process.

We have defined a specific Architecture Description Language (ADL) in order to describe the implementation of services in terms of connected J2EE software components. The purpose of this ADL is to reflect the structure of the applications and to provide the information necessary for distribute them dynamically (and not to generate the applications).

The ADL specifies three types of components: presentation components (servlet, jsp), business components (EJBs) and data components (database). Presentation components are those usually migrated in edge-oriented applications. Our approach allows the migration of business components (and their dependencies) and data components. However, moving data components can be complex: it actually implies the configuration of the target servers to maintain the database connections available.

For each component, functional interfaces specify the methods provided to other components and the methods required from other components. Each functional interface is defined by its name, its type (provided or required) and the signatures of its methods. Functional interfaces are Java interfaces written by the developer. Control interfaces are optional and can be used to configure the components and manage their lifecycle (actions of the autonomic manager on environment).

The ADL also includes information to guide the deployment decisions. First, each component has an attribute named *moveable* which indicates if the component can be moved to another location at run time. In some cases, components have to remain on the initial server e.g. for security reasons. Second, provided interfaces have an *accessibility* attribute which can be set to local, remote or both. On a provided interface, this attribute indicates whether the component can be accessed remotely or not (*i.e.* whether the EJB has remote or local interfaces). The *colocalized* attribute of required interfaces positioned to "yes" indicates that binding a local interface is mandatory. Indeed, in some case, required components have to be installed on the same machine as the requesters e.g. for performance reasons.

A required interface may also have property attributes which are used for the dynamic selection of components fitting the requested interface. Such information is crucial for the autonomic manager when it has to decide whether or not component has to migrate. For instance, if such a component requires a local service, the autonomic manager has to check that a component providing this service is existing or deployable on the target edge server. It is important to note that using trading instead of naming allows increasing the number of potentially useable components.

Figure 3 provides an example of architectural description including two presentation components, three logical components (business components) and two data components.

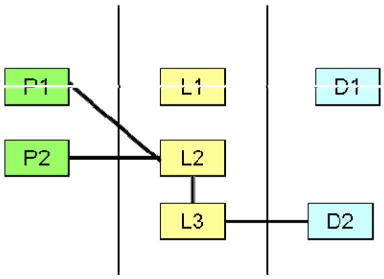
## 4.3 Analyzer and Planner

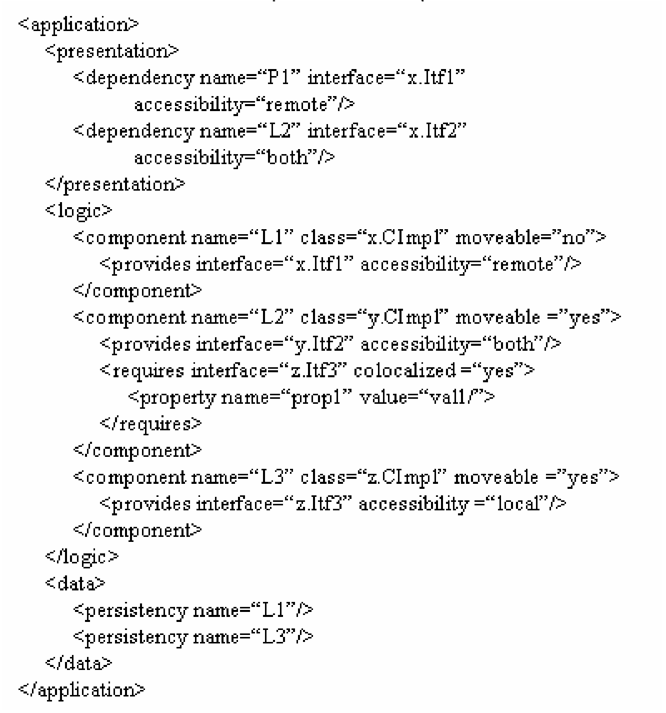
One main goal of the analyzer is to maintain a coherent representation of the global system. The planner has then to decide the actions to be undertaken in order to get the best usage of the computing infrastructure.

Both components are implemented using Event-Condition-Action (ECA) rules. This appears to be an efficient technique to deal with the large amount of data to be handled and with the non deterministic nature of the algorithms to be implemented. Managing a set of rules turned out to be rather easy. The dynamic update of the rule set was very useful when the demand evolves. In our prototype, presented in the next section, rules are expressed in Java. Presumably this is not a good solution but it was adequate for our experiment. The use of rules raises another problem. If the policies contain many rules, the system behavior is hard to analyze and, consequently, to debug.

The analyzer contains deductive rules which purpose is to build a representation of the environment. In particular, these rules have to handle the addition and withdrawal of resources (edge servers). An example of such rules is:

```
ON edge declaration
IF the new edge server is not know already
   THEN add the new edge server to the list of available resources
```





```
<application>
   <presentation>
      <dependency name="P1" interface="x.Itf1"
             accessibility="remote"/>
      <dependency name="L2" interface="x.Itf2"
             accessibility="both"/>
   </presentation>
   <logic>
      <component name="L1" class="x.CImpl" moveable="no">
         <provides interface="x.Itf1" accessibility="remote"/>
      </component>
      <component name="L2" class="y.CImpl" moveable ="yes">
         <provides interface="y.Itf2" accessibility="both"/>
         <requires interface="z.Itf3" colocalized ="yes">
            <property name="prop1" value="val1/">
         </requires>
      </component>
      <component name="L3" class="z.CImpl" moveable ="yes">
         <provides interface="z.Itf3" accessibility ="local"/>
      </component>
   </logic>
   <data>
      <persistency name="L1"/>
      <persistency name="L3"/>
   </data>
</application>
```

**Fig. 3.** An application example and corresponding ADL

The planner contains action rules which purpose is to build an ordered set of actions to be sent and executed on touch points. A common example of action rule is the deployment of a component and user redirection when a threshold is reached. For example, the deployment of a presentation component P when a number of users is reached (5000) could be written as:

***ON*** *user connection*
***IF*** *number of hosted session == 5000*
***THEN*** *deploy the presentation component P* ***and*** *redirect the user on the Edge Server*

This rule is evaluated when a user connects on the primary server. Then the planner looks at the number of hosted sessions on the service provider infrastructure. If the condition is valid, then the planner decides to deploy the requested page. To do so, it uses the architecture description to get the dependencies and to find the set of components to deploy (presentation, logic and data), and the bindings to redefine. Finally, the user is redirected on the edge server.

The rules have been implemented using the Drools engine (Drools.org). Mandarax and Jrules have also been evaluated. In our context, due to the very large amount of information to be processed, the Mandarax performances were not satisfying. Drools and JRules have similar performances, even if Drools may be a bit more powerful when inserting a high number of objects.

### 4.4 Executor

The purpose of the Executor is to execute (on the J2EE servers) the set of actions that has been decided and scheduled by the planner. There are two types of actions: edge server configuration and application configuration (including the management of the EJBs life cycle).

All the actions that can be executed regarding the applications are defined in the JSR 88 that provides a way to manage application deployment. It means that a J2EE server complying with this specification provides remote control over the deployed component lifecycle. It is then possible to remotely ask for application installation, update or uninstallation. These requests are sent to the servers using the JMX standard.

Regarding servers configuration, the following actions are particularly important:

– the configuration of the JDBC connectors which link data components and databases. When a data component is moved to an edge server, (part of) a database may be replicated. In this case, a local access must be configured. If the database is not moved, a remote access has to be specified.
– the configuration of the registry which indicates where the components of an application are running. Updating the registries is necessary when applications are split and run on different machines. Components deployed on edge servers must have access to a well configured registry in order to bind the remote component still running on the primary servers.

These actions are also sent to the server in accordance with JMX, using the JSR 77 for J2EE management.

## 5 Prototype and Simulations

The autonomic system presented in this paper has been validated in different ways. First, we have developed an application on a limited number of machines in order to validate the business interest of our approach. Second, we have tested the system on a simulated environment in order to evaluate its ability to scale up.

### 5.1 Video on Demand Application

We have tested our solution on a use case dealing with video on demand on a Content Delivery Network. To do so, we have used the OSGi (www.osgi.org) based implementation of JOnAS [3] for the edge servers and the autonomic architecture presented in this paper.

OSGi is a software services deployment platform. Thanks to the introduction of OSGi within JOnAS, it is possible to deploy on demand and at runtime non-functional services needed by the installed applications. We are then able to delegate to the application servers the management of non functional application dependencies that normally fails to the autonomic manager.

The experiments were successful. Our system allows the dynamic deployment of part of the applications on edge servers. For instance, we are able to dynamically cache media files on edge servers in order to improve the response time. If no edge server is present at a given time, we deploy a streaming server (we use Apple' Darwin stream server) still taking advantage of OSGi capabilities.

It was not possible however to evaluate every aspect of our architecture. In particular, a thorough evaluation of the performances would have required a number of machines exceeding the resources we had. We then decided to use simulated environments to fully test our architecture.

## 5.2 Simulation

This section presents a simulation of the Edge Computing environment and shows the effect of the administration policies. To ensure this approach is realistic, we validate it by simulating the behavior of a classical environment. It shows that response time strongly increases with the number of clients. We then simulate an Edge environment, evaluating different management policies.

### 5.2.1 Simulated Environment

We based our simulations on the J-Sim simulator (www.j-sim.org), extended in order to support a sufficient number of elements. The network is made of different entities:

- Primary servers representing the service provider infrastructure (mainframe, cluster …)
- Edge servers representing ISP infrastructure (routing and hosting)
- Clients

The simulated network is made of 5000 clients, one primary server and 20 ISP networks (each with 1 Edge Server). To model low cost machines for Edge Servers, we have chosen a ratio of 15 between the power of an Edge server and the power of a primary server.

Clients are driven by two different scenarios. Every client launches 15 queries and has a thinking time of 30s. In the first scenario, if the client is redirected, all others queries are treated by the Edge server. There are no dependencies between the Edge server and the primary server. In this case, the whole application is replicated on the Edge servers. The second scenario imposes that 20% of the clients' connections must use the primary server. This is a simplified version of TPC-W, a well known transactional web e-Commerce benchmark (www.tpc.org). In this case, only the presentation part and the logic part are replicated.

To simplify the simulation, the primary server hosts a unique application composed of three parts: a presentation part, a logic part and data. In each simulation, we measure the client response time, the number of redirected clients, and the "Edge periods" (period during which Edge servers are used to host part of the application).

### 5.2.2 Simulations

The scenario we are using simulates a connection peak on a primary J2EE server. It is divided into four steps characterized by the speed of client queries. In the first step, client connections arrive at a slow rate (P1) with respect to the primary server power. At this stage, the server can easily treat to all of them. During the second step, client connections are very frequent (P2). The primary server cannot face this rush; the response time grows. Then, the rush slows down, but the primary server still has difficulties to answer quickly to all clients (P3). Finally, the client connections reach again a low rate (P4).

The first simulation does not use the Edge servers. Figure 4 shows the response time in this case. It clearly appears that the primary server alone is not able to guaranty a reasonable response time.

The second simulation uses Edge servers with a threshold-oriented policy: when the primary server reaches a given number of hosted sessions, it launches the deployment process on Edge Servers and redirects new clients. The threshold was chosen to redirect clients only if the response time is greated than 20s. With a lower threshold, more clients would be redirected and more edge servers may be needed. On the other hand, the mean response time should be smaller.

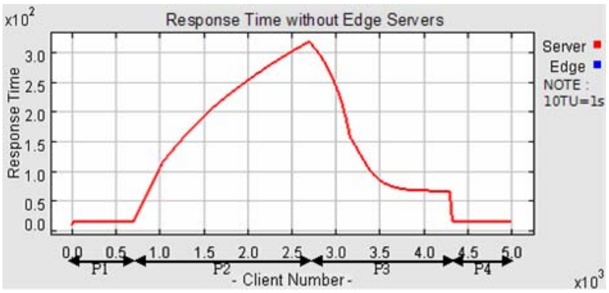


**Fig. 4.** Response time per client without Edge server

Figure 5 shows the effect of Edge Server on the client response time. It clearly appears that the first redirected clients wait a substantial amount of time due to the deployment of the application on Edge Servers. When the application deployed on the Edge Server is ready, redirected clients have a low response time. Indeed, Edge Server does not host a lot of sessions, so clients are treated quickly despite the "low" power of the machine. In our simulation 384 clients are redirected. The mean response time in this case is 9s against 15s without Edge Servers. In addition, 10% of the network bandwidth is saved. However edge computing has a financial cost: in this example, edge servers have to be rented during 11 minutes (see "edge period").

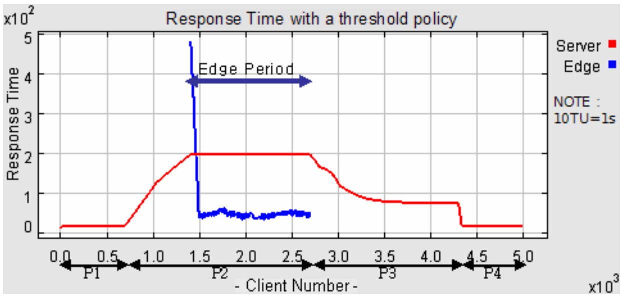


**Fig. 5.** Response Time per client with a threshold policy

In the third simulation, we use Edge servers with a proactive policy. The purpose of this policy is to detect a peak of connections and to launch the migration before the redirection threshold is met. However, when launching the migration in advance, the redirection threshold is not yet reached. This approach can be inadequate if the workload changes rapidly. As illustrated in Figure 6, this policy increases the "Edge Period". In our example, edge servers have to be rent during 13 minutes but redirected clients do not wait during the migration.

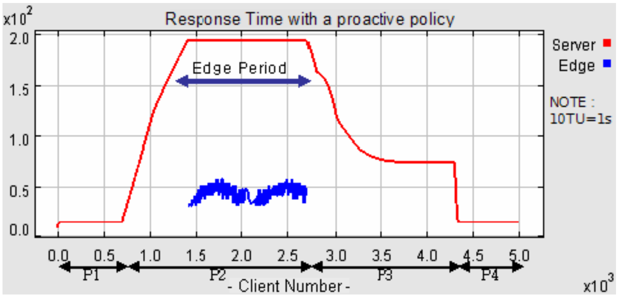


**Fig. 6.** Response time per client with a proactive policy

### 5.2.3 Summary

Simulation has shown the ability of our system to scale up when the number of clients increases. It has also shown that the reaction time of the system, that is the time needed to decide on a migration, remains low.

We have also investigated different management policies. It is interesting to note that many policies can be applied on an Edge Computing environment. For example, it can be decided to use edge servers as soon as the primary server is started. So clients never pay for migration. However, in this case, service providers must rent resources during all the application lifetime. The choice of a policy depends on the available resources and the desired response time but also on the components to migrate / replicate. For example, if Edge servers are really slow and shared by several service providers, the threshold should be higher to avoid overloading of Edge servers.

## 6 Related Work

We have partly tackled two different domains to perform this work. The first one is Edge Computing. The second one is Autonomic Computing.

Edge computing is today more than a concept: many architectures are already deployed and substantial benefits are provided. In the Content Delivery Network (CDN) domain, edge servers are used to stream sounds or video with a good quality and without latency and to adapt the media to the client viewer [15]. In the Web cache domain, edge servers allow the caching of static and dynamic web pages. However, these applications focus only on the migration of the presentation layer of applications. Some academic work has studied the migration of the data layer of applications [6][16]. To our knowledge, there are currently no significant results about the migration of complete applications or of the logical (business) layer of applications.

Autonomic is a recent but very popular domain. Numerous studies have been started on both the industrial side and the academic side [7][8][10][11]. We can mention here interesting work on the use of probes to collect information on the environment, on the use of ECA rules [12] or constraints [9] to analyze the environment and to determine actions to be undertaken.

Autonomic managers are used today in order to supervise environments such as clusters (some work has been done in the J2EE context [14]). In this domain, the resources to be managed are owned and managed by a single actor. In the edge computing domain, resources are owned by different actors and may be multi-operated. Data to be collected in order to take a decision are then different and, we believe, more numerous. Also, the large-scale distribution of the servers raises additionnal problems. At present time, there is not a lot of on going work about fine grained distribution of applications.

## 7 Conclusion

We have presented in this paper an innovative approach to dynamically distribute applications on edge servers. We promote a fine-grained approach where presentation, business and data components can be deployed on edge servers depending on the situation.

We have also developed the idea that advanced edge computing environements are difficult to manage and that autonomic computing can help. We have then presented an autonomic manager, which architecture is derived from the seminal work of IBM. The autonomic manager uses an ADL-based description of the applications in order to dynamically distribute them on the best places. Our approach has been validated on a real-size application and tested more thoroughly in a simulated environment. The autonomic manager we propose is generic and can be adapted to any domains whereas application descriptions used by the manager are specific to the J2EE domain.

Short terms perspectives concerning this work are to improve the data polling on the environment. Adding new data types may help to refine the rules and decrease the response time. We also intend to extend the use of the autonomic manager to other domains than J2EE.

## References

1. Kephart, J., Chess, D. “The vision of autonomic computing”, *IEEE computer,* 36(1), 2003
2. Desertot, M., Escoffier, C., Donsez, D. “Autonomic Management of J2EE Edge Servers”, *3rd International Workshop on Middleware for Grid Computing*, MGC’05, Grenoble, November 2005
3. Desertot, M., Donsez, D. “Infusion of OSGi Technology into a J2EE Application Server”, *OSGi World Congress*, Presentation, Paris, October 2005
4. Weihl, A., Jay, P., William, E. “Edge computing: Extending Enterprise Applications to the Edge of the Internet” *In Proceedings of the 13th International World Wide Web Conference,* May 2004
5. Levy, S., Gummadi, K., Dunn, R., Gribble, S., An, H. “Analysis of Internet Content Delivery Systems”. *In Proceedings of the 5th Symposium on Operating System Design and Implementation,* December 2002
6. Gao, L., Dahlin, M. ”Application specific data replication for edge services”. *In Proceedings of the International World Wide Web Conference,* May 2003
7. Kephart, J.O., Walsh, W.E., Watson, T.J. “An Artificial Intelligence Perspective on Autonomic Computing Policies”, *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks* POLICY'04, 2004
8. Agarwal, M., Bhat, V., Liu, H., Matossi, V. “Automate: Enabling Autonomic Applications On The Grid”, *http://automate.rutgers.edu/*
9. Dearle, A., Kirby, G., McCarthy, A. “A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications”, *In Proceedings of the International Conference on Autonomic Computing,* May 2004
10. Breitgand, D., Henis,E., Shehory, O. “Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management” *In Proceedings of the International Conference on Autonomic Computing,* June 2005
11. Kumar, V., Cooper B.F., Schwan K. “Distributed Stream Management using Utility-Driven Self-Adaptive Middleware” *In Proceedings of the International Conference on Autonomic Computing,* June 2005
12. Srivastava, B., Bigus, J.P., Schlosnagle, D.A. “Bringing Planning to Autonomic Applications with ABLE”, *In Proceedings of the International Conference on Autonomic Computing,* May 2004
13. IBM Redbook, “Taking Websphere Commerce to the Edge”, *online book*, http://www.redbooks.ibm.com/abstracts/sg246456.html
14. Akkerman, A., Totok, A., Karamcheti, V. “Infrastructure for Automatic Dynamic Deployment of J2EE Applications in Distributed Environments”, *Third International Working Conference on Component Deployment, CD’05*, Grenoble, November 2005
15. Bertini, M., Cucchiara, R., Del Bimbo, A., Prati, A. “Content-based video adaptation with user's preferences”, *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo*, ICME’04, Taipei, June 2004
16. Gao, L., Dahlin, M., Zheng, J., Alvisi, L., Iyengar, A. "Dual-Quorum Replication for Edge Services”, *ACM/IFIP/USENIX, 6th International Middleware Conference*, Middleware 2005, Grenoble, November December 2005

# Short Papers

# Ubiquitous Zone Networking Technologies for Multi-hop Based Wireless Communications[*]

Namhi Kang, Ilkyun Park, and Younghan Kim[**]

Ubiquitous Network Research Center, Soongsil University, Seoul Korea
{nalnal,ikpark,yhkim}@dcn.ssu.ac.kr

**Abstract.** This positioning paper presents u-Zone (ubiquitous-Zone) based multi-hop wireless network architecture and its components. Several networking technologies to support reliability, enhanced scalability, heterogeneity, internet connectivity, and mobility are briefly presented.

## 1 Introduction

The proposed u-Zone based network architecture is intended to support ubiquitous computing paradigms without restrictions of time and place. As a promising approach to ubiquitous networking, we focus on a hybrid MANET (Mobile Adhoc Network). Both u-Zone master (u-ZM) and uT-Gateway (uT-GW) are the key components of the architecture (see Fig. 1). They assist mobile nodes to communicate with each other in a scalable and efficient fashion.

**Table 1.** Requirements of ubiquitous community networks

| Requirement | Proposed solutions |
|---|---|
| Robustness and efficiency | Wireless backbone (u-ZM mesh) |
| Scalability | Hierarchical network architecture based on u-ZM & $H_2O$ routing strategies |
| Support of heterogeneity | HRPC, DYMO(for heterogeneous OSs) |
| Internet connectivity | uT-GW, IP Auto-configuration |
| Mobility | IP Auto-configuration with $H_2O$ |

In MANET, there exist difficulties in finding a stable route. Such a problem becomes harder as the size of network is larger. This is mainly due to the fact that MANET is self-configured in the absence of centralized network infrastructures. In addition, to build MANET in the real world, we are required to consider several challenges that include the mobility of nodes resulting in frequent topology changing, scarce network resources, low computing power and limited energy of nodes, and difficult handling in scalability and reliability. Such considerations and our solutions are summarized in Table 1.

[*] This research is supported by the ubiquitous Autonomic Computing and Network Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea.

[**] Correspondent author.

## 2 u-Zone Based Community Networks

Fig. 1 illustrates the proposed u-Zone based network architecture including the first prototype of both u-ZM and uT-GW. A large scale of MANET is divided into a set of sub-regions. A sub-region is referred to as a u-Zone, where a u-ZM is placed. Community network may consist of one or more u-Zones. The u-ZM is the central component to form a hierarchical architecture. Unlike ordinary nodes, the u-ZM has high computing power and robust electrical power as a super node (neither a source nor a sink). Hence, the architecture is regarded as a semi-infrastructured MANET. In some scenarios, a sink node of sensor network may be a member node of a u-Zone. Also a mobile node is able to access Internet or different kinds of networks through the uT-GW.

**Robustness and efficiency:** In the architecture, u-ZMs are interconnected to form a infra/backbone mesh topology (i.e. wireless backbone (WBB)). If the WBB is not used, traffic may concentrate on a few nodes resulting in high end-to-end delay and loss probability due to congestion at the nodes and high energy consumption of the nodes. In addition, such an approach offers a way to achieve spatial reuse thus to enhance the performance of overall network [1].

**Scalable Routing:** Our routing strategy called H2O (Hierarchically Optimized Hybrid) borrows the basic concept of cluster based routing protocols [1]. That is, H2O also consists of two levels: intra u-Zone routing and inter u-Zone routing in a hybrid way. Unlike ZRP, however, H2O gives more opportunities to select appropriate combination of routing methodologies according to the requirements of applications or network environments (see Fig. 2).

**Heterogeneity:** To demonstrate H2O, we have implemented DYMO (Dynamic MANET On-demand Routing Protocol) [3] as a reactive routing protocol and ported OLSR (Optimized Link State Routing Protocol) [4] as a proactive

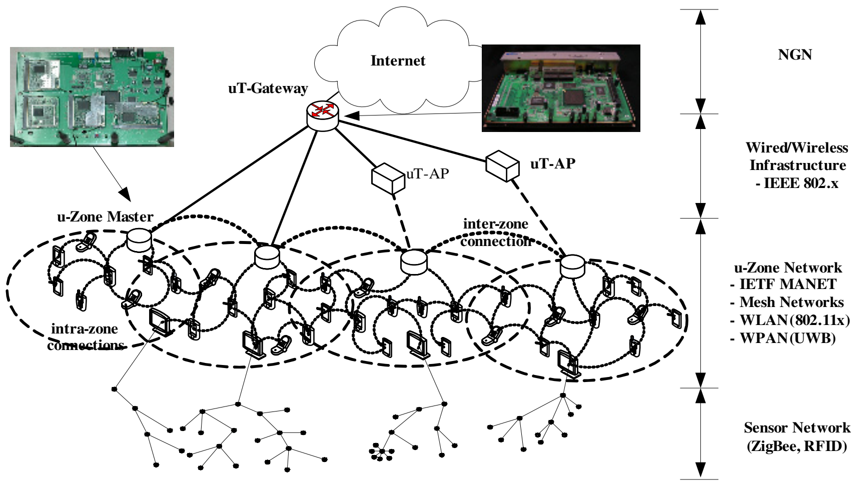
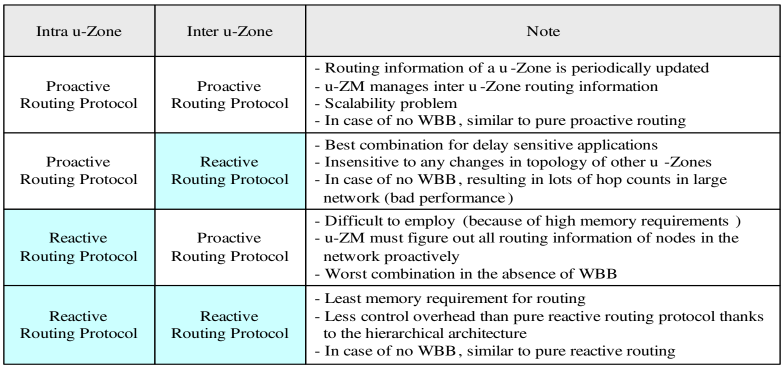


**Fig. 1.** u-Zone based network architecture with standard trends

routing protocol. To support heterogeneity of mobile nodes, the u-ZM contains the heterogeneous routing protocol coordinator (HRPC) which performs bridging functionality between DYMO and OLSR. Further, the DYMO have been implemented for various systems (Linux, Windows XP, Windows CE).

**IP address auto-configuration:** Recently, several schemes have been proposed for address allocation in multi-hop based wireless networks, but most of schemes do not consider the global internet connectivity. For the allocation of global-scope addresses to the mobile nodes, we have proposed a scheme called HAA (Hybrid Address Auto-configuration) [5]. The HAA was implemented in the form of daemon process on the Linux system.



| Intra u-Zone | Inter u-Zone | Note |
|---|---|---|
| Proactive Routing Protocol | Proactive Routing Protocol | - Routing information of a u -Zone is periodically updated<br>- u-ZM manages inter u -Zone routing information<br>- Scalability problem<br>- In case of no WBB, similar to pure proactive routing |
| Proactive Routing Protocol | Reactive Routing Protocol | - Best combination for delay sensitive applications<br>- Insensitive to any changes in topology of other u -Zones<br>- In case of no WBB, resulting in lots of hop counts in large network (bad performance) |
| Reactive Routing Protocol | Proactive Routing Protocol | - Difficult to employ (because of high memory requirements )<br>- u-ZM must figure out all routing information of nodes in the network proactively<br>- Worst combination in the absence of WBB |
| Reactive Routing Protocol | Reactive Routing Protocol | - Least memory requirement for routing<br>- Less control overhead than pure reactive routing protocol thanks to the hierarchical architecture<br>- In case of no WBB, similar to pure reactive routing |

**Fig. 2.** Four different routing combinations

## 3 Conclusion and Further Work

In this positioning paper, we have briefly introduced the u-Zone based network architecture and technologies necessary to meet several requirements of the future oriented ubiquitous network. Further work aimed at integrating various networking technologies into the u-Zone Master, thereafter building a real test-bed to evaluate a campus scale of hybrid MANET.

## References

1. N. Kang, I. Park, and Y. Kim. Secure and Scalable Routing Protocol for Mobile Ad-hoc Networks, Lecture Notes in Computer Science, Vol 3744, Oct. 2005.
2. Z. J. Haas and M. R. Pearlman. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. IETF Internet Draft, 1998.
3. I. D. Chakeres, E. M. Royer and C. E. Perkins, Dynamic MANET On-demand Routing Protocol, IETF Internet draft, draft-ietf-manet-dymo-03.txt, Oct. 2005.
4. T. Clausen and P. Jacquet, Optimized Link State Routing Protocol (OLSR), IETF RFC, RFC 3626, Oct. 2003.
5. I. Park, Y. Kim and S. Lee. IPv6 Address Allocation in Hybrid Mobile Ad-Hoc Networks. IEEE WSTFEUS 2004, May 2004

# Proposal for Self-organizing Information Distribution in Peer-to-Peer Networks

Arne Handt

Freie Universität Berlin, Institut für Informatik,
Working Group Networked Information Systems,
Takustr. 9, D-14195 Berlin, Germany
handt@inf.fu-berlin.de
www.ag-nbi.de

**Abstract.** The usage of peer-to-peer networks often follows a query-response paradigm, where users initiate searches which then have to be routed over the network efficiently. This paper proposes an approach for self-organizing information distribution in peer-to-peer networks that inverts this paradigm: data objects actively travel through the net to those nodes for which they are relevant. The underlying mechanism is rooted in the principles of Swarm Intelligence and relies on the dissemination of artificial pheromones, where each pheromone represents one particular relevance criterion that applies to a given data object. Data objects leave trails of these pheromones at each node they visit and move along gradients of pheromone concentration to regions in which they are relevant.

## 1 Introduction

Peer-to-peer systems provide a flexible infrastructure for sharing resources. The discovery of resources in such a network commonly follows a query-response paradigm, where users initiate searches which have then to be routed over the network efficiently. This paper proposes an approach for self-organizing information distribution in peer-to-peer networks that inverts this paradigm: data objects actively travel through the net to those nodes for which they are relevant. One possible application scenario for such a system would be a novel infrastructure for the distribution and the discovery of RSS feeds, where feed items are not only delivered upon request, but in which each item travels as a data object through a peer-to-peer net to those nodes that might be interested in this item. A premise for this approach is that each node maintains a set of criteria which can be used to determine the relevance of a given data object.

## 2 Algorithm

Our approach to achieving this is rooted in the principles of Swarm Intelligence [1,2], which mimics the behavior found in animal swarms. Its core insight is that swarms of individuals, which act locally and based on a small set of simple rules, can solve problems which would be impossible to solve by the individual alone. This phenomenon of *emergent behavior* has been successfully exploited in numerous applications and techniques, e.g. [3], [4]. Vital for our approach is the use of artificial pheromones that represent relevance criteria and are disseminated by data objects as these travel through the net. The resulting pheromone concentrations are exploited to optimize the network structure as well as to control the propagation of data objects. Furthermore, a charging mechanism is used to prevent data objects from trailing off into areas of the network where they are not relevant: each hop reduces a data object's charge, encountering a matching pheromone recharges it, and a new hop requires the charge to be above a particular threshold (see Table 1). Before travelling to a new node, the

**Table 1.** Outline of Data Object Behavior

```
at each node:
    reduce charge
    score data object according to local pheromones
    recharge
    deposit pheromones
    pick up local pheromones
    if charge is above threshold: copy to each of the current node's neighbors
```

data object deposits all its pheromones and picks up new matching pheromones, thus spreading them through the net. Figure 1 illustrates the behavior of a data object representing an article on black holes which visits two nodes that have pheromones representing the topics "Cosmology" and "Quantum Theory" respectively. Since both topics apply to the imaginary article, it collects both pheromones. One of the effects that are intended by the design of this behav-

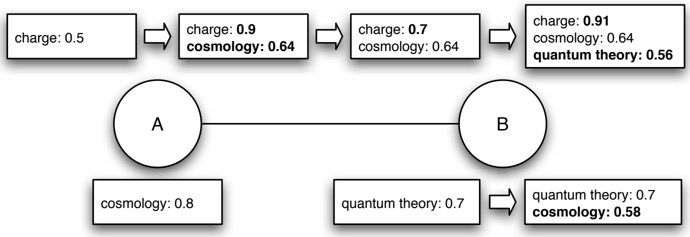

charge: 0.5
charge: **0.9** **cosmology: 0.64**
charge: **0.7** cosmology: 0.64
charge: **0.91** cosmology: 0.64 **quantum theory: 0.56**
A
B
cosmology: 0.8
quantum theory: 0.7
quantum theory: 0.7 **cosmology: 0.58**

**Fig. 1.** Example

ior is the emergence of gradients of pheromone concentration in the network. Since the pheromone concentration influences data objects by recharging them, this leads to data objects moving along concentration gradients to nodes of high pheromone concentration, i.e. nodes to which they are highly relevant, and they are prevented from flooding areas of the net for which they are irrelevant.

## 3 Related Work

Our approach bears resemblances to several systems and methods in the context of Swarm Intelligence, some of which are shortly outlined in this section.

*SemAnt* [5] is an algorithm for routing queries in peer-to-peer documents with an adaptation of the Ant Colony Optimization [4] metaheuristic. SemAnt presupposes the use of a global taxonomy for the network and uses concepts in the taxonomy as pheromones for query routing.

*Swarmix* [6] is an algorithm for peer-to-peer-based collaborative filtering. A Swarmix peer publishes its interest profile and uses the information of other profiles to infer the relevance of a given data object for itself.

## 4 Conclusion and Future Work

In this paper, we proposed a mechanism for self-organized information distribution in peer-to-peer networks. Its basic idea is that data objects travel through the net to find nodes for which they are relevant. This is carried out by disseminating artificial pheromones through the net and exploit the pheromone concentration to optimize the data object propagation as well as the network structure. A simulation of this approach with real-world data is currently under development. Its purpose is to evaluate performance, reconsider design decisions and find concrete values for the algorithm's parameters. One vital question that still has to be answered concerns the definition of relevance criteria. It must allow for a fast scoring of data objects and derivation of pheromones that can be carried by data objects through the net.

## References

1. James Kennedy and Russell C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.
2. Mitchel Resnick. *Turtles, Termites, and Traffic Jams.* The MIT Press, 2001.
3. Ahmed Charles, Ronaldo Menezes, and Robert Tolksdorf. On the Implementation of Swarm Linda. In *ACM-SE 42: Proceedings of the 42nd Annual Southeast Regional Conference*, pages 297–298, New York, NY, USA, 2004. ACM Press.
4. Marco Dorigo and Gianni Di Caro. The Ant Colony Optimization Meta-Heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.
5. Elke Michlmayr, Arno Pany, and Gerti Kappel. Using Taxonomies for Content-based Routing with Ants. In *Proceedings of the 2nd Workshop on Innovations in Web Infrastructure*, 2006.
6. E. Diaz-Aviles, L. Schmidt-Thieme, and C.-N. Ziegler. Emergence of Spontaneous Order Through Neighborhood Formation in Peer-to-Peer Recommender Systems. In *Proceedings of the 1st Workshop on Innovations in Web Infrastructure*, 2005.

# Autonomic Security for Home Networks

Mohamad Aljnidi and Jean Leneutre

CNRS - UMR 5141 LTCI - ENST - INFRES department
46, rue Barrault - 75013 Paris - France
{Mohamad.Aljnidi, Jean.Leneutre}@enst.fr

**Abstract.** Home networks are becoming prevalent and interest in their security is increasing. We introduce in this paper an autonomic security model, in which we deal with a home network as an ad hoc network in general, but also we consider its particularities. We show how autonomy is required in different aspects of the proposed solution. Above all, we address autonomy to minimize the intervention of home users, who generally lack experience, in the management of the security infrastructure.

## 1 Introduction

Such as all communication networks, a home network is prone to security attacks. However, a home network needs special security solutions, taking into consideration its ad hoc nature, in addition to other particularities, including but not limited to, longterm inter-device relations, quasi-static topology, diversity of networking technologies, heterogeneity of devices and inexpert administrators.

We study security of home networks in the context of a research about autonomic security for mobile networks. Our main assumption is that home devices can depend on the existence of each other under time constraints; Because a home is an ultimate meeting point for its mobile devices, we can define T as the longest time for which a device can be away from home. The value of T goes from some hours to a couple of days according to mobility needs or preferences.

Security solutions for ad hoc networks address decentralization and self-organization [1] [2]. This is also required for the special case of home networks [3], but we can impose limitations thanks to home particularities, using the constant T for instance. A Previous research [4], which we generalize and enhance in a part of our work, defines a home network as a longterm community, and eventually proposes a certificate-based decentralized model. Another one [5], which inspired us in terms of security architecture, proposes a generic design for a self-organized security system that can be applied in the case of home networks.

## 2 Requirements

Devices of a home network are not homogeneous nodes. If we use asymmetric cryptography we will have performance problems with light-duty devices. If we use symmetric cryptography we will loose the chance for a better security implementation in heavy-duty devices.

**Requirement $\Re 1$:** A security process should be able to adapt itself to the cryptographic capabilities of the involved devices.

Devices of a home network may belong to a single resident. In this case, we can consider that there is a trust relationship between devices. We may have the same situation with devices that belong to many residents who trust each other. In general, a home network would be shared by many family members of different ages, and maybe temporary users such as guests. In this case for example, trust is not necessarily complete between devices.

**Requirement $\Re 2$:** A security process should refer, when needed, to authorization rules defined according to levels of inter-device trust.

A home network is subject to variations in device population [4]. According to $\Re 2$, and in terms of generalization of [4], we also deal with variations in trust levels. Moreover, our model implies variations in the security management infrastructure. In all cases, a variation might compromise the network.

**Requirement $\Re 3$:** Variations in device population, inter-device trust levels or security management infrastructure should happen securely.

A home device is usually expected to make part of a home network over a long period. In other words, longterm inter-device relations are to be established. It's more efficient to secure the whole relation between two devices instead of securing each communication separately. Besides, we should avoid compromising the network during a relation establishment. On the other hand, according to $\Re 1$ and $\Re 2$, we need to categorize secure relations according to the differences between the involved devices in terms of capabilities and trust levels.

**Requirement $\Re 4$:** Inter-device secure relations should be securely established and categorized according to device capabilities and trust levels.

We designate certain heavy-duty devices as authority nodes. The main role of such devices is to manage variations. Besides, they may assume a security server role during the establishment of secure relations, especially when light-duty devices are involved. Actually, the constant T assumption is made to prove that authority nodes can always be considered available for relation establishments. This eventually implies a limitation of decentralization, which we can sometimes avoid if the relation establishment is between heavy-duty devices. Anyway, this limitation can be acceptable since relation establishments are occasional in a home network. However, this authority-related type of centralization becomes important if it persists during data exchange in a secure relation.

**Requirement $\Re 5$:** Devices bound by a secure relation of any category should be able to communicate securely without any contact with a third party.

Each device in the home network stores security information for its relations. An authority node stores additional security information to be used as management data during variations and relation establishments. Security management data are updated on an authority node after the variations that involve it. An authority node may not be involved in a variation, but it is expected afterward to have updated its security management data accordingly.

**Requirement $\Re 6$:** Authority nodes should be able to synchronize their security management data after network variations.

## 3 Device Categories

We categorize home devices according to their computation and storage capabilities ($\Re 1$). We suppose that a security platform is to be installed on each home device before adding it to the network. At installation time, a device evaluation module automatically determines the device category according to a security configuration policy (self-configuration [6]). The installed modules work either as applications, or as the constituents of an autonomic security layer supporting the application layer. In both cases, the installed platform is irrespective of the underlying networking technologies.

We consider two device categories: LD (Light-duty Device): the device can support symmetric cryptography and store a limited set of symmetric keys, and HD (Heavy-duty Device): the device can be an LD, and besides, it can support asymmetric cryptography and store its asymmetric key data and a set of certificates and access control policies.

We suppose that the security system can be asked to exclude a certain communication port on a device. This exclusion is used to insecurely communicate with a device that can't even be an LD, which avoids constraints on existing networks. This is also useful for isolation of external communications. Nevertheless, the data exchanged internally with an excluded port is automatically monitored according to a protection policy(self-protection [6]).

## 4 Security Model

The home network is a set of device communities. A mutual trust relation relies the devices of one community ($\Re 2$). A device is in the security perimeter of the network if it belongs to one of its communities. A single HD in a community is selected to be its authority node, while any other HD of it can be a delegated authority node. The loss or breakdown of a main or a delegated authority node is automatically detected, and the system eventually designates another HD as a replacement (self-healing [6]). We suppose that the home network includes one HD at least. This guarantees that there is always an HD that can be designated as a main or a delegated authority node in many communities, especially in temporary cases of emergency. Delegated authority nodes assume security management temporarily while accompanying devices away from the home network coverage for a period greater than T. This way, neither a variation nor a relation establishment will be blocked for more than a period of T. In other words, we can export a home subnetwork with all the functionalities of the security system.

We define nine secure network variations ($\Re 3$): Two variations are related to trust levels: community integration or revocation. Three others are related to the security management infrastructure: authority replacement, delegation or delegation termination. And finally, four variations are related to the device population: insertion, removal, banishment or reinsertion. Reinsertion is used to cancel a banishment. An autonomic operation of synchronizing security management data among authority nodes (self-optimization [6]) is carried out within and after variations ($\Re 6$).

Secure Authority-Authority Relations (AAR) are automatically created after community integrations, and secure Authority-Device Relations (ADR) are automatically created after device insertions or reinsertions (self-configuration [6]). Creation and distribution of keys and certificates automatically take place when AAR and ADR relations are established.

A secure relation can be created between any two devices of the network ($\Re 4$), if they are in the security perimeter, even if they don't belong to the same community. When such a relation is created between two devices, they can communicate securely using the distributed keys or certificates and independently of any other device in the network ($\Re 5$). Authentication protocols, which may involve authority nodes depending on device categories, are needed ($\Re 4$) for establishing an HHR (HD-HD Relation), an HLR (HD-LD Relation) or an LLR (LD-LD Relation). A relation-dedicated symmetric key is created for an HLR or an LLR, while certificates are used in an HHR.

Authorization policies are exchanged ($\Re 4$) in the context of an AAR, and during the establishment of a secure relation between two devices of different communities ($\Re 2$). We categorize the result as a Low-Trust Relation (LTR), compared to High-Trust Relations (HTR) between the devices of one community. HD devices can store authorization policies, while an LD asks its interlocutor for permission proofs during communications in the context of an LTR.

## 5 Conclusion

We presented the main ideas and guidelines of a security model, which is the basis of our first research work in terms of autonomic security solutions for mobile networks. It opens the door for future research tracks, including but not limited to, intra-device autonomic security elements, inter-community autonomic security information negotiation and synchronization, and specification of self-management high-level policies for mobile networks.

## References

1. L.M.Feeney, B.Ahlgren, A.Westerlund: Spontaneous networking: an application-oriented approach to ad hoc networking. Communications Magazine, IEEE **39** (2001) 176–181
2. L.Zhou, Z.J.Haas: Securing ad hoc networks. IEEE Network (1999)
3. C.M.Ellison: Home network security. Intel Technology Journal **6** (2002)
4. N.Prigent, C.Bidan, J-P.Andreaux, O.Heen: Secure long term communities in ad hoc networks. In: First ACM workshop on Security of Ad Hoc and Sensor Networks, Fairfax, Virginia (2003)
5. T.Messerges, J.Curkier, T.Kevenaar, L.Puhl, R.Struik, E.Callaway: A security design for a general purpose, self-organizing, multi-hop ad hoc wireless network. In: First ACM workshop on Security of Ad Hoc and Sensor Networks, Fairfax, Virginia (2003)
6. J.O.Kephart, D.M.Chess: The vision of autonomic computing. Computer **36** (2003) 41–52

# Hovering Data Clouds: A Decentralized and Self-organizing Information System*

Axel Wegener[1], Elad M. Schiller[2], Horst Hellbrück[1], Sándor P. Fekete[2], and Stefan Fischer[1]

[1] Institut für Telematik, Universität zu Lübeck
{wegener, hellbrueck, fischer}@itm.uni-luebeck.de
http://www.itm.uni-luebeck.de
[2] Institut für mathematische Optimierung, Technische Universität Braunschweig
{s.fekete, e.schiller}@tu-bs.de
http://www.math.tu-bs.de/mo

**Abstract.** With ever-increasing numbers of cars, traffic congestion on the roads is a very serious economic and environmental problem for our modern society. Existing technologies for traffic monitoring and management require stationary infrastructure. These approaches lack flexibility with respect to system deployment and unpredictable events (e.g., accidents). Moreover, the delivery of traffic reports from radio stations is imprecise and often outdated. In the project AutoNomos we aim at developing a decentralized system for traffic monitoring and managing, based on vehicular ad-hoc networks (VANETs). Our objective is to design a system for traffic forecasting that can deliver faster and more appropriate reactions to unpredictable events. In our design, cars collect traffic information, extract the relevant data, and generate traffic reports. A key concept are so-called *Hovering Data Clouds* (HDCs), which are based on the insight that many crucial structures in traffic (e.g., traffic jams) lead an existence that is independent of the individual cars they are composed of. The result is an elegant, robust and self-organizing distributed information system. In this paper we demonstrate first experimental results.

## 1 Introduction

Mobility and individuality are cornerstones of our modern society; this explains why car traffic has become so crucial for many aspects of our life. Unfortunately, this importance leads to ever-increasing numbers of cars; given the limited road capacities, frequent traffic hold-ups are common in urban regions and highways all over the world. Beyond the individual loss of freedom and mobility due to time spent in blocked traffic, these predicaments also have a very serious large-scale impact, due to increasing pollution and economic loss. As a consequence, ever-increasing efforts have been spent on optimization of road usage by means of traffic monitoring and management.

* Supported by DFG Focus Program "Organic Computing" (SPP 1183) project "AutoNomos" (Fe 407/11-1).

Traditional online traffic forecasting systems require stationary infrastructure, such as magnetic loops, roadside relay stations, a massive central processing unit, and finally radio stations that broadcast the generated traffic reports (see [1]). Unfortunately, this centralistic approach suffers from a number of inherent design problems: a huge amount of data has to be gathered and communicated to a central server; the necessary central computations are enormous; and the results have to be distributed and implemented in a timely fashion.

Quite often, such a centralized approach is not just complicated, but also inappropriate. Many phenomena are local by nature, so they only affect a subset of traffic participants. Moreover, fast and flexible response is essential for defusing many local traffic situations. All this makes a distributed approach more flexible, faster and more accurate for dealing with dynamic, local traffic phenomena.

Recent advances in wireless communication technologies such as WLAN and GPS allow short-to-medium-range communication systems among vehicles on an ad-hoc basis. Vehicular ad-hoc networks (VANETs) aim at a number of commercial applications such as the improvement of car safety (see [2], [3]) or entertainment such as online gaming. The same equipment is also suited for decentralized traffic forecasting. In contrast to a fixed infrastructure, this approach provides precise results with cost-efficient equipment [4].

Existing centralized implementations require carefully planning with respect to the location of the sensing infrastructure. If the number of sensors is too small or their position is suboptimal, forecasting precision is compromised. On the other hand, just the situations that are critical (e.g., traffic jams) provide a sufficient number of cars to form a VANET and collect traffic statistics throughout the road, making a distributed approach simple and appealing.

Our approach provides fine granularity that improves the forecasting precision; it can also adapt more quickly to sudden, unpredictable events like accidents and traffic congestions, which defy even advanced forecasting models (see [5]).

This work is part of a project that aims at develop a system that provides the drivers with online reports of the current (and upcoming) traffic condition.

## 2 Design Criteria and System Concepts

Traditional system designs and ad-hoc unicast approaches such as [6] do not meet our requirements of scalability (i.e., the number of cars) and the unpredictable nature of car movements. Moreover, flooding of unrefined traffic information across the network does not scale.

In the project AutoNomos, we take a different, localized and distributed approach, in which cars collect information about traffic events, aggregate the relevant data, and generate dynamic traffic reports in a self-organized manner. A crucial concept for this challenge is what we call a *hovering data cloud* (HDC).

HDCs are motivated by traffic phenomena that exist and prevail independent of the individual vehicles they consist of. They are self-organizing entities that are not restricted to particular hosting nodes, nor to fixed regions. This makes them somewhat similar to the virtual mobile nodes described in [7], but the distributed concept allows for a wider range of structures. Practically speaking, HDCs are associated with phenomena such as traffic jams, they are responsible for capturing the phenomena's events and characteristics, and they arise with the onset of the phenomenon. At any time, an HDC has a distinct origin defined by a center and an expanse, i.e. the propagation range. Both can change over time, accounting for the represented event. In this first paper we focus on the description of a traffic density with motionless HDCs.

HDCs expand when nodes selectively forward *HDC messages* in the direction of interest, e.g. towards traffic that is directed to HDC's origin. The forwarding is bounded by the parameter expanse of the HDC, which depends on the underlying traffic event. To extend the propagation range with affordable bandwidth, HDCs are aggregated by their similarity of underlying events.

The desired behavior can only be achieved by self-organizing systems, as the set of hosting nodes changes over time. Locality is also required. The novelty of our approach is that we not only take the traditional storage-centric point of view that assigns the responsibility for data processing and data forwarding to a particular set of nodes.

## 3 A Distributed Algorithm for Implementing HDCs

A HDC includes a generic dataset that hosts traffic statistics. Cars that travel through an HDC region inform the other cars in the region about their present location and datasets. Periodically, a single car updates the HDC with its recent traffic statistics and initiates the propagation of HDC messages. The initiating car can be the one that is closest to the HDC origin.

We have taken an approach of random broadcasts for message propagation and neighborhood discovery, because of the elegant balance between communication loads and refresh rates (see the MILE project [8] for more details). The *update interval* for HDC messages creation and delivery set the trade-off between network loads and refresh rates. In order to reduce the communication overhead, we limit the lifetime of HDCs in absence of refreshing HDC messages. Their propagation expanses are restricted to a predefined "horizon" standing for area (and time) of interest of cars.

We have implemented an HDC that is fed by a simple periodic event: the measurement of car density in a stationary region. Possible extensions may include traffic information such as drive-through speed and lane change behavior. These parameters can reveal interesting irregularities, say, due to blocked lanes or road hazards that drivers are trying to avoid. The potential of the HDC concept to detect unpredictable events is far greater than that of existing implementations.

## 4 Evaluation

We present simulation results for stationary events that monitor car density on predefined locations. These events are set every 1 km (where the events' radius is set to 500 m) on a road with two lanes in one direction. The traffic simulator SUMO [9] generates appropriate movement traces of cars (see [10]). We extended the network simulator ns2 [11] with our HDCs algorithm and fed ns2 with the mobility traces. The communication model is based on the standard IEEE 802.11 with a communication range set to 250 m. We produce variations of the car density and compare it to the propagated and received HDC messages. We are mainly interested in the accuracy of the measurements (car density and time), propagation delay and network overhead produced by the HDCs.

The flow of cars starts with 1500 cars per hour, and increases sharply to 4000 during the simulation. The average speed is stable at around 26 m/s. Figure 4.a depicts the real number of cars passing the 6.5 km mark of the road.

Figure 4.b and Fig. 4.c show the projected traffic density at 6.5 km as received by cars 5.5 km behind (at position 1 km). Measurements are illustrated as points in the curves. In the first simulation run (Fig. 4.b) the update interval for HDC messages is set to 2.5 s, which consumes about 19 kbit/s of bandwidth per km of the road. In the second run the update interval is increased to 5 s (Fig. 4.c), which reduces bandwidth consumption to approximately 9 kbit/s. Note that the trade-off between the HDC message update interval and the bandwidth consumption affects the granularity of traffic reports, because every point on the curve in Fig. 4 represents a received HDC message. Moreover, the curves show that the HDC message update interval influences the propagation delay. Delay decreases from 66 s in Fig. 4.c to 22 s in Fig. 4.b. With advanced forwarding techniques we aim at further decreasing the delay without increasing bandwidth.

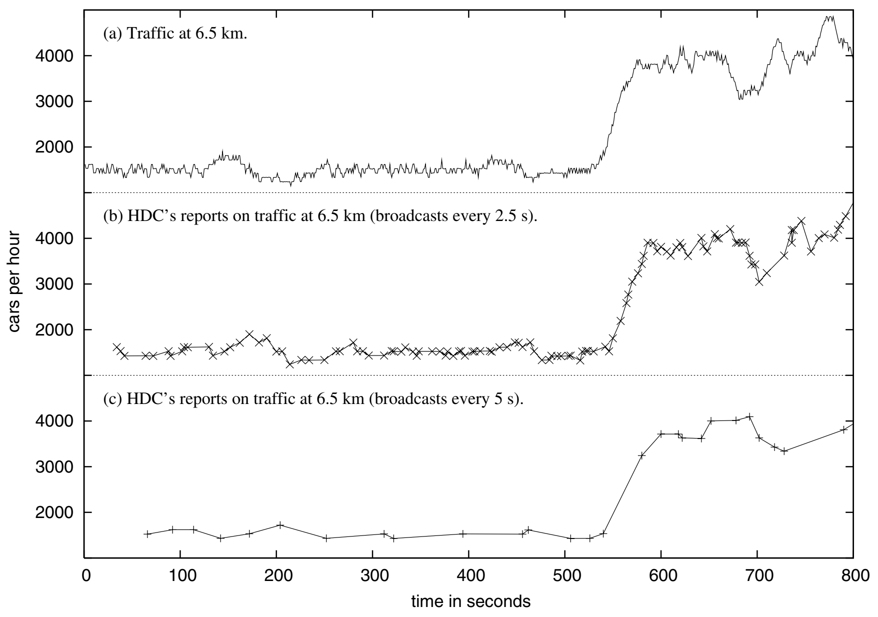


**Fig. 1.** Car density over time. Comparison of real car density and that reported by HDC messages.

## 5 Conclusion and Outlook

The AutoNomos project proposes a new approach for reporting traffic conditions. In contrast to centralized implementations, we facilitate a self-organizing car-to-car communication system. In this work, we introduce the key design concepts for sampling and reporting traffic information. The concept of HDCs allows information flow that is independent of the underlying car flow. Our preliminary tests demonstrate the feasibility of the proposed approach and the HDC concept.

A key topic in this project is to illuminate chain reaction scenarios that may occur when reporting on traffic. For example, the traffic congestion in a network of roads may oscillate, or the Braess paradox may appear (see [12]). Traffic forecasting services that includes additional traffic management aspects, e.g. route planning and road hazard notification, can increase drivers' confidence. We plan to investigate pattern recognition techniques for marking and mapping road hazards. Moreover, we will experiment with large-scale and complex traffic settings, and develop novel design concepts.

## References

1. Mazur, F., Chrobok, R., Hafstein, S., Pottmeier, A., Schreckenberg, M.: Future of traffic information - online-simulation of a large scale freeway network. IADIS International Conference WWW/Internet 2004 **1** (2004) 665–672
2. Franz, W., Hartenstein, H., Mauve, M.: Inter-Vehicle-Communications Based on Ad Hoc Networking Principles – The FleetNet Project. Universitätsverlag Karlsruhe (2005)
3. Car2car communication consortium. Web (2005) `http://www.car-to-car.org/`.
4. Varshney, U.: Vehicular mobile commerce. Computer **37**(12) (2004) 116–118
5. Chrobok, R., Pottmeier, A., Marinosson, S., Schreckenberg, M.: On-line simulation and traffic forecast: Applications and results. In: Proc. of the Internet and Multimedia Systems and Applications. (2002) 113–118
6. Nadeem, T., Dashtinezhad, S., Liao, C., Iftode, L.: Trafficview: A scalable traffic monitoring system. In: IEEE International Conference on Mobile Data Management. (2004) 13–26
7. Dolev, S., Gilbert, S., Schiller, E., Shvartsman, A., Welch, J.: Autonomous virtual mobile nodes. In: 3rd Workshop on Foundations of Mobile Computing (DIAL-M-POMC). (2005)
8. Hellbrück, H., Fischer, S.: Mine and mile: Improving connectivity in mobile ad-hoc networks. Mobile Computing and Communications Review MCCR **8**(4) (2004) 19–36
9. Krajzewicz, D., Hertkorn, G., Rössel, C., Wagner, P.: Sumo: An open-source traffic simulation. In: Proc. of the 4th Middle East Symposium on Simulation and Modeling. (2002) 183–187
10. Krauß, S., Wagner, P., Gawron, C.: Metastable states in a microscopic model of traffic flow. Physical Review E **55**(304) (1997) 5597–5602
11. University of Southern California: Ns-2: Network simulator-2. Web (2005) `http://www.isi.edu/nsnam/ns/`.
12. Murchland, J., Braess, D.: Braess's paradox of traffic flow. Transpn. Res. **4** (1970) 391–394

# Defending Grids Against Intrusions

Alexandre Schulter, Kleber Vieira, Carlos Becker Westphall, and Carla Westphall

Networks and Management Laboratory, Federal University of Santa Catarina, Florianopolis, Brazil
{schulter, kleber}@inf.ufsc.br, {westphal, carla}@lrg.ufsc.br

**Abstract.** Current intrusion detection technology is limited in providing protection against the intrusions that may violate the security of grids. We present the mechanisms necessary to integrate a grid-based intrusion detection system with other systems so as to provide protection against all the intrusions a grid may be subjected to. A case study that makes use of simulations and a proof-of-concept implementation is presented as an early evaluation.

**Keywords:** Computational grids; security; intrusion detection.

## 1 Introduction

Computational grids are emerging as tools to facilitate the secure sharing of resources in heterogeneous environments. Security is one of the most challenging aspects of grid computing and Intrusion Detection Systems (IDS) [1] have an important role in grid security management. IDSs are responsible for the detection of intrusions in information systems and the responses to them, usually as alert notifications.

As described in our previous work [2], the need for intrusion detection in grids and the shortcomings of the available solutions motivated our Grid-based Intrusion Detection System (GIDS) approach. To cover all the classes of intrusions a grid may be subjected to, our recommended GIDS acts as a high-level component that utilizes functionality of lower-level Host-based IDS (HIDS) and Network-based IDS (NIDS) provided through inter-IDS communication.

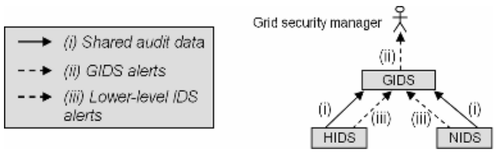


**Fig. 1.** Integration of GIDS with lower-level IDSs

GIDS integration with the other IDSs is illustrated in Fig.1. To achieve the desired security level, HIDS and/or NIDS are installed at certain grid nodes and network domains and share relevant information with GIDS. The detection of the typical host computer and network attacks is then summed with the detection of grid-specific attacks and behavior anomalies of grid users.

## 2 IDSs Integration

The integration of GIDS with lower-level IDSs can be achieved if the following requirements are supported: (a) the sending of alerts from lower-level IDSs to warn GIDS about locally detected intrusions; (b) the sending of alerts from lower-level IDSs to warn GIDS about grid attack trails; (c) the sending of audit data from HIDS to GIDS; and (d) standard communication between HIDS/NIDS and GIDS.

The lower-level IDSs must (a) alert GIDS about any intrusions detected locally in their acting domain. Alerts of any attack trails that can be correlated by GIDS with other trails to detect grid-specific attacks must also be (b) sent by the IDSs.

To identify misuse committed by grid users, GIDS must analyze their behavior and this is done with resource usage data. Host-based audit data sources are the only way to retrieve information about user activities in a grid and, therefore, HIDS are responsible for (c) sending audit data to GIDS.

As lower-level IDSs might be heterogeneous and write their alerts in different formats, interoperability is harder without the use of (d) standards. It is demanded that they speak the same language: the IDMEF message format and the IDXP protocol [3].

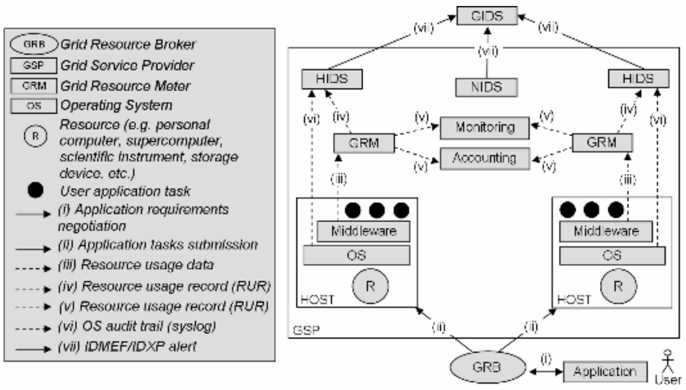

GRB Grid Resource Broker
GSP Grid Service Provider
GRM Grid Resource Meter
OS Operating System
R Resource (e.g. personal computer, supercomputer, scientific instrument, storage device, etc.)
User application task
(i) Application requirements negotiation
(ii) Application tasks submission
(iii) Resource usage data
(iv) Resource usage record (RUR)
(v) Resource usage record (RUR)
(vi) OS audit trail (syslog)
(vii) IDMEF/IDXP alert
GIDS
HIDS
NIDS
Monitoring
Accounting
GRM
Middleware
OS
R
HOST
GSP
GRB
Application
User

**Fig. 2.** Interactions between users, resources, and IDSs

The interactions between users, grid infrastructure, and the IDSs are depicted in Fig. 2. HIDS receive resource usage records from grid resource meters [4] and syslog [1] data from the operating systems. GIDS, on the other hand, receives and analyzes IDMEF/IDXP alerts sent by the various HIDS and NIDS. The content of these alerts refers to typical host and network attacks, trails of possible grid attacks, and user resource usage records used to identify the occurrences of misuse.

## 3 Case Study

To evaluate the mechanisms described in the previous section, a case study of a simulated grid environment was performed with GridSim [5]. In this simulation, user applications (*gridlets*) were scheduled for processing at the grid's hosts and IDMEF messages were generated out of gridlet resource usage records. These messages were sent to a prototype GIDS, simulating the HIDS interactions (Fig. 3). GIDS applied a feed forward neural network that was trained to identify great deviations in the demanded computational power for the gridlets submitted by each user.

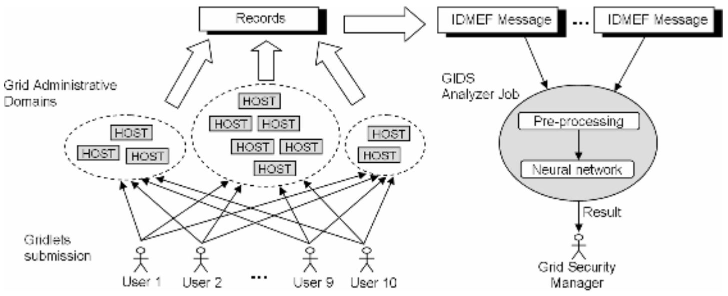


**Fig. 3.** Simulation environment

In the training and experimentation phases the net was fed with a certain volume of input data obtained from the last gridlets submitted by the users. A lower number of gridlets resulted in a lower volume of data analyzed by the net and a greater difficulty for it to learn and to identify anomalies caused by the possible intruders.

## 4 Conclusions

The purpose of this work is to describe the mechanisms necessary to integrate a GIDS with lower-level IDSs in order to provide protection against all the possible grid intrusions. As described in Section 2, the use of standard protocols and formats is focused and makes possible the interoperability of heterogeneous systems. The case study of a simulated grid and a prototype GIDS implementation described in Section 3 shows the feasibility of the mechanisms, although further experimentation is needed for a more comprehensive evaluation.

## References

1. Debar, H. Dacier, M., Wespi, A.: Towards a Taxonomy of Intrusion-Detection Systems. Int. J. Computer and Telecommunications Networking, Vol. 31, No. 9 (1999) 805-822
2. Schulter, A., Navarro, F., Koch, F., Westphall, C.: Towards Grid-based Intrusion Detection. In: Proc. 10th Network Operations and Management Symposium, Canada (2006)
3. Brandão, J. E., Fraga, J. S., Mafra, P.: A New Approach for IDS Composition. In: Proc. IEEE 2006 International Conference on Communications, Istambul, Turkey (2006)
4. Lim, D. et al.: MOGAS: A Multi-Organizational Grid Accounting System. In: Int. J. on Information Technology, Singapore, Vol. 11, No. 4 (2005)
5. Sulistio, A., Poduvaly, G., Buyya, R., Tham, C.: Constructing A Grid Simulation with Differentiated Network Service Using GridSim. In: Proc. of the 6th Int. Conference on Internet Computing, Las Vegas, USA (2005)

# ORCA – Towards an Organic Robotic Control Architecture*

Florian Mösch[1], Marek Litza[1], Adam El Sayed Auf[1], Erik Maehle[1], Karl E. Großpietsch[2], and Werner Brockmann[3]

[1] University of Lübeck, Institute of Computer Engineering, Lübeck, Germany
[2] Fraunhofer Institute of Autonomous Intelligent Systems, St. Augustin, Germany
[3] University of Osnabrück, Institute of Computer Science, Osnabrück, Germany

**Abstract.** We are working on a modular and self-organizing component based software architecture for autonomous mobile robots. To reach a certain degree of fault-tolerance without analyzing all kinds of possible error conditions, "Organic Components" will be added to the system to detect recognize variations from a defined "normal state" and then try to find counter measures. Once an action is identified to help in certain situations, the component will store that information and use it if a similar situation is reached later. The system will be self-optimizing and self-healing. We started to evaluate adaptive filters as one possible implementation for components detecting deviations from the normal system state.

## 1 Requirements for an Organic Robotic Control Architecture

We develop and evaluate a new software architecture for mobile robots. Autonomous acting robots are complex systems interacting in a unstructured environment. To make them more reliable, state of the art fault-tolerance methods can be used. While the results may be satisfying, the costs for such a system grow high. Often, a compromise must be found between fault-tolerance and robustness on the one hand and the costs on the other. This applies especially where systems are designed for a mass market like entertainment systems or house-keeping robots. In this area, current fault-tolerance methods would rise the system cost unacceptable high. Contrariwise, robots that shall navigate in an unstructured, changing environment like typical households are, have to be tolerant against unforeseen situations, defects, etc. What makes conventional fault-tolerant systems so expensive is the analysis of the faulty situations that could occur.

Instead of analyzing all these erroneous situations that could occur, we define the "normal" or "healthy" situation in which our system shall stay. The system shall detect when it leaves this defined "good" working area and find back to a normal state. Therefore the system must know which possible actions exist

* This work is supported by DFG (SPP 1183, MA 1412/7-1).

to change its own state. Our system will learn like an organic system on its own which action is most appropriate in a particular "bad" situation it may be confronted with.

Another problem we address is the growing complexity of software for robotic systems. Typical robot software systems are built in a modular way and are hierarchically organized. Software modules shall be reusable and easy to "compose" to new systems. Our software components can describe themselves and autonomously combine each other like peers in a decentralized peer to peer network.

To monitor other components and change their parameters or structure if the system leaves its defined normal working area, "Organic Components" will be added to this self-organizing structure.

Our software architecture is built from components that use a simple interface to exchange data or change parameters of a component. The interface is similar to the one of an MCA2 module [4]. Container components can enclose a group of components. Systems can so be organized hierarchically.

Instead of composing a complex software system manually from these modules, the components will organize themselves, i.e. the connections between them will not necessarily be defined by the system designer but whenever possible are established autonomously.

Each component is self-explanatory. It announces to other components which data it produces and which data it needs for its own operation. Matching components will be connected like peers in a peer to peer network.

Simple systems can be built using primitive modules that sample sensor information, process the sensor data and control actuators. Complementing modules could change an actuator's parameter to optimize a robot for speed or energy consumption. Disabling, removing or replacing that component later would leave the robot fully functional. If an actuator fails, but redundant parts exist, an optimization component could be (re)activated to change parameters so that the robot works best with the remaining parts. Activating these components in situations that can be detected as "not normal" can lead to self-optimizing and self-healing systems.

## 2 Self-organizing Components

Components are self-explanatory and can define the necessary connections on their own. A component's description contains a description of all signals and parameters of the component. A signal description consists of type and a name, optionally "normal" values or ranges can be defined. Other components can interpret this data and take countermeasures if they detect discrepancies.

In order to self-organize, the components need to share a common interface to interact and e.g. publish and parse their descriptions. Two kinds of components will be generated from a common code-base: components that interact locally via function calls within one process and distributed components that use a network layer and build a "real" peer to peer network. All implementations of one component share one identical interface description which is similar to the components "self-description".

## 3 "Organic" Components

In order to accomplish an organic system, the architecture will be supplemented by generic "organic components". These components can be loaded statically or dynamically when resources are available and then monitor other components behavior. Like the cells of an immune system, our organic components can monitor each other and detect when they look "strange" i.e. their parameters are not in their normal range as defined in a components self-description. ([5],[6],[7])

When an organic component detects a faulty component it tries to change the components parameters to bring the component back to normal operation. If such an attempt is successful, the component may store this information and use it when a similar situation occurs later. This is similar to a biological immune system which "learns" how to fight common diseases.

Another approach to detect a components malfunction is to monitor its behavior over time independently from its defined normal state.

Adaptive filters [8] are well suited to detect a change in a signal's long term behavior. Generic adaptive filters may be added to the system, monitor a component's signals and activate other components when a component appears to change its behavior.

## References

1. German Organic Computing Initiative, http://www.organic-computing.de
2. Brockmann, W., Maehle, E., Mösch, F.: Organic Fault-Tolerant Control Architecture for Robotic Applications. 4th IARP/IEEE-RAS/EURON Workshop on Dependable Robots in Human Environments, Nagoya, Japan (2005)
3. Schoeler, T., Mueller-Schloer, C.: An observer/controller architecture for adaptive reconfigurable stacks. In: M. Beigl, P. Lukowicz (eds.): Systems Aspects in Organic and Pervasive Computing. ARCS2005. Springer LNCS 3432, Berlin, Heidelberg (2005) 139–153
4. Scholl, K.U., Albiez, J., Gassmanni, B.: MCA – An Expandable Modular Controller Architecture. In: 3rd Real-Time Linux Workshop. Milano, Italy (2001)
5. Greensmith, J., Cayzer, S.: An artificial immune system approach to semantic document classification. 2nd Int. Conference on Artificial Immune Systems ICARIS 2003, LNCS 2787, Springer-Verlag (2003) 136–146
6. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. Proc. of 1994 IEEE Symposium of Security and Privacy, IEEE Computer Society Press (1994)
7. Lee, D., Jun, H., Sim, K.: Artificial immune system for realization of cooperative strategies and group behavior in collective autonomous robots. Proc. of the 4th Int. Symposium on Artificial Life and Robotics (1999) 232–235
8. Grosspietsch, K.-E.: Adaptive Filters for the Dependable Control of Autonomous Robot Systems, 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) – Workshop 16 (2005) 284b.

# Posters

# Active Element Network with P2P Control Plane

Michal Procházka[1,3], Petr Holub[1,3], and Eva Hladká[2,3]

[1] Institute of Computer Science
[2] Faculty of Informatics,
Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic
[3] CESNET z. s. p. o., Zikova 4, 160 00 Praha 6, Czech Republic
michalp@ics.muni.cz, hopet@ics.muni.cz, eva@fi.muni.cz

**Abstract.** Multi-point data distribution for synchronous multimedia communication poses interesting problem for networking environment and it is usually implemented by either native or virtual multicast. We describe and evaluate a scalable network of Active Elements (AE) that implements user-empowered virtual-multicast overlay network for synchronous data distribution and processing in the network. The AE network is based on strict separation of control plane and data plane. The control plane is organized as peer-to-peer network in order to achieve robustness and user-empowered approach while sacrificing efficiency to some extent. The data plane which handles the actual data distribution is optimized for efficiency and allows pluggable implementation of different distribution models. Each plane solves robustness and scalability by own.

We present a prototype implementation with control plane based on JXTA peer-to-peer substrate. Control plane is responsible for managing and controlling AEs in the AE network and also to gather information about state of AE network for the data distribution schemes. Prototype consists of stand-alone modular application which implements server side and also client side. We have evaluated control plane behavior, scalability, robustness, and efficiency. In order to achieve better results in robustness and scalability of control plane, we have had to do specific changes into the JXTA peer-to-peer substrate. The results show that AE network can grow significantly without any restriction because there is linear dependency between number of exchanged messages and number of AEs in the network. Growing number of the AEs in the network does not have influence on the number of messages send and receive by each single AE. Robustness of the AE network also reaches very high level. As a consequence to our changes in JXTA, the latency when AE recognizes failure of the neighbor AE decreased from 60 s to about 1 s. Both attributes can be adjusted to better fit host environment and needs.

Generally AE network is used for any multimedia applications, that rely on RTP/UDP data transmission like MBone videoconferencing tools (RAT, VIC) and shared whiteboard (WB/WBD). Also unidirectional "broadcasting" applications like VideoLAN Client can utilize AE networks for distribution of their data.

# A Monitoring Infrastructure for the Digital on-demand Computing Organism (DodOrg)

Rainer Buchty

Universität Karlsruhe (TH), Institut für Technische Informatik, 76128 Karlsruhe, Germany
buchty@ira.uka.de

**Abstract.** The Digital on-demand Computing Organism (DodOrg) is a novel system concept based on biological concepts. Major part of DodOrg is a sophisticated monitoring infrastructure spanning all system layers from hardware to application, providing necessary information for system surveillance and the adaptive processes triggered by an organic middleware and the low-power planning.

The Digital on-demand Organism (DodOrg) [BBB+06] is an ambitious project funded by the *Deutsche Forschungsgesellschaft (DFG)* within the priority program 1183 (SPP1183) "Organic Computing". The idea behind DodOrg is to adopt biological concepts for use within digital architectures.

DodOrg is based on a heterogeneous, adaptive multicore architecture comprising of several processing elements (*cells*) connected through a peer-to-peer network. Hence each cell contains dedicated router functionality in addition to its core functionality.

Independent monitoring is distributed over the entire system architecture by embedding monitoring functionality into each part of the system. Hence, monitoring is also part of every cell. Basic preprocessing (semantic compression) on hardware level minimizes communication overhead. Aggregate monitoring functions can be realized using dedicated monitoring cells or monitoring organs created from arbitrary processing elements.

Monitoring must closely interface with the planning infrastructure, i.e. organic middleware and low-power planning, using dedicated message and communication types (defining content, frequency, and level of autonomy) and communication modes (unicast, multicast, or broadcast).

## References

[BBB+06] Jürgen Becker, Kurt Brändle, Uwe Brinkschulte, Jörg Henkel, Wolfgang Karl, Thorsten Köster, Michael Wenz, and Heinz Wörn. Digital On-Demand Computing Organism for Real-Time Systems. In Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, and Erik Maehle, editors, *ARCS'06 Workshop Proceedings*, pages 230–245. Gesellschaft für Informatik e.V., March 2006. Lecture Notes in Informatics (LNI) P-81, ISBN 3-88579-175-7.

# Autonomic Network Management for Wireless Mesh and MANETs

Shafique Ahmad Chaudhry, Ali Hammad Akbar, Faisal Siddiqui, and Ki-Hyung Kim

Division of Information and Computer Engineering, Ajou University, Korea
{shafique, hammad, faysal, kkim86}@ajou.ac.kr

**Extended Abstract.** The realization of the envisioned ubiquitous computing paragon has resulted into new breeds of hybrid networks, e.g., u-Zone [1]. In u-Zones, a high-speed wireless mesh is used as the backbone—allowing MANETs nodes to be connected as a stub network. Hybrid in their nature, these networks belong to the generation of networks that deal with high levels of heterogeneity, mobility, and variability.

The existing network management architectures address either MANETs [2] or mesh [3] in isolation, and do not meet the dynamic paradigm and fluctuating requirements for u-Zone networks. Highly dynamic node and network configurations, management and network tasks excessively overlapping, mobility, unpredictable environment, and lack of centralized control make continuous human-supervised management almost impossible. The catering of such challenges, plus the continuous growth factor, demand an autonomous management architecture that facilitates all self-management aspects. The challenges of dynamic configuration, on-the-fly resource allocation, and reliable service provisioning can hardly be met without embedding a robust context-aware mechanism within network management framework.

Our contribution is to provide an autonomic, policy-based, context-aware and hierarchical manager-agent framework that is adaptable and robust to network variations, node failures, and dynamic user requirements. The policy-based framework provides flexibility and scalability by enabling the network operators to specify the high level networking requirements, in terms of configuration, healing, accounting, optimization and general policies. The context information is gathered, analyzed and synthesized to predict the current and future network behavior. Correspondingly, required management functions are identified, invoked, and executed through the mobile agents, on the managed nodes. The system parameters optimize themselves by estimating new transitions and reacting accordingly, with the least human intervention.

## References

1. Chaudhry, S.A., Akbar, A.H., Siddiqui, F.A., Yoon, W.S.: Autonomic Network Management for u-Zone Network. UbiCNS, Korea, (2005).
2. Yong-Lin, S., DeYuan, G., Jin, P., PuBing, S.: A Mobile Agent and Policy-based Network Management Architecture. Proceedings, ICCIMA 2003, (2003) Page(s): 177- 181.
3. Minseok, Oh.: Network Management Agent Allocation Scheme in Mesh Networks. Communications Letters, IEEE Volume 7, Issue 12, (2003) Page(s): 601 – 603.

# Author Index