Teruo Higashino (Ed.)

LNCS 3544

Principles of Distributed Systems

8th International Conference, OPODIS 2004
Grenoble, France, December 2004
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3544

Teruo Higashino (Ed.)

Principles of Distributed Systems

8th International Conference, OPODIS 2004
Grenoble, France, December 15-17, 2004
Revised Selected Papers

Springer

Volume Editor

Teruo Higashino
Osaka University
Graduate School of Information Science and Technology
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
E-mail: opodis@ist.osaka-u.ac.jp

Library of Congress Control Number: 2005928959

CR Subject Classification (1998): C.2.4, D.1.3, D.2.7, D.2.12, D.4.7, C.3

ISSN 0302-9743
ISBN-10 3-540-27324-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-27324-0 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11516798 06/3142 5 4 3 2 1 0

Preface

Preface

Sponsors

Organization

With the support of

Laboratory of ENS Génie Industriel and INPGrenoble, France

Laboratory of CNRS, IMAG, INPGrenoble, INRIA and UJF, France

University of Osaka, Japan

Table of Contents

Invited Session I

Session I (Design of Distributed Systems I)

Session II (Ad-Hoc Networks and Mobile Agents)

Session III (Grid and Networks)

Invited Session II

Session IV (Security)

Session V (Distributed Algorithms)

Session VI (Self-stabilization)

Session VII (Design of Distributed Systems II)

Session VIII (Sensor Networks)

Session IX (Task/Resource Allocation)

Protocol System Integration, Interface and Interoperability

David Lee[1], Christine Liu[2], and Mihalis Yannakakis[3]

[1] Department of Computer Sciences and Engineering, Ohio State University
[2] Bell Labs Research China, Lucent Technologies
[3] Department of Computer Science, Columbia University

Abstract. Heterogeneous network protocol systems are integrated together to fulfill complex tasks and their interoperability is a major hurdle for the network reliability and quality of services. We identify a new equivalence relation of states that preserves the integrated system interface behaviors. Based on this state equivalence we study the minimization of the system components with respect to their interfaces and design an efficient polynomial time minimization algorithm. We apply our technique to GMPLS protocols and obtain a significant state space reduction. We discuss integrated protocol system verification and interoperability testing with the minimized state system without resorting to the global state space information.

1 Introduction

With the rapid growth of Internet, new protocols are being developed and integrated into the existing network systems, such as GMPLS (Generalized Multi-Protocol Label Switching, an IETF Standards for all optical network management and interface with Internet, ATM network, and other user networks), OUNI (an OIF Standard for Optical User Network Interface), and VoIP (Voice over IP). Heterogeneity is a prominent feature of integrated network systems, and interoperability is ubiquitous and has become a major hurdle for system reliability and quality of services. When two or more system components are integrated to interface with each other to perform a required task the capability to operate as desired is called interoperability, which is an essential aspect of the correctness of integrated protocol systems. Interoperability testing is to check the interfaces and interoperations among integrated system implementations, and verification is to analyze the system design for integration and interfaces. The focus of both analyses is the interface among the integrated system components.

However, the number of states of integrated systems is often too large for a formal analysis due to the well-known state explosion problem. For our application, we want to reduce the state space by hiding the internal behaviors of the components as much as possible, while preserving completely the system interfaces. More formally, given a component or a subsystem, some of whose transitions are *interface transitions* while the rest are *internal* transitions, we want to obtain a reduced system that: (1) Has the

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 1–19, 2005.

same interface transitions, and (2) Has the same sequences of interface transitions, as the original system. Although this may appear to call for minimization of a nondeterministic system (because of the internal transitions, the interface behavior of the system may be in general nondeterministic), a problem that is known to be computationally hard, we show that this is not the case here. After an elimination of internal states of the system without affecting its interface behaviors, we define an appropriate state equivalence relation and present an efficient algorithm to compute it. Based on this state equivalence we can reduce the overall state space while preserving exactly the integrated system interfaces. This state space reduction facilitates formal verification and interoperability testing of integrated systems. We design and implement a polynomial time algorithm for the state space reduction, discuss its applications to integrated system verification and interoperability testing, and report the experimental results on GMPLS protocols.

In Section 2 we give some background on the model for integrated protocol systems, define interface graphs and state formally the problem. In Section 3 we study interface graphs, define state interface equivalence, and analyze its properties. In Section 4 we present a polynomial time algorithm for state equivalence and minimization of interface graphs, and its extension to minimization of integrated systems. The algorithm is applied to LMP of GMPLS for state space reduction and the experimental results are reported in Section 5. The properties of the minimized interface graphs are further studied in Section 6 for the applications to the verification and interoperability testing of integrated systems.

2 A Formal Model

A protocol system consists of a set of communicating components. Each component is represented by a finite state transition system or a finite state machine, i.e., it consists of a set of states, one of which is designated as the initial state, and a set of transitions labeled by actions, or by inputs and outputs. Some of the transitions involve interaction with other components (eg. sending or receiving messages) and are called *interface transitions*, while others represent internal local actions of the components and are called *internal transitions*. For the purposes of the problems studied in this paper, it does not matter whether transition systems or finite state systems are used as the underlying model; the issues and the algorithms are the same in both cases.

Example 1. The Link Connectivity Verification (LCVA) module of the active node of the Link Management Protocol (LMP) of GMPLS is shown in Fig. 1. It contains 5 states; the initial state is *Down*. Each transition is labeled by a pair *a/b* where *a* is the event (input) and conditions that cause the transition to occur, and *b* is the effect of the transition (output or actions that take place as a result of the transition). For example, the transition from state Test1 to state Test2 causes the sending of a message *Testmsg*, as indicated by the label *!Testmsg*. The label on the arrow from Test2 to Down is a shorthand for two transitions: one transition takes place if a message *TestStatusFailure* is received, and the other transition takes place if the *Timerexpiry* event occurs, a local internal event of the component. This component has 3 interface transitions: the abovementioned two transitions and the transition from Test2 to Up/Free. The other transitions are all internal transitions. □

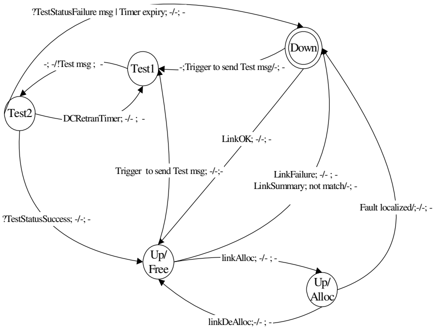


Fig. 1. A module from LMP of GMPLS

The components of a protocol system are integrated together to fulfill required tasks. The joint behaviors of the system are represented by the composition of the different components formed by taking their Cartesian product in the usual way [6,7]. The set of states is the Cartesian product of the components' state sets; the initial state is the tuple of the initial states. A transition of the composed system corresponds either to a local (internal) transition of a component (i.e. all local states remain the same in this case except for that of the component that makes the local transition), or to the simultaneous execution of matching interface transitions of different components, eg corresponding to the sending and receiving of a message. (A model may include separate components for the channels or other communication media, to separate the occurrence of the sending of a message with the reception at the other end.) We are interested only in the portion of the system that is reachable from the initial state.

The Cartesian product often leads to the familiar state explosion problem; the number of states of the product is too large. To cope with this problem various techniques have been developed in the areas of system verification, validation and testing. One can use heuristics to *prune* search state space and *random walks* for efficient exploration yet without recording the searched states. *Symbolic representation* of the state transition graphs and efficient algorithms for their manipulation avoid explicit construction of the state graph of the concurrent system. Concurrent software tends to be less structured and asynchronous, and *partial order reduction* reduces the number of interleaving sequences for analysis. *Compositional reasoning* exploits the modular structure of complex systems and conducts analysis on the components separately of a decomposed system with an assume-guarantee reasoning. *Abstraction* deals with data portion of systems to reduce the complexity of model checking. *On-line minimization* reduces transition system state space on-the-fly without constructing the whole state space [1,12] and *compositional minimization* performs stepwise bisimulation reduction steps [4]. Protocol systems often contain replicated components, and this system *symmetry* is exploited to reduce the state explosion. *Induction* analyzes families of parameterized protocol systems by providing an invariant process that represents a large number of systems with different parameter values. Detailed references can be found in [2,7]. For program testing, [11] uses an incremental reachability graph for test sequence generation for concurrent programs. Then a graph reduction is applied with path preservation (for test generation), which includes: collapse, τ-state elimination, and prune.

For the verification and interoperability testing of integrated systems, the focus is on the interface transition sequences, which characterize the system interfaces. Observing that often many of the transitions are internal transitions, we investigate state space minimization such that: (1) The interface transition sequences remain unchanged; (2) State and path information can be either preserved or retrieved (yet without blowing up the search space) for verification of integrated system properties and for constructing executable interoperability testing sequences. We can perform such minimization to the individual components at the outset before composing them, and then continue minimizing partial products iteratively as they are being computed.

2.1 System Integration and Interface Graphs

Consider a graph G of a transition system that may be an individual component or a product of several components. A subset of the transitions is specified as *interface transitions* while the rest as *internal* transitions. Let Σ be the set of interface transitions. The behaviors of the system can be represented by all the possible executable sequences (scenarios) $\boldsymbol{t}$ from the initial state v_{init} of the system graph G. Practical experiences show that integrated system interoperability problems manifest themselves when components are interfacing with each other, that is, while interface transitions are executed. Change internal transitions to τ-moves ("silent" or "invisible" transitions). Such a graph is called an *interface graph.* In terms of revealing interoperability problems, two scenarios with an identical interface transition sequence (with different τ-moves in between) provide the same information and, therefore, we do not care about τ-moves in a scenario. Specifically, let $t = t_1, t_2, ..., t_r$ be a scenario of G. Its *projection* $\pi(t)$ is obtained by removing all the τ-moves, and is a sequence of interface transitions only, i.e. a string over Σ. Two scenarios r and t are equivalent if and only if $\pi(\text{r}) = \pi(t)$. Therefore, all the integrated system behaviors are represented by the set of distinct sequences of interface transitions: $S(G)=\{\pi(t) : t \in \boldsymbol{t}\}$ where $\boldsymbol{t}$ is the set of all the scenarios from G and can be infinite. Thus, $S(G)$ is a language over the alphabet Σ consisting of all the interface transitions of G. To reduce the system complexity while maintaining the system interface behaviors, we want to obtain a reduced interface graph G^* that has the same set of interface transitions and is *interface equivalent* to the original interface graph G, i.e., $S(G^*)=S(G)$. This is the trace equivalence [14] or language equivalence [8] with respect to the alphabet Σ of interface transitions.

Note that different interface transitions of the system may have the same action (or Input/Output) label. However, we treat them as distinct because they represent execution of the action in distinct contexts, and these may have quite different implications for the integrated system interoperability testing and verification. For example, in testing of a component, we may want to generate tests that exercise all the interface transitions of the component. If we were to reduce the component while preserving only equivalence with respect to action (or I/O) labels of the transitions, then we would lose useful paths and may even eliminate some of the interface transitions, hence the reduced graph would not be sufficient for the task. Consequently, in our minimization we want to preserve the set $S(G)$ of all interface transition sequences.

In summary, we have an interface graph with τ -edges and distinct interface transitions, and we want to minimize it with respect to trace (language) equivalence. G can be viewed as an automaton whose transitions are labeled by elements of the alphabet Σ or τ, and all states are regarded as accepting. It is a nondeterministic automaton because of the τ transitions. Recall that nondeterministic automata do not have a unique minimum automaton in general, and moreover, minimization is PSPACE-complete [8]. We will show however that in this case we can do this efficiently.

3 Minimization of Interface Graphs

We propose a reduction by merging states while preserving the interface transition sequences and also the needed state and path information. We present a polynomial time algorithm for the reduction.

We use the standard procedure of merging two nodes: they are merged into one node that inherits all the incoming and outgoing edges of the two merged nodes.

We first derive necessary and sufficient conditions for a pair of nodes u and v to be merged while preserving interface equivalence. Recall that all the nodes in an interface graph are reachable from the initial node v_{init}. A node v is *τ-reachable* from node u if there is a path of τ-move edges from u to v. Given a node u, its *successor* nodes, denoted by succ(u), are all the τ-reachable nodes from u, and its *predecessor* nodes, denoted by pred(u), are all the nodes from which u is τ-reachable. Let S be the set of all the start nodes of interface transitions and let E be the set of all the end nodes of interface transitions and also the initial state v_{init}; in effect, we regard v_{init} as the end state of an artificial interface transition that starts the system. Define Ssucc(u)=succ(u) $\cap$ S, and Epred(u)=pred(u)$\cap$E. Ssucc(u) is the set of all the successors of u, which are the start nodes of an interface transition. Epred(u) is the set of all the predecessors of u, which are either the initial node or an end node of an interface transition.

Proposition 1. (*Node Merging Condition*) Given an interface graph G, merging two nodes u and v yields an interface equivalent graph if and only if: every node in Ssucc(u) is τ-reachable from every node in Epred(v) and every node in Ssucc(v) is τ-reachable from every node in Epred(u).

Sketch of Proof. Obviously node merging can only produce additional interface transition sequences. Thus, we only need to verify that a merging of nodes does not introduce any new interface transition sequences. It does introduce a new interface transition sequence if and only if the merging connects two disconnected interface transition subsequences: one from the initial node to v (u) and the other starting from u (v). This is the case if and only if there is $y\in$ Epred(v) (or Epred(u)) and $x\in$ Ssucc(u) (or Ssucc(v)) but x is not τ-reachable from y. □

Corollary 1. For a strongly connected component (SCC) of τ-moves in an interface graph, all the nodes in the SCC can be merged into one node to obtain an interface equivalent graph. □

The condition of Proposition 1 is symmetric but not transitive: it may be the case that pairs *(u,v)* and *(v,w)* satisfy the condition, but the pair *(u,w)* does not. Furthermore, node merging is not independent, i.e., merging a pair of nodes may affect the validity of merging other pairs of nodes. Specifically, suppose that two pairs of nodes, u and v, u' and v', satisfy the Node Merging Condition in Proposition 1. However, merging u and v may change the topology of the graph G so that u' and v' do not satisfy the same condition anymore.

Example 2. In Fig 2, one can easily check that merging nodes w and u (u and v) is valid. However, merging both pairs is invalid; it would introduce a new interface transition sequence *ab*. Obviously, merging w and u would disable the Node Merging Condition of u and v. □

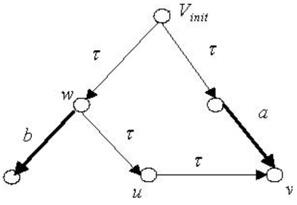


Fig. 2. An Example

Consequently, we cannot first identify all the pairs of nodes, which satisfy the Node Merging Condition, and then merge all of them. Instead, after merging a pair of nodes, we would have to find another pair of nodes that can be merged, and repeat the process iteratively.

The following is obvious from Proposition 1:

Corollary 2. Suppose that node pair u and v is invalid for merging, i.e., it does not satisfy the Node Merging Condition in Proposition 1. Then it remains invalid for merging after merging other valid node pairs. □

However, from Example 2, a pair of nodes may lose its validity for merging after merging other valid pairs. We will show that a simple preprocessing procedure that eliminates internal nodes allows valid pairs to be merged independently.

3.1 Ubiquitous Interface Graph and Church-Rosser Property of Node Merging

Consider a node u, which is only incident to τ-moves. If all the incident τ-moves are incoming edges, then u is a sink node of τ-moves. We can remove u along with all the incident τ-moves, resulting in an interface equivalent graph. Similarly, we can remove source nodes (except for the initial node), which are only incident to outgoing τ-moves. The resulting graph contains two types of nodes: (1) Incident to at least one interface transition; or (2) incident to only τ-moves yet neither sink nor source node. We can remove type (2) node u as follows. For each pair of incoming τ-move $p \rightarrow u$ and outgoing τ-move $u \rightarrow q$, add a τ-move $p \rightarrow q$ if it is not there, and remove u along with all its incident τ-moves. Obviously, the resulting graph is interface equivalent to the original one. Since each operation reduces the number of nodes, we can repeat the process until all the nodes are incident to at least one interface transition. Note that the number of τ-moves may increase in the worst case. Yet our main concern in dealing with state explosion is the number of nodes.

In summary, given an interface graph, we can conduct a simple preprocessing to reduce it to an interface equivalent graph where each node is incident to at least one interface transition. We call such an interface graph *ubiquitous* (interface occurs with every node – everywhere). From now on we assume that all the interface graphs are ubiquitous.

Lemma 1. In a ubiquitous interface graph, the Node Merging Condition is: (1) Invariant with respect to merging of valid node pairs, and (2) Transitive.

Proof. We show (1); claim (2) follows from (1) and Corollary 3. Let the given interface graph be G and the resulting interface graph be G' after merging a valid node pair u and v. From Corollary 2, a pair of nodes u' and v' remains invalid for merging in G' if it was *not* in G. We now show that if they were valid for merging in G, then they remain valid in G'. Assume on the contrary that u' and v' become invalid for merging in G'. Then from Proposition 1 there exist $y \in \mathrm{Epred}(v')$ and $x \in \mathrm{Ssucc}(u')$ in G' such that x is not τ-reachable from y (the symmetric condition can be handled by the same argument). Since u' and v' were valid for merging in G, from Proposition 1, either $y \notin \mathrm{Epred}(v')$ or $x \notin \mathrm{Ssucc}(u')$ in G; otherwise, since x was τ-reachable from y, it also is in G', a contradiction. There are three cases.

Case 1. $y \notin \mathrm{Epred}(v')$ and $x \in \mathrm{Ssucc}(u')$ in G. Since $y \notin \mathrm{Epred}(v')$ in G and $y \in \mathrm{Epred}(v')$ in G' there is a path of τ-moves from y to v, and a path of τ-moves from u to v', and merging nodes u and v makes v' τ-reachable from y. Since G and G' are ubiquitous, there is an interface transition incident to v', and there are two cases: (A) $v' \in$ E; and (B) $v' \in$ S. Case (A) Since $v' \in \mathrm{Epred}(v')$, $x \in \mathrm{Ssucc}(u')$, and node pair u' and v' was valid to be merged in G, x is τ-reachable from v' in G and hence in G'. Since v' is also τ-reachable from y in G', x is τ-reachable from y in G', a contradiction. Case (B) Since $y \in \mathrm{Epred}(v)$, $v' \in \mathrm{Ssucc}(u)$ and node pair u and v was valid to be merged in G, v' was τ-reachable from y in G, a contradiction.

Case 2. $y \in \mathrm{Epred}(v')$ and $x \notin \mathrm{Ssucc}(u')$ in G. Since $x \notin \mathrm{Ssucc}(u')$ in G and $x \in \mathrm{Ssucc}(u')$ in G', there is a path of τ-moves from u' to v and a path of τ-moves from u to x, and merging of u and v makes x τ-reachable from u'. Since G is ubiquitous, there is an interface transition incident to u'. There are two cases: (A) $u' \in$ S; and (B) $u' \in$ E. Case (A) Since $u' \in$ Ssucc(u'), $y \in$ Epred(v'), and node pair u' and v' could be merged in G, u' was τ-reachable from y in G and hence is also in G'. Since x is τ-reachable from u' in G', x is τ-reachable from y in G', a contradiction. Case (B) Since $u' \in$ Epred(v), $x \in$ Ssucc(u), and node pair u and v could be merged in G, x was τ-reachable from u' in G, a contradiction.

Case 3. $y \notin$ Epred(v') and $x \notin$ Ssucc(u') in G. Since merging node pair u and v makes $x \in$ Ssucc(u') and $y \in$ Epred(v') in G', in graph G there were paths of τ-moves: from y to v, from u to v', from u' to v, and from u to x. See Fig 3. Therefore, $y \in$ Epred(v) and $x \in$ Ssucc(u) in G. Since node pair u and v could be merged in G, x was τ-reachable from y in G and hence also in G', a contradiction. □

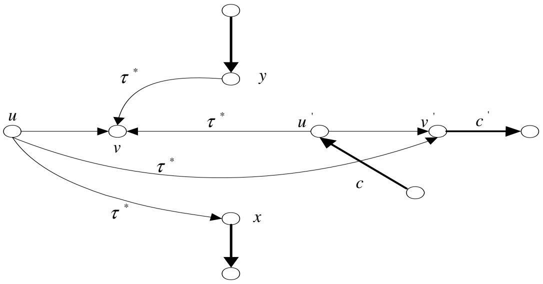


Fig. 3

Corollary 3. On a ubiquitous interface graph, the operation of merging node pairs, which satisfy the Node Merging Condition, has the Church-Rosser property, i.e., they can be merged in an arbitrary order. □

Definition 1. In a ubiquitous interface graph G two nodes u and v are *interface equivalent*, denoted by $u \equiv v$, if they satisfy the Node Merging Condition; equivalently, $u \equiv v$ if merging them yields an interface-equivalent graph. □

Remarks (State Equivalence Relations)

(1) By Lemma 1, node interface equivalence is indeed an equivalence relation. On the other hand, from Example 2, this is not the case if there are internal nodes, and that is why we defined it only for ubiquitous interface graphs. (2) Two interface equivalent nodes u and v may well not be trace- (or observationally) equivalent: there may be a path starting at u whose projection is a sequence of interface transitions that does not have a corresponding path from v. Thus, although reduction by trace or observational equivalence also preserves the interface language, it is a weaker reduction than that from interface equivalence and may not merge some states.

(3) There is a variety of equivalence relations defined in the literature (see [3] for a comprehensive list). As far as we know, state interface equivalence is different. One

observation is that it is common in the literature to identify states with processes; namely, a state *u* is identified with the process *P(u)* that has *u* as its initial state, and equivalence of two states *u* and *v* (with respect to some equivalence notion) is defined as equivalence of the two processes *P(u)* and *P(v)*. However, as we noted, even though the processes *P(u)* and *P(v)* may not be trace-equivalent, still we may be able to merge states *u* and *v* while preserving trace equivalence for the whole graph (starting from the initial state). □

By the Church-Rosser property of node merging according to interface equivalence, we can merge all nodes in each equivalence class. The resulting graph *G** has obviously the same interface transitions and is interface equivalent to *G*. The graph is unique up to the names of the nodes and the addition or deletion of transitive τ edges. We show furthermore, that *G** is *the minimum* graph with these properties.

Theorem 1. For a ubiquitous interface graph *G*, let *G** be the interface-equivalent graph obtained by merging interface-equivalent nodes. Let *G'* be any interface graph that has the same set Σ of interface edges (i.e. its interface edges are in 1-1 correspondence with those of *G*) and that is interface-equivalent to *G*. Then *G'* has at least as many nodes as *G**.

Sketch of Proof. Let *u,v* be two distinct nodes of *G**. We distinguish cases depending on whether *u,v* are in E or S. We will show here only the case $u \in E, v \in S$; the other cases can be argued similarly. Let *a* be an interface edge into *u* and *b* an interface edge out of *v*. Let *u'*,*v*' respectively be the tail and head of the corresponding edges *a,b* in *G'*. We now argue that $u' \neq v'$.

Suppose that *u'*=*v'*. Since *G'* contains an interface sequence that contains the subsequence *ab*, the graph *G** must have a τ path from *u* to *v*. Since *u,v* are not interface equivalent (otherwise they would have been merged), there exist *x* in Epred(*v*), and *y* in Ssucc(*u*) such that there is no τ path from *x* to *y*. Let *c* be an interface edge into *x* and *d* an interface edge out of *y*, and let *x'*,*y'* be the tail and head of the corresponding edges in *G'*. Since *G** has an interface sequence that contains the subsequence *cb,* the same must be true for *G'*, hence *x'*∈Epred(*v'*). Similarly since *G** has an interface sequence that contains the subsequence *ad*, we must have *y'*∈ Ssucc(*u'*). Since *u'*=*v'*, there is a τ path from *x'* to *y'*, and therefore *G'* has an interface sequence that contains the subsequence *cd,* whereas *G** (and hence *G)* does not. □

We remark that it may be possible in some cases to duplicate some of the interface edges and construct thereby an equivalent graph with fewer nodes. For example, if all the incoming edges of an S node *v* are τ edges, then we can eliminate *v* and add appropriate edges from its predecessors to its successors. This however will introduce multiple copies of an interface edge, which may impact the use of the graph for testing and verification: Consider for example the problem of generating tests to cover all interface edges – now we would have to cover more edges. We defer further discussion to the full paper.

4 Minimization Algorithm

Given an interface graph, we first conduct a preprocessing to remove all the nodes, which are incident to τ-moves only, obtaining an interface equivalent ubiquitous graph. We then shrink SCCs of τ-moves, obtaining a Directed Acyclic Graph (DAG) with respect to τ-moves. We can then check interface equivalence of every pair of nodes and merge the equivalent ones. This naïve algorithm costs $O(n^4)$ where n is the number of nodes of the interface graph. We now present an efficient algorithm with a cost of O(mn) where m is the number of edges of the interface graph. Recall that S is the set of start nodes of interface transitions and E consists of the set of end nodes of interface transition and the initial node; since the graph is ubiquitous, every node belongs to S or E or both.

Lemma 2. Two nodes u and v are interface equivalent, i.e., $u \equiv v$, if and only if:

Case 1. $u,v \in$ S: Epred(u)=Epred(v);
Case 2, $u,v \in$ E: Ssucc(u) = Ssucc(v);
Case 3, $u \in$ E, $v\in$ S: v is τ-reachable from u, and
every node in Ssucc(u) is τ-reachable from every node in Epred(v).

Sketch of Proof.

Case 1. If Epred(u)=Epred(v), then each node in Ssucc(u) is τ-reachable from those in Epred(u) and hence in Epred(v), and the Node Merging Condition is satisfied. Conversely, assume that Epred(u)$\neq$Epred(v) and, without loss of generality, assume that there is a node $y\in$ Epred(v) but $y\notin$ Epred(u). Since $u\in$S, $u\in$Ssucc(u), and it is not reachable from y$\in$ Epred(v). From Proposition 1, u and v are not interface equivalent.

Case 2 can be proved similarly.

Case 3. From Proposition 1, the conditions are obviously sufficient. Conversely, if v is not τ-reachable from u, then merging u and v will introduce a new interface transition sequence that contains an interface transition going to u (u in E) and an interface transition out of v (v in S). □

A direct checking of conditions in Case 3 for each pair of nodes is costly. For a node u, let

PS(u)= $\hbar_{y\in Epred(u)} Ssucc(y)$ and

SP(u)= $\hbar_{x\in Ssucc(u)} Epred(x)$.

Proposition 2. Two nodes $u \in$ E, $v\in$ S satisfy the Node Merging Condition if and only if Ssucc(u) = PS(v) if and only if Epred(v)=SP(u). □

From Proposition 2, Case 3 conditions in Lemma 2 can be checked with PS(·) instead (or equivalently with SP(·)) and if done properly, this reduces the overall cost to O(mn) as follows. Denote the interface equivalent minimization of a graph G by $MIN(G)$:

Algorithm 1. (Interface Graph Minimization)

```
Input:  An interface graph G, which is a τ-move DAG
Output: A minimized interface equivalent graph MIN(G)
1. compute topological order of  nodes: $v_1,...,v_n$
2. for i=n down to 1
3.     if $v_i \in S$ then Ssucc($v_i$):= {$v_i$}
4.                         else Ssucc($v_i$):=$\phi$ ;
5.    for each τ-edge ($v_i$, $v_j$), out of  $v_i$
6.         Ssucc($v_i$):= Ssucc($v_i$) $\cup$ Ssucc($v_j$);
7. for i=1 up to n
8.     if $v_i \in E$ then
9.         Epred($v_i$):= {$v_i$}, PS($v_i$):=Ssucc($v_i$);
10.                    else Epred($v_i$):=$\phi$ , PS($v_i$):=V;
11.      for each τ-edge ($v_j$, $v_i$), into  $v_i$
12.          Epred($v_i$):= Epred($v_i$) $\cup$ Epred($v_j$);
13.          PS($v_i$): = PS($v_i$) $\cap$ PS($v_j$);
14. radix sort and order the set Ssucc(u) for u in E,
           and order the sets Epred(v) and PS(v) for v in
S;
15. for each pair of nodes u and v
16.      if  (u,v ∈ S  ∧ Epred(u)=Epred(v)) ∨
17.       (u,v ∈ E ∧ Ssucc(u) = Ssucc(v)) ∨
18.       ((u ∈ E ∧ v ∈ S) ∧ (Ssucc(u) = PS(v)))
19.       merge nodes u and v;
20.  return minimized interface graph MIN(G)
```

Fig. 4. Algorithm 1: Interface Graph Minimization

Line 2-6 and 7-13 compute Ssucc(·) (Epred(·) and PS(·)) in a reverse (normal) topological order in time O(mn). We can represent each set as a list of nodes or as a characteristic vector, and use radix sorting to sort all the sets and order them lexicographically in time $O(n^2)$; at the end we can assign each set an integer (between 1 and 3n) so that equal sets receive the same integer. Checking identical sets for the three Cases of Lemma 2 in Line 16-19 takes a constant time for each pair of nodes. Since there are on the order of n^2 pairs of nodes to be checked, the total cost is $O(n^2)$.

An alternative (and generally more efficient) method for computing the equivalent nodes is the following. Scan the sorted list of the sets Ssucc(·), Epred(·) and PS(·), and partition the list into segments of equal sets (note that all equal sets are consecutive). For each segment, merge all E nodes u whose set Ssucc(u) is in the segment, merge with them any nodes v in S whose PS(v) set is in the segment; merge together all S nodes v whose Epred(v) set is in the segment. We have:

Theorem 2. Given an interface graph *G*, Algorithm 1 takes time O(*mn*) to construct a minimal interface equivalent graph where *m* and *n* are the number of edges and nodes in *G*, respectively. □

4.1 Minimal Interface Graph of Integrated Systems

For an analysis of integrated systems we want to construct a minimal interface graph *MIN*(*G*) of the Cartesian product *G* of all the components. Often we cannot afford to construct *G* due to state explosion. Indeed, there is no need to obtain *G* first. We can minimize each component first, take the Cartesian product of two components, minimize it and continue in this manner to obtain the minimal interface graph. Note that before minimization of an interface graph we first make it an interface equivalent ubiquitous graph using the procedure in Section 3.1. However, we need to justify first that if two nodes $u \equiv v$ in a component then they (all their duplicates) remain equivalent in the Cartesian product:

Theorem 3. If $u \equiv v$ in a component *A* (or *B*) then all their duplicates remain interface equivalent in the Cartesian product $A \otimes B$.

We need the following lemma, whose proof we omit:

Lemma 3. For a τ--path from node *P* to *Q* in $A \otimes B$, there is a τ -path of τ -moves in *A* (or *B*) only, from *P* to some node *W*, and a τ -path of τ -moves in *B* (or *A*) only, from W to Q . □

We are ready to prove the theorem.

Sketch of Proof. Assume $u \equiv v$ in *A*. In $A \otimes B$ node *u* (*v*) is duplicated to $(u,s_i), i = 1,...,n$ ($(v,s_i), i = 1,...,n$) where *n* is the number of nodes in *B*.To prove $(u,s_i) \equiv (v,s_i), i = 1,...,n$, we only need to show that any node $(x,s_j) \in$ Ssucc(u,s_i) is τ -reachable from any node (y,s_k) $\in$Epred(v,s_i) and that any node $(x,s_j) \in$ Ssucc(v,s_i) is τ -reachable from any node $(y,s_k) \in$Epred(u,s_i).

From lemma 3, there exist a node (v,s_k) so that there is a path of τ -moves (only in *A*) from (y,s_k) to (v,s_k) and a path of τ -moves (only in *B*) from (v,s_k) to (v,s_i) . Hence *y*∈ Epred(*v*) in *A* and s_i is τ -reachable from s_k in *B*.

Similarly, there exist a node (x,s_i) so that there is a path of τ -moves (only in *A*) from (u,s_i) to (x,s_i) and a path of τ -moves (only in *B*) from (x,s_i) to (x,s_j) . Hence *x*∈ Ssucc(*u*) in *A,* and s_j is τ -reachable from s_i in *B*.

Since *y*∈ Epred(*v*), *x*∈ Ssucc(*u*), and $u \equiv v$ in *A*, from Proposition 1, *x* is τ -reachable from *y* in *A* and, hence, there is a path of τ -moves from (y,s_k) to (x,s_k) . Since s_i is τ -reachable from s_k and s_j is τ -reachable from s_i in *B*, s_j is τ -reachable from s_k in *B* and, therefore, there is a path of τ moves from (x,s_k) to (x,s_j) . Hence, there is a path of τ moves from (y,s_k) to (x,s_k) and then to (x,s_j) , and (x,s_j) is τ-reachable from (y,s_k) .

Similarly, we can show that any node $(x,s_j) \in$ Ssucc(v,s_i) is τ -reachable from any node $(y,s_k) \in$Epred(u,s_i). □

From Theorem 3, we can do the interface equivalence minimization either before or after taking the Cartesian product. Denote this operation by *MIN*, we have:

Proposition 3. For the interface equivalent minimization, we have:

(1) $MIN[MIN(A)\otimes MIN(B)]\equiv MIN[A\otimes MIN(B)]\equiv MIN[(MIN(A)\otimes B]\equiv MIN(A\otimes B);$

(2) $MIN(A\otimes B)\equiv MIN(B\otimes A)$; and

(3) $MIN[A\otimes(B\otimes C)]\equiv MIN[(A\otimes B)\otimes C].$ □

From the above proposition, we have:

Corollary 4. Given an interface graph of an integrated system, which is a Cartesian product of more than one component, the interface minimization can be performed on individual components or Cartesian products of all or a subset of the components, and the resulting graphs are interface equivalent. □

5 Experiments on LMP of GMPLS

We report experimental results of the minimization algorithm on the Link Management Protocol (LMP) of GMPLS.

The IETF Standard GMPLS is a protocol suite that uses advanced network signaling and routing mechanisms to automatically set up end-to-end connections for all types of network traffic and provides a unified control plane and the necessary linkage between the IP and optical layers, allowing interoperable and scalable networks in both IP and optical domains. GMPLS protocol stack is composed of several protocols, including LMP, CR-LDP extension, RSVP-TE extension, and OSPF-TE extension. LMP is a protocol running between neighboring nodes and is used to manage TE links and verify reachability of the control channel. LMP consists of four major features: control plane management (CPM), link property correlation (LPC), link connectivity verification (LCV), and fault management (FM). Correspondingly, there are four main modules in each of the two communicating nodes. See Fig. 5.

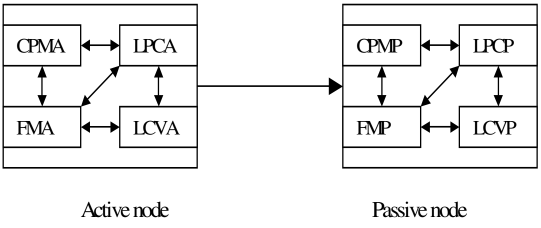


Fig. 5. LMP Modules

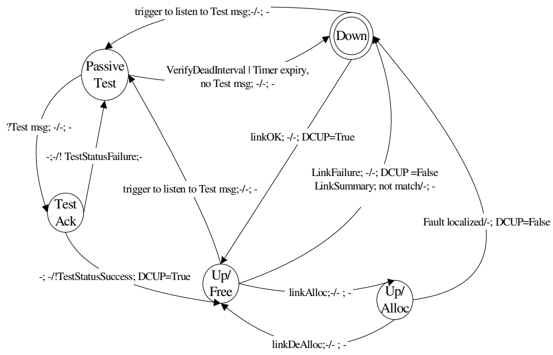

trigger to listen to Test msg;-/-; -
Down
Passive Test
VerifyDeadInterval | Timer expiry, no Test msg; -/-; -
?Test msg; -/-; -
-;-/! TestStatusFailure;-
linkOK; -/-; DCUP=True
LinkFailure; -/-; DCUP =False
LinkSummary; not match/-; -
trigger to listen to Test msg;-/-; -
Test Ack
Fault localized/-; DCUP=False
-; -/!TestStatusSuccess; DCUP=True
Up/ Free
linkAlloc;-/- ; -
Up/ Alloc
linkDeAlloc;-/- ; -

Fig. 6. LCV Passive EFSM

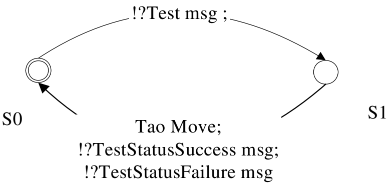


Fig. 7. *MIN*(*MIN*(LCVActive) $\otimes$ *MIN*(LCVPassive))

Each module is represented by an Extended Finite State Machine (EFSM), as is often done in protocols. An EFSM is an FSM extended with variables; transitions have besides input and output an associated *predicate* on the values of the variables, which is a condition (guard) on the occurrence of the transition, and has an *action* which is a transformation on the values of the variables. If all the variables have finite domains (eg. Boolean, finite counters) then an EFSM is simply a succinct representation of an ordinary FSM (see [7,13] for more details). As a first step we obtain from each EFSM the part of the corresponding FSM that is reachable from the initial state; this is called the *reachability graph*.

In Fig. 1 we showed the LCV module for the active node; the following figure shows the passive LCV module. In each transition label, the first two components show the input and the predicate and the latter two components show output and action.

There are 5 states and 11 transitions in each module and among them 3 are interface transitions in Active node and Passive node, respectively:

Active: -; -/!Test msg ; -
?TestStatusSuccess; -/-; -
?TestStatusFailure msg; -/-; -
Passive: ?Testmsg;-/-;-
-;-/! TestStatusSuccess;DCUP=True
-;-/! TestStatusFailure;-

The Cartesian product of the two reachability graphs contains 25 states and 103 transitions. Applying our minimization algorithm, we obtain $MIN(MIN(\text{LCVActive}) \otimes MIN(\text{LCVPassive}))$, which contains 2 states and 4 transitions. See Fig. 7.

We now consider the integration of the 6 modules (excluding FMA and FMP) of the two communicating LMP nodes. The Cartesian product of the reachability graphs of all the modules contains 177,000 states and 2,595,900 transitions; it is hard to handle for any available formal verification and testing techniques. Applying our minimization algorithm and taking advantage of the interface equivalence invariance of the order of minimizations on the components and products, we obtain an interface equivalent graph of 2,912 states and 24,987 transitions, which are manageable.

6 Applications

We discuss applications of our interface graph minimization technique to integrated system verification and testing. We are concerned with integrated system interoperability and want to analyze system behaviors, which are involved with interfaces among system components and ignore component local behaviors by changing them to τ-moves. We want to show that the minimized interface graph contains sufficient information for verification analysis and for constructing executable interoperability testing. We omit all the proofs.

6.1 Interface Livelock

As a simple case study, we discuss livelocks. More sophisticated properties, such as temporal properties, can be analyzed similarly.

Certain system states are specified as *progress-states* where system operation makes progress such as messages sent or received. For our study we only consider interface transitions, which are incident to progress states, i.e., integrated system makes progress before or after a system interface. We call such interface transition as *progress interface transition*. We are not concerned with system progress from components' internal behaviors. A non-progress interface cycle is a reachable cycle in the graph that contains at least one interface transition and yet does not contain any progress interface transitions. A non-progress interface cycle is called an *interface livelock*. When a communication system contains an interface livelock it can go through the cycle infinitely many times with infinitely many system interactions among the system components yet without making any progress.

We now show that to detect interface livelocks we only have to search a minimized interface graph and hence the search space is significantly reduced:

Theorem 4. Interface livelock is invariant with respect to interface equivalence minimization. □

Corollary 5. An integrated protocol system is interface livelock free if and only if its minimal interface equivalent graph is interface livelock free. □

From Corollary 5, the problem is reduced to checking interface livelocks of the minimal interface graph G^* of the integrated system. There is a variety of algorithms published on checking livelocks [2,7]. Yet they do not have interface transitions involved. We describe an algorithm that is applicable for interface livelock detection. It is a variant of algorithms for non-progress and accepting cycle detection [7].

Algorithm 2. Given an interface graph *G* with an initial node v_{init}, we construct an interface graph *G'* that is identical to *G* except that all the progress interface transitions are removed. Initially, all the nodes are not visited. We conduct a DFS in *G* from v_{init}. Whenever we visit a node *v* in *G* that is not visited we mark it *visited* and "jump" to *G'* to continue search from there. If we identify an SCC in *G'* with at least one interface transition, then we have found an interface livelock, since in the SCC we can construct a cycle with an interface transition yet there are no progress interface transitions in *G'*. Otherwise, we mark all the searched nodes in *G'* as *visited* along with the corresponding nodes in *G*, and return to node *v* in *G* to continue to search from there. The algorithm terminates either if it finds an interface livelock or all the nodes in *G* (and *G'*) are visited without identifying any interface livelock. In the latter case, *G* is interface livelock free. □

Obviously, the algorithm has a cost of DFS:

Proposition 4. Given an interface graph *G*, Algorithm 2 either finds an interface livelock or concludes that *G* is interface livelock free in time O(*m*) where *m* is the number of edges in *G*. □

Algorithm 2 determines whether an interface graph is interface livelock free. However, it does not identify all the possible livelocks if there are any. (Note that there may an exponential number of them.) This can be achieved by the following algorithm that is a modification of an algorithm in [5]:

Algorithm 3. Given an interface graph *G,* we have a tree walk from the initial node v_{init}. We continue from a current leaf node so long as no node is repeated along the tree path from v_{init}. We modify the algorithm by adding two indices at each node *v*, *I*(*v*) and *P*(*v*) where *I*(v) records the number of interface transitions and *P*(*v*) records the number of progress interface transitions from v_{init} to *v* along the tree path. Upon detecting a simple cycle while visiting a node *v*, i.e., there is an outgoing edge from *v* to *u*, which is a node on the tree path from v_{init} to *v*, we check whether the simple cycle from *u* along the tree path to *v* and then from *v* back to *u* is an interface livelock. It is an interface livelock if and only if there are no progress interface transitions and at least one interface transition, and this is the case if and only if: (1) $P(u)=P(v)$ and $v \rightarrow u$ is not a progress interface transition; and (2) $I(u) < I(v)$ or $\mathrm{I(u) = I(v)}$ but $v \rightarrow u$ is a non-progress interface transition. When we complete the tree walk we have checked all the simple cycles and identified all the simple interface livelocks if there are any. □

Proposition 5. Given an interface graph, all the simple interface livelocks can be obtained in time proportional to the size of a simple path (cycle) tree rooted at v_{init} . □

6.2 Interoperability Testing

Interoperability testing is to check the interoperations among integrated system implementations. Ideally, one might want to test on all possible interface transition sequences to reveal interoperation errors. However, the number of executable interface transition sequences could be infinite. This problem has been studied in [5,10] with different coverage criteria.

Suppose that we use a procedure for interoperability testing sequence generation and that we want to apply it to the minimized interface graph instead of the original graph, which is often impossible. In this case, the tests generated from minimized interface graph consist of interface transitions and τ-moves. We need to further process so that: (1) Each test is executable, i.e., it consists of a consecutive sequence of internal and interface transitions in the whole integrated system (the Cartesian product of all the original system components); (2) It contains the same interface transition sequence, i.e., they have the same projection to interface transition sequences; and (3) Without constructing the whole Cartesian product of all the original system components, i.e., we only need the minimized interface graph and the involved individual component information.

Suppose that we have a test sequence (path) ***p*** from a minimized interface graph G^*; it consists of interleaving interface transitions and τ-move sequences. We now discuss how to construct an executable test sequence according to the above three requirements. The basic idea is: we replace τ-move sequences between a pair of interface transitions by consecutive internal transitions, which can be obtained by examining the involved individual components only. From Lemma 3,

Proposition 6. Suppose that a τ-move sequence **τ**= $\tau_1\tau_2\hbar\ \tau_r$ in a reachability graph of a Cartesian product is from state $(s_1,...,s_k)$ to $(t_1,...,t_k)$ where s_i and t_i are states in component $G_i, i=1,...,k$. Then state t_i is reachable from s_i in G_i, i.e., there is a path of τ-moves ω_i and hence internal transition sequence z_i in G_i from s_i to t_i, *i=1,…,k*. Consequently, there is an internal transition sequence $z_1z_2\hbar\ z_k$ in the Cartesian product from state $(s_1,...,s_k)$ to $(t_1,...,t_k)$. □

Note that there is no need to construct the Cartesian product graph; we only need a minimized interface graph and a graph of each involved component. Furthermore, there is no need to construct the connecting τ-move sequences a_i ; we only need to find an internal transition sequence z_i in G_i from s_i to t_i , which can be easily constructed by a BFS in G_i, i=1,…,*k*. We summarize:

Algorithm 4. (Interoperability Test Sequence Generation)

```
input: Integrated system $G=\otimes_{i=1}^{k} G_i$ with initial node $v_{init}$.
output: A set $\Gamma$ of executable test sequences in G with a desired fault coverage
construct a minimized interface graph $G^*$ from G;
construct a set P of paths in $G^*$ from $v_{init}$
    with a desired fault coverage;
$\Gamma = \phi$;
for each path p in P
    construct an executable test sequence
        z from $v_{init}$ in G;
    $\Gamma = \Gamma \cup$ z;
return $\Gamma$                                        □
```

As an experiment, we use the interoperability test sequence generation software tool, called *ITIS*, in [5] with Basic coverage and apply it to LMP/GMPLS: (1) Construct minimized interface graph G^* ; (2) Generate Basic Coverage tests using *ITIS*; (3) Convert each test to an executable one in LMP/GMPLS.

A simple example is the communicating LCV modules, see Fig. 7. There are only two nodes and each node represents a module, LCVA (Active) and LCVP (Passive), respectively. From this minimized interface graph, 3 test sequences are generated with Basic Coverage: (1) *!Test message*, τ-move; (2) *!Test message, !TestStatusFailure*; (3) *!Test message, !TestStatusSuccess*. Using Algorithm 4, the 3 corresponding executable interoperability testing sequences are generated, involving both LCVA and LCVP:

(1) **LCVA**: I/O: Trigger to send Test msg /, I/O: */!Test message*
LCVP: I/O: linkOK/, I/O: Trigger to listen to Test msg /, I/O: *?Test message /*

(2) **LCVA**: I/O: Trigger to send Test msg /, I/O: */!Test message*, I/O: *?TestStatusFailure/*
LCVP: I/O: linkOK/, I/O: Trigger to listen to Test msg /, I/O: *?Test message /*, I/O: */!TestStatusFailure*

(3) **LCVA**: I/O: Trigger to send Test msg /, I/O: */!Test message*, I/O: *?TestStatusSuccess/*
LCVP: I/O: linkOK/, I/O: Trigger to listen to Test msg /, I/O: *?Test message /*, I/O: */!TestStatusSuccess*

7 Conclusion

For a study of integrated protocol system interface and interoperability, we investigate interface graphs and their minimization, identify a new state equivalence relation suitable for this purpose, and develop and implement an efficient algorithm for it. The technique is applied to the GMPLS protocol and we also discuss how it can be used for verification and interoperability testing. A similar method can be used more generally if we want to focus on a part of the system or on a particular feature that involves a selected subset of transitions (not necessarily for interfaces); a minimum equivalent system can be computed efficiently, which contains these transitions and preserves exactly all the involved traces.

Acknowledgements

We thank Xiao-tian Yin and Hui Jian for the comments and help with part of the experiments.

References

[1] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, *Minimal model generation*, Proc. CAV, 197-203, 1990.

[2] E. M. Clarke, O. Grumberg and D. A. Peled, *Model Checking*, MIT Press, 1999.

[3] R.J. van Glabbeek, *The Linear Time – Branching Time Spectrum I*, in *Handbook of Process Algebra,* Begstra, Ponse, Smolka eds., Elsevier, 3-99, 2001.

[4] S. Graf, B. Steffen, G. Luttgen, *Compositional minimization of finite state systems using interface specifications,* Formal Aspects of Computing, 1996.

[5] R. Hao, D. Lee, R. Sinha and N. Griffeth ,*Integrated System Interoperability Testing with Applications to VoIP*, IEEE/ACM Trans. on Networking, Oct. 2004. An early version appeared in FORTE/PSTV 2000.

[6] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[7] G. J. Holzmann, *Design and Validation of Computer Protocols*, Prntice Hall, 1991.

[8] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Reading, MA: Addison-Wesley, 1979.

[9] P. Kanellakis and S. Smolka, *CCS Expressions, Finite State Processes and Three Problems of Equivalence*, Information and Computation, Vol. 86, 1983, pp. 43-68.

[10] S. Kang and M. Kim, *Test Sequence Generation for Adaptive Interoperability Testing*, in Proc. Protocol Testing Systems VIII, 1995, 187-200.

[11] P. V. Koppol, R. H. Carver, and K.-C. Tai, *Incremental Integration Testing of Concurrent Programs*, IEEE Trans. on Software Eng., 28(6), 607-623, 2002.

[12] D. Lee and M. Yannakakis, *Online minimization of transitions systems,* Proc. ACM STOC, 264-274, 1992.

[13] D. Lee and M. Yannakakis, *Principles and Methods of Testing Finite State Machines - A Survey,* Proceedings of IEEE, Vol. 84, No. 8, 1090-1123, 1996.

[14] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

DART: Distributed Automated Regression Testing for Large-Scale Network Applications

Brent N. Chun

Intel Research Berkeley, Berkeley, CA, USA

Abstract. This paper presents DART, a framework for distributed automated regression testing of large-scale network applications. DART provides programmers writing distributed applications with a set of primitives for writing distributed tests and a runtime that executes distributed tests in a fast and efficient manner over a network of nodes. It provides a programming environment, scripted execution of multi-node commands, fault injection, and performance anomaly injection. We have implemented a prototype implementation of DART that implements a useful subset of the DART architecture and is targeted at the Emulab network emulation environment. Our prototype is functional, fast, and is currently being used to test the correctness, robustness, and performance of PIER, a distributed relational query processor.

1 Introduction

Recently, we have seen the emergence of a number of novel wide-area applications and network services. Examples include distributed hash tables (DHTs) [24, 19, 21, 18, 31], wide-area storage and archive systems [11, 12, 5], distributed query processors [10, 29], content distribution networks [14, 8], robust name services [17], and routing overlays [1, 25]. These distributed applications provide diverse functionality to end users, but nevertheless have one common goal: to deliver correct behavior and high performance in the presence of high concurrency, node and network failures, and transient and persistent performance anomalies. Designing and implementing applications with these characteristics presents significant technical challenges.

With sequential (i.e., single-node) applications, unit testing [4] is an effective and widely used mechanism for building correct, robust, and maintainable software. In unit testing, users write tests that exercise and verify the functionality of specific parts of an application. Over time, users build up a collection of such tests, each covering an increasing fraction of the application's overall functionality. A testing framework automates the execution of unit tests and is applied whenever the application is modified. The end result is that code changes can be automatically verified to have not broken existing functionality (as covered by the unit tests), thereby leading to increased confidence when performing significant modifications to existing code. Building on these ideas, the motivation of this work is to develop an analogous set of automated testing mechanisms with associated benefits for large-scale network applications.

Designing appropriate mechanisms for automated testing of distributed applications presents several challenges. First, such mechanisms need to be fast and scalable to

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 20–36, 2005.

enable large-scale testing and performance analysis. This, in turn, will enable programmers developing distributed applications to obtain rapid feedback on the implications of incremental design and implementation choices. Second, such mechanisms should be flexible to allow applications to be tested along multiple dimensions including correctness, robustness (e.g., in the presence of faults), and performance. Finally, these mechanisms should enable testing under a wide range of operating conditions in terms of network delays, bandwidth, and packet loss in addition to node and network faults and performance anomalies.

To address these challenges, we have designed DART, a framework for distributed automated regression testing. DART provides users with a programming environment and a set of primitives which can be used to construct a wide variety of distributed tests. Building on a set of scalable cluster tools, DART also provides a runtime that enables efficient execution of such distributed tests at scale. DART targets cluster-based network emulation environments such as Emulab [30] and ModelNet [26] to enable testing under a wide range of network operating conditions. Such environments typically provide two networks: an emulated network to emulate wide-area network delays, bandwidth, and packet loss and a separate, non-emulated control network (e.g., 100 Mbps or Gigabit Ethernet). It is the latter network that DART uses to efficiently and reliably control the execution of distributed tests.

We have implemented a prototype of DART that is targeted to the Emulab [30] network emulation environment. The system implements a core subset of our design which provides enough functionality that we have found it to be useful in practice. In particular, we have and continue to use DART to test and benchmark PIER [10], a distributed relational query processor that runs over a DHT. This paper describes the motivation, design, implementation, and performance analysis of DART and is organized as follows. In Sect. 2, we motivate the need for automated large-scale testing for distributed applications. In Sect. 3, we present DART's system architecture. In Sect. 4, we describe a prototype implementation of DART targeted for Emulab. In Sect. 5, we measure the performance of our DART implementation for core primitives, a baseline distributed application, and PIER. In Sect. 6, we present related work and in Sect. 7, we conclude the paper.

2 Large-Scale Distributed Testing

With single-node applications, unit testing frameworks provide two key components to the programmer: a set of commonly used mechanisms for writing tests and a runtime that automates test execution. Common mechanisms in unit testing frameworks include templates for setting up and tearing down unit tests, functions for verifying that actual outputs match expected outputs, and functions for communicating test outcomes back to the user. Using these mechanisms, programmers write tests that verify the functionality of specific parts of their application. Depending on the test, verification might include verifying that actual outputs match expected outputs, that bad / corner case inputs are handled correctly, that an application meets expected target performance metrics, and so forth.

A key benefit of these unit testing frameworks is that they *lower the barrier* to verifying correctness, robustness, and performance in an application's implementation. By providing a common set of mechanisms to write tests and a runtime to execute tests, unit testing frameworks make developing, maintaining, and applying unit tests less cumbersome and less error prone by factoring out a common set of machinery and by automating the test execution process. When the barrier to running tests is low, programmers employ them more often and subsequently reap the benefits of verifying that what worked before continues to work even after significant code changes.

While unit testing is pervasive in the world of single-node applications, there has been little work on providing an analogous set of mechanisms for large-scale distributed applications. We believe that providing such mechanisms will be a key enabler towards rapidly building distributed applications that are correct, robust, and deliver high performance under a wide range of operational environments. Providing such mechanisms requires factoring out and implementing commonly used mechanisms for distributed testing and implementing a runtime layer that executes these mechanisms in a fast and efficient manner. Ensuring that the testing infrastructure is itself fast and robust is key since rapid, correct feedback to the programmer usually implies that the programmer will use the system more often when developing.

3 Architecture

This section describes the DART system architecture. As mentioned, the goal of a DART system is to support automated testing of large-scale distributed applications. For a given distributed application, a user may wish to perform a variety of tests that test the application's correctness, robustness, and performance under a range of operating environments. DART supports automated execution of a suite of such distributed tests, where each test involves: (i) setting up (or reusing) a network of nodes to test the application on, (ii) setting up the test by distributing code and data to all nodes, (iii)

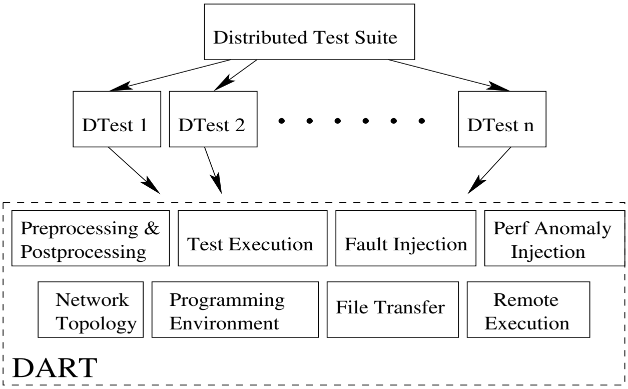


Fig. 1. DART architecture. Each distributed application has a suite of distributed tests. Each test is instantiated and executed using DART

executing and controlling the distributed test, and finally (iv) collecting the results of the test from all nodes and evaluating them. To support this automation, DART relies on a number of components (Fig. 1) which are described further in this section.

3.1 Network Topology

The first step in executing a DART test is setting up a network of nodes to test the application on. In emulated network environments, such networks are constructed using a set of cluster machines with emulated inter-node network delays, bandwidth, and packet loss. In Emulab [30], for example, users set up experiments consisting of network topologies which specify end hosts, routers, and network links with varying delay, bandwidth, and loss characteristics. Each experiment is then physically instantiated using a set of cluster nodes, a per-experiment VLAN, and wide-area network emulation using DummyNet [20]. ModelNet [26], another emulation environment, provides similar functionality. In addition, its adds per-hop delay, bandwidth, and loss emulation as well as distillation of large network topologies which enables trade-offs between scalability and emulation accuracy to be made (e.g., when using large network topologies [7]).

Given a target environment, a DART implementation provides two ways for a user to specify network topologies. First, DART provides a set of parameterizable network topologies (routers and end hosts), each of which maps down to a description in an underlying network topology language (e.g., Emulab ns-2 files). Second, DART supports raw network topologies as expressed in the target platform's network topology language. In DART, parameterizable topologies are provided mainly as a convenience. Such topologies might include topologies representative of real networks, topologies which might be easy or hard for different classes of applications, and/or topologies that reflect realistic end host heterogeneity in terms of last-hop bandwidth, latency, and host availability [22]. In many cases, we anticipate parameterizable topologies will provide a sufficiently broad range of environments to test and characterize the behavior of a distributed application before moving towards real wide-area network environments (e.g., PlanetLab [15], RON [2], etc.) where additional noise can make it difficult to ascertain whether observed problems are due to the application or due to the infrastructure and the real world.

3.2 Remote Execution and File Transfer

The second step in executing a DART test is setting up the test by distributing code and data to all nodes. Efficiently setting up and subsequently (Sect. 3.4) executing distributed tests in DART relies heavily on two key components of the DART runtime: multi-node remote execution and multi-node file transfer. In DART, there are a number of cases where multi-node remote execution is needed. For example, in testing a peer-to-peer application, multi-node remote execution might be used to start the application up on all nodes in the system and, some time later, to start a set of clients who issue requests. Before such a test can even run, code and data will also need to be distributed to all nodes, and this further requires having the ability to perform multi-node file transfers. Remote execution needs to be efficient because nodes might be controlled in various ways throughout a test (e.g., starting up servers, starting up clients, creating and controlling adversaries, etc.). File transfer needs to be efficient because code and data may be large and distributing such data to multiple nodes in a large scale test will be costly if it is read from, say, a centralized NFS file server. Consequently, a DART implementation needs to provide fast remote execution and file transfer primitives if the system aims to scale up to large system sizes.

3.3 Scripting and Programming Environment

To facilitate writing distributed tests, DART provides scripting to specify high-level details of test execution and a minimal programming environment which provides low-level details for writing actual distributed test code that runs on the system. Each test in DART has both an XML test script and test code and data. The test script specifies a unique test name, a unique topology name (to enable topology reuse), a network topology (e.g., an Emulab ns-2 file), test code and data, a test duration, a preprocessing script, a set of scripted commands, a set of scripted faults, a set of scripted performance anomalies, and a postprocessing script. Test scripts are interpreted by DART and associated actions are executed using the DART runtime. For example, a script for a distributed storage system might specify code and data for the storage system, start a set of storage servers on all nodes, start a client that writes and reads specific data, and verify consistency of the results in a postprocessing script.

DART provides a minimal programming environment to facilitate the writing of distributed test code. When executing DART tests, one node is designated as the master while all remaining nodes are designated as slaves. The DART runtime uses the master as the point of control for executing and coordinating the entire test. Similar to GLUnix [16], any scripted command executed on any node through DART is provided with the following environment variables:

- DART_TEST: unique test name.
- DART_NODES: space-delimited list of node IP addresses on the emulated network.
- DART_NUM_NODES: number of nodes in the DART test.
- DART_MY_VNN: node number from 0 to DART_NUM_NODES - 1.
- DART_MASTER: master's emulated IP address.
- DART_GEXEC_MASTER: master's control IP address.
- DART_MY_IP: this node's emulated IP address.
- DART_GPID: globally unique identifier for this particular test instance.
- DART_COMMON_DIR: directory for code and data common to all nodes.
- DART_MY_INPUT_DIR: input directory for per-node code and data.
- DART_MY_OUTPUT_DIR: output directory for per-node code and data (e.g., for writing test output, logfiles, etc).
- DART_ALL_OUTPUT_DIR: aggregated output directory of all DART_MY_OUTPUT _DIR directories. This directory is populated during a collect phase at the end of a test.

Using these environment variables facilitates writing distributed tests using DART. For example, consider testing the correctness of query evaluation in PIER. Such a test needs to instantiate a PIER process on every node and it needs to instantiate clients on a subset of nodes, each of which will issue queries to the system and save the results for verification. Starting PIER up on a node minimally requires at least one piece of information: the IP address of a landmark node to bootstrap all nodes into the DHT. Using the above environment, one obvious possibility for this is to simply use the DART master (`DART_MASTER`). Each PIER process will also want to save relevant output for potential debugging (e.g., `stderr` in case an exception occurs) and PIER clients will need to save query results for postprocessing to verify query evaluation correctness. Using the above environment, capturing program output would be done by simply writing files to `DART_MY_OUTPUT_DIR`. When the test completes, DART collects output from all `DART_MY_OUTPUT_DIR` directories on all nodes and places them in `DART_ALL_OUTPUT_DIR` on the master where the results of the test are then computed (e.g., checking actual output against known, correct output).

3.4 Preprocessing, Execution, and Postprocessing

The third and fourth steps of executing a DART test are executing and controlling the distributed test and, lastly, collecting the results of the test from all nodes and evaluating them. Each distributed test in DART goes through preprocessing, execution, and postprocessing phases to compute the results of the test. Each of these phases is scripted by the user using the primitives provided by DART. Given a network of nodes (e.g., an experiment on Emulab) and code and data that has been distributed to those nodes, preprocessing is the first stage and entails executing whatever commands that are necessary before actually running the test. For example, if software packages (e.g., RPMs or tarfiles) were distributed as part of the code and data distribution phase, then preprocessing would be the place where one-time installations of this software would take place. We separate preprocessing from the actual execution of the test since, for a given application, we expect it will be frequently be the case that an application performs the same preprocessing in each of a series of tests (e.g., installing the same set of RPMs, such as the Java JDK in PIER's case).

Once preprocessing is complete, DART then proceeds to the execution phase where execution and control of the distributed test is performed to completion. This phase primarily entails scheduling and executing user-specified, scripted commands on specific subsets of nodes at specific points in time (e.g., starting a set of servers up, starting a set of clients, etc.). Further, depending on the test, it might also involve injecting faults and performance anomalies in certain parts of the system at certain points in time. A churn test for a peer-to-peer application, for example, might involve first starting the application on all nodes in the system, letting the system stabilize for several minutes, then injecting a sequence of node join (scheduled command) and leave (scheduled process or node fault) events into the system and measuring the system's behavior over time (e.g., the success or failure of routing requests in the case of structured peer-to-peer overlays).

Finally, once the distributed test has finished executing, a postprocessing stage is performed to collect all the output from all the nodes and to apply a user-specified postprocessing test to process the test's output and verify its goodness. The definition of goodness will be specific to the application and the type of test being performed. For example, a correctness test might verify that actual replies to client requests match the correct, expected values (which would be computed offline a priori). A robustness test might verify that after killing some subset of nodes that the system continues to function as expected (e.g., suppose it was designed to be k-fault tolerant). Finally, a performance test might compute the overall performance numbers from all nodes and verify that these performance numbers lie within some expected bounds. Each test produces output, which may optionally be sent back to the user's machine (e.g., performance numbers) and returns a or a depending on whether the test succeeded or failed (as defined by the user).

3.5 Fault Injection

To understand how a distributed application behaves in the presence of node and network faults, DART also provides fault injection primitives which may be specified by the programmer when scripting a distributed test. Which primitives are supported in a particular implementation will depend on the capabilities of the underlying platform. In the best case, node, process, and network failures are all supported and can be scripted to execute at specific times on specific parts of the system (e.g., a specific subset of nodes):

- **Node failures:** specifies hard failures of specific subsets of nodes over specific periods in time. In Emulab, such failures can be scripted using underlying support from Emulab's event system.
- **Process failures:** specifies the hard failure of specific processes (e.g., by name, by `uid`, etc.) on a given node. In contrast to node failures, the node continues to operate properly.
- **Network failures:** specifies the failure of specific parts of the network at specific points in time. As with node failures, network failures can also be scripted through support from Emulab's event system (e.g., to turn a network link off at a specific time).

3.6 Performance Anomaly Injection

In addition to hard node and network faults, another important class of failures of interest are performance failures [3]. For example, consider the case where a 1.5 Mbps network link does not fail completely but its effective bandwidth drops to 0.001 Mbps. While technically the link has not failed in the sense that it fails to route packets, the performance impact of such a performance degradation is likely to have significant implications for application performance. Understanding how applications behave in the presence of such performance faults is an important step towards building robust distributed applications. Towards this end, DART provides a set of primitives to introduce performance anomalies into the system. Similar to hard failures, the types of scripted performance anomalies supported by DART include:

- **Node and process performance anomalies:** decreased or varying CPU, memory, network, and I/O performance. Such anomalies might be introduced by using sufficient powerful schedulers [28, 9, 6, 23] in combination with support from the underlying emulation environment.

– **Link performance anomalies:** increased delay, decreased bandwidth, and increased packet loss in specific parts in the network. Such anomalies might be introduced using support provided by the underlying target platform (e.g., using Emulab's event system to dynamically change link delays, bandwidth, and packet loss).

4 Implementation

We have implemented a DART prototype targeted to the Emulab network emulation environment. Our prototype is implemented using a combination of C and Python and supports a subset of the architecture described in Sect. 3. Parameterizable network topologies, efficient multi-node remote execution and file transfer, a scripting and programming environment, and preprocessing, execution, and postprocessing of arbitrary scripted commands at specific times on subsets of nodes are all supported. Our prototype is functional, efficient, and is currently being used on a routine basis for testing, debugging, and benchmarking PIER.

4.1 GEXEC and PCP

As mentioned, multi-node remote execution and file transfer are key primitives that are used heavily throughout DART and hence need to be fast and efficient. To address this need, we have designed and implemented GEXEC, a fast multi-node remote execution system, and PCP, a fast, multi-node file transfer utility. Both systems rely on a hierarchical design based on a k-ary tree of TCP sockets over a specific set of nodes (e.g, nodes specified using the `GEXEC_SVRS` environment variable for GEXEC). Such trees are built on every invocation of either the `gexec` or `pcp` command using a k tree building step which involves routing tree create messages down to leaf nodes and routing tree create acknowledgments back to the root. We use a tree-based approach primarily for parallelism and to utilize aggregate resources across all nodes.

GEXEC provides multi-node remote execution of arbitrary commands by routing commands down the tree to all nodes. For all commands, GEXEC supports transparent forwarding of Unix signals, `stdin`, `stdout`, and `stderr` to allow control of remote processes and also obtain remote output. Control and data are all transferred over the tree, down in the case of signals and `stdin` and up in case of `stdout` and `stderr`. Two remote execution models are supported: default and detached. In default mode, the failure model is that if any node fails during the execution, GEXEC aborts on all nodes. In contrast, in detached mode, GEXEC simply builds the tree, starts the command on all nodes, and exits. Both modes are used in DART (e.g., default mode for executing bootstrapping commands, detached mode for running the application being tested, which might crash).

PCP provides fast multi-node file transfer by routing files down the tree in an incremental fashion in 32 KB chunks. Starting with the root, chunks are sent to each node's children. As each chunk is received, each node writes the chunk to local disk, then forwards the chunk off to each of its children. Because files are transferred using a k-ary tree and transferred in chunks (which incur small store-and-forward delays as compared to sending the entire file at once), PCP provides both parallelism and pipelined execution that leads to very high aggregate bandwidth usage. Generally, the optimal choices for tree fanout and message size will depend on node network bandwidth, the network's configuration, and disk write bandwidth. As we show in the next section, using a fanout of and 32 KB messages delivers high performance on Emulab and thereby makes multi-node file transfer a highly efficient primitive in our DART prototype.

4.2 Master and Slaves

Our DART prototype targets the Emulab network emulation environment and uses GEXEC and PCP as the basis for fast distributed test execution (Fig. 2). In our implementation, tests are remotely instantiated and controlled using two machines: `users.emulab.net` and a master node arbitrarily chosen from the set of nodes in the test's network topology. We use `users.emulab.net` to manage network topologies for DART (e.g., creating and destroying experiments). Each node in an Emulab experiment is assigned one or more emulated IP addresses and one control IP address. We use `users.emulab.net` to obtain information about the network configuration of each Emulab experiment. This information is subsequently used to control distributed test execution by running GEXEC and PCP over the fast, control network.

Each Emulab experiment created using DART is bootstrapped with a few common features that are required for DART to operate properly. First, each node is bootstrapped with a small set of core software including GEXEC, PCP, and `authd`, an authentication service used by both GEXEC and PCP. Second, each node is configured to boot the RedHat 7.3 Linux distribution which uses the Linux 2.4.18 kernel. The common software set is required since this software forms of the basis of the DART runtime. The use of Linux on the nodes is needed primarily because the versions of GEXEC and PCP currently used in DART do not run on FreeBSD, the other node operating system available on Emulab.

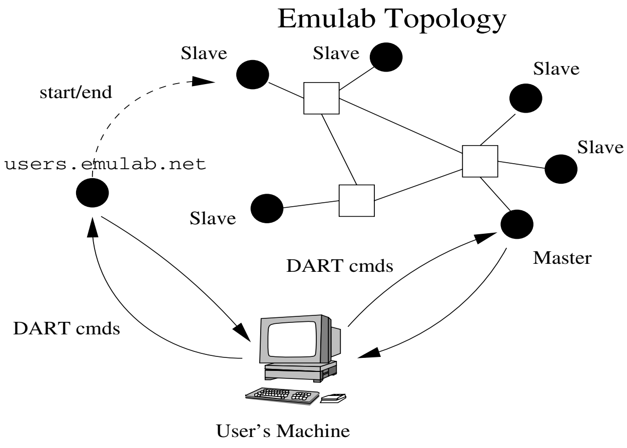


Fig. 2. DART implementation on Emulab

Once an Emulab topology is instantiated, all subsequent control is done through the master which essentially serves as a proxy for executing distributed tests in DART. Among the master's tasks are: distributing code and data to all nodes, providing the programming environment for distributed tests, and performing preprocessing, execution, and postprocessing of tests across all nodes. In our current implementation, we use `ssh` to securely execute commands on the master and use GEXEC to execute commands and PCP to transfer code and data to other nodes in the system. For example, to reset an experiment such that it can be reused, we use `ssh` to send a reset command to the master and use GEXEC, invoked from the master, to quickly reset all nodes in the network by remotely removing old files and killing old processes from the previous test.

5 Evaluation

In this section, we analyze the performance of our DART implementation. We begin by measuring the performance of two key primitives: multi-node remote execution and multi-node file transfer. As described in Sect. 4, these primitives are implemented by GEXEC and PCP, respectively, and are used extensively in our DART prototype. Next, we analyze the overall performance of performing DART tests for both a baseline distributed application and PIER, a distributed relational query processor. All experiments were performed on Emulab. The first set of experiments were performed on 64 Pentium III nodes: 18 of which were 600 MHz nodes with 256 MB of memory, 46 of which were 850 MHz nodes with 512 MB of memory. The second set of experiments were performed on 32 Pentium III nodes: 10 of which were 600 MHz nodes and 22 of which were of the 850 MHz variety. All nodes in both cases ran the Linux 2.4.18 kernel and were connected via 100 Mbps Ethernet.

5.1 Performance of DART Primitives

The first set of measurements characterizes the performance of multi-node remote execution and multi-node file transfer using GEXEC and PCP. Figure 3 depicts remote execution performance on multiple nodes using GEXEC. Each curve corresponds to GEXEC's performance using a different tree fanout. Recall that GEXEC performs multi-node remote execution by first building a k-ary tree where k is the fanout at each non-leaf node and using this tree to control remote execution. Each point on each curve represents the remote execution time (milliseconds) to execute a simple command (`/bin/date`) on nodes ( ). Each point on each curve is the average of 30 different runs on a subset of Emulab nodes. Overall, we observe that remote execution using GEXEC is fast (typically about 100 ms) and that remote execution times do not appreciate much as we scale the system size up. This, in turn, implies fast and efficient control of distributed tests in DART using GEXEC.

Next, we perform a similar experiment to measure the performance of multi-node file transfer using PCP. Similar to GEXEC, PCP also builds a k-ary tree and uses this tree to perform parallelized, pipelined file transfer. Figure 4 shows the aggregate bandwidth delivered when distributing a 34.7 MB file (the Java 1.4.2_03 JDK RPM) to nodes ( ) using PCP and using 32 KB messages. Each curve cor-

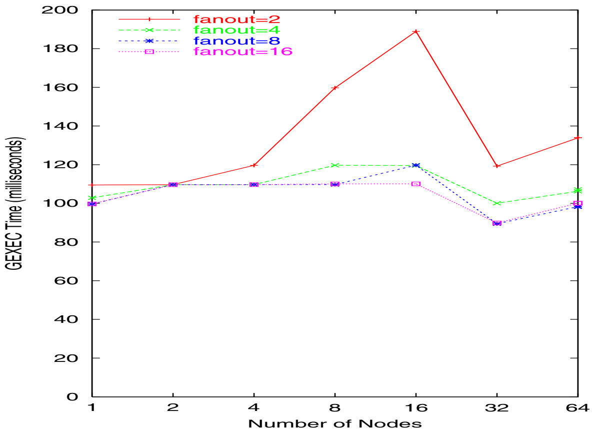


Fig. 3. GEXEC performance on Emulab. Each curve corresponds to a different tree fanout, while each point represents the remote execution time (milliseconds) to execute a simple command (`/bin/date`) on n nodes ($n = 1, 2, 4, \ldots, 64$)

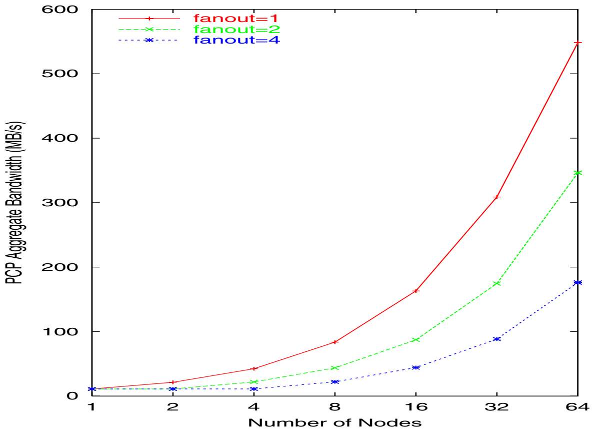


Fig. 4. PCP performance on Emulab. Each curve corresponds to a different tree fanout, while each point represents the aggregate bandwidth delivered when distributing a 34.7 MB file (the Java 1.4.2_03 JDK RPM) to n nodes ($n = 1, 2, 4, \ldots, 64$) using PCP

responds to a different tree fanout and each point on each curve is the average of 20 different runs. Using a tree fanout of 1 (i.e., a chain), we observe that PCP is able to deliver an average of 548 MB/s of aggregate bandwidth when distributing a 34.7 MB file to 64 nodes. Larger tree fanouts do not help in the case of Emulab since each node is connected by 100 Mbps Ethernet (i.e., 12.5 MB/s of peak bandwidth) and each node can write to disk at least that fast. Hence, our DART prototype uses PCP's default fanout of 1 which, as shown, delivers high performance and enables data to be moved around efficiently when conducting large-scale DART tests.

5.2 Overall DART Performance

The second set of measurements quantify the overall performance of performing DART tests for both a baseline distributed application and a real distributed application (PIER). The baseline distributed application is the null distributed application. It's an application that runs on 32 nodes but does not perform any computation. The test returns immediately and thus the times associated with this test are, in the current implementation, a lower bound on the total time to execute a distributed test in DART. PIER, as mentioned, is a distributed relational query processor that runs over a DHT. We use DART to routinely perform a number of tests on PIER. In this instance, we present performance results when testing the correctness of a distributed selection query on 32 nodes using different query plans (e.g., different packet sizes). (The test queries static per-node data and hence we know what the correct query result ought to be.) The time to perform this particular test once the test has been set up on all nodes is 700 seconds. The goal of these measurements is to show that the overhead of performing DART tests is small relative to distributed test times, which we anticipate will involve running a test for at least several minutes (e.g., as in the PIER selection query test) in most cases.

Each test involves four potential components. First, there is the time to set up the network topology for the test (*esetup*). This involves the time to securely transfer an Emulab network topology file to `users.emulab.net` and to instantiate the Emulab experiment. Second, there is the time to set up a particular distributed test (*dsetup*). The main cost here is transferring code and data to the master node in the Emulab experiment and distributing code and data to the slaves. Third, there is the time to perform preprocessing, execute and control the distributed test, collect the results on the master, and perform postprocessing (*drun*). Fourth, there is the cost of reseting the test environment on all nodes (*dreset*)). This involves clearing out results from the previous test and killing all processes associated with the previous test. Note that a test may reuse a network topology from a previous experiment if that test uses the same topology (e.g., the same 32-node topology in our measurements). When running a test for the first time on a network topology, no *dreset* cost is incurred since the system is clean, whereas when reusing a topology for a different test, the *dreset* cost must be paid.

Table 1 shows the overall times (seconds) to run distributed tests on 32 Emulab nodes using DART for a baseline null application and a 700 second correctness test in PIER for a distributed selection query. For both the baseline and for PIER, we present

Table 1. Breakdown of overall times (seconds) to run distributed tests on 32 Emulab nodes using DART for a baseline null application and a 700 second correctness test in PIER for a distributed selection query

	Base	Base reuse	PIER	PIER reuse
esetup	202.3	–	206.3	–
dsetup	16.2	16.0	52.6	46.2
drun	28.8	29.2	758.7	735.7
dreset	–	4.2	–	4.0
Total	247.3	49.4	1017.6	785.9

results when a new Emulab experiment is instantiated and when an existing Emulab experiment is reused (the reuse columns), the latter case requiring an additional reset component to prepare for a new test.

We observe the largest baseline cost to be *esetup*, the time to instantiate a new 32-node Emulab experiment. Measurements on Emulab revealed this time to be, on average, 204.3 seconds which is consistent with previous measurements [30]. The relatively high cost of creating a new Emulab experiment suggests reusing existing Emulab experiments when conducting tests on the same network topology. As mentioned, reusing a topology requires an additional reset phase to clear old files and kill old processes. Our measurements indicate that these costs are, on average, 4.1 seconds which is relatively low. Still, this number is relatively high compared to GEXEC remote execution times. (We use GEXEC to clear old files and kill old processes from the master.) This is largely due to our use of a new `ssh` connection each time we communicate with either `users.emulab.net` or the master. This overhead is also a significant component in the other baseline costs as well, namely *dsetup* and `drun` which on average were 16.1 seconds and 29.0 seconds respectively. When reusing the network topology, the total baseline cost to execute a null distributed test on 32 nodes was 49.4 seconds.

Turning to PIER, the key numbers of interest are the *dsetup* and *drun* times. We measured the average *dsetup* time for PIER to be 49.4 seconds, while for the baseline, the average *dsetup* cost was 16.1 seconds. The main difference between the two is the additional cost associated with transferring code and data to the master and from the master to all slaves. In the PIER case, code and data transferred from the user's desktop to the master was 3.32 MB in size (four different directories), while code and data transferred from Emulab's NFS fileserver to the master totaled 37.0 MB, the size of the Java 1.4.2_03 JDK and the static data being queried. As shown in Fig. 4, transferring data from the master to all slaves using PCP is efficient. However, as with the baseline, liberal use of new `ssh` connections again incur significant overhead. In the current implementation, each directory being transferred causes a new `ssh` connection to be created to the master, each of which usually takes approximately 2-3 seconds. We intend to optimize this by establishing a single secure connection with the master and reusing it in the future. This should reduce the gap between the baseline and PIER by approximately 12-18 seconds.

Despite the overhead of multiple `ssh` connections to the master, we see that the overhead of using DART to perform distributed tests of PIER is still quite reasonable relative to the typical time to perform a meaningful test. In this case, the selection query correctness test needs to run for 700 seconds. This includes a 120 second delay to allow the DHT to stabilize and for PIER to build up a multicast tree to perform query dissemination to all nodes. It also includes the time to perform a selection query in four different ways, in each case allowing the query to run for 120 seconds and leaving 10 seconds in between each query to avoid query interference. Finally, a minute is alloted before finally shutting down the test, which leads to a test time of 700 seconds. Relative to the total time, the DART overhead in this case is 11.3% (i.e., 85.9 seconds out of 785.9 seconds) which we believe is quite reasonable given the `ssh` performance improvements we intend to make and the fact that distributed testing using DART is entirely automated and does not require any human intervention.

6 Related Work

There have been relatively few efforts aimed at building frameworks for large-scale testing of distributed applications. In this relatively small space, the closest related project is TestZilla [27]. Like DART, TestZilla provides a framework for testing distributed applications and leverages a set of scalable cluster-based tools in its implementation. In TestZilla, distributed tests are executed through a centralized coordinator and the system provides mechanisms for network topology specification (in a non-emulated cluster setting), file system and process operations, barrier synchronization, and logging and collection of output files. Architecturally, DART and TestZilla share many of the same characteristics although both aim to provide slightly differing feature sets. Unlike DART, which focuses on wide-area distributed applications in an emulated network environment, TestZilla is focused primarily on cluster-based applications in a Windows environment. As a consequence of this, TestZilla relies heavily on Windows-specific features in its implementation. In terms of scalability, both systems rely on scalable cluster-based tools for test control. Unfortunately, given that no published numbers on TestZilla's performance were available, a direct performance comparison could not be made.

ACME [13] provides a framework for automatically applying workloads, injecting perturbations, and measuring the performance and robustness of distributed services based on user specifications written in XML. It targets both emulated network environments such as Emulab and ModelNet as well as real wide-area testbeds such as PlanetLab. In ACME, control, measurement, and injection of perturbations is done through per-node sensors and actuators which, in turn, are controlled through a distributed query processor. Like DART and TestZilla, control in an ACME experiment is done using a centralized experiment control node. Using the query processor, measurements are taken by issuing queries which read desired sensors on multiple nodes in the system. Similarly, actions (e.g., rebooting a node, modifying a link's bandwidth) are invoked by issuing queries that invoke appropriate actuators. Early experience using ACME to evaluate the robustness of three key-based routing routing layers (Chord, Tapestry, and FreePastry) showed that ACME was able to uncover a number of interesting properties and bugs under various workloads and perturbations. Compared to DART, ACME shares many of the same goals. Architecturally, however, ACME differs quite a bit owing to its use of a distributed query processor and the sensor/actuator abstraction as the basis of its implementation.

7 Conclusion

We have developed DART, a framework for distributed automated regression testing of large-scale network applications. We presented the DART system architecture and described the mechanisms DART provides, including scripted execution of multi-node commands, fault and performance anomaly injection, and the runtime layer that supports these mechanisms. We have implemented a DART prototype that implements a useful subset of the architecture and are using this prototype in ongoing testing and benchmarking of PIER, a distributed relational query processor. Our prototype is built on fast and efficient multi-node remote execution and file transfer primitives and incurs reasonable overheads (e.g., 11.3% overhead for a PIER selection query correctness test) for typical distributed tests of interest. Future work on DART includes implementation of additional test mechanisms (e.g., fault injection using Emulab's event system), additional performance optimizations, and further work on gaining experience using DART to test PIER and other wide-area distributed applications. We believe that distributed testing frameworks will be a key enabler towards rapidly building distributed applications that are fast, robust, and deliver high performance across the wide-area.

Acknowledgements

We would like to thank the Emulab team for providing access to the Utah Emulab cluster and for being highly responsive to numerous questions and various feature requests.

References

1. Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (October 2001).
2. Andersen, D. G., Balakrishnan, H., Kaashoek, M. F., and Morris, R. Experience with an Evolving Overlay Network Testbed. *ACM Computer Communications Review 33*, 3 (2003), 13–19.
3. Arpaci-Dusseau, R. H. *Performance Availability for Networks of Workstations*. PhD thesis, University of California, Berkeley, 1999.
4. Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.
5. Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (October 2001).
6. Demers, A., Keshav, S., and Shenker, S. Anaylsis and Simulation of a Fair Queueing Algorithm. In *Proceedings of the 35th IEEE Computer Society International Conference (COMPCON)* (March 1990), pp. 380–386.
7. Ellen W. Zegura, K. C., and Bhattacharjee, S. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96* (March 1996).
8. Freedman, M., Freudenthal, E., and Mazières, D. Democratizing Content Publication with Coral. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation* (March 2004).
9. Hand, S. Self-Paging in the Nemesis Operating System. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation* (February 1999).
10. Huebsch, R., Hellerstein, J. M., Lanham, N., Loo, B. T., Shenker, S., and Stoica, I. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Data Bases* (September 2003).
11. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems* (November 2002).

12. MUTHITACHAROEN, A., MORRIS, R., GIL, T., AND CHEN, B. Ivy: A Read/Write Peer-to-peer File System. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation* (December 2002).
13. OPPENHEIMER, D., VATKOVSKIY, V., AND PATTERSON, D. A. Towards a Framework for Automated Robustness Evaluation of Distributed Services. In *Proceedings of the 2nd Bertinoro Workshop on Future Directions in Distributed Computing (FuDiCo II): Survivability: Obstacles and Solutions* (June 2004).
14. PAI, V. S., WANG, L., PARK, K., PANG, R., AND PETERSON, L. The Dark Side of the Web: An Open Proxy's View. In *Proceedings of the 2nd Workshop on Hot Topics in Networks* (November 2003).
15. PETERSON, L., CULLER, D., ANDERSON, T., AND ROSCOE, T. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets-I* (October 2002).
16. PETROU, D., RODRIGUES, S. H., VAHDAT, A., AND ANDERSON, T. E. GLUnix: A Global Layer Unix for a Network of Workstations. *Software - Practice and Experience 28* (1998), 929–961.
17. RAMASUBRAMANIAN, V., AND SIRER, E. G. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proceedings of the ACM SIGCOMM '04 Conference on Communications Architectures and Protocols* (August 2004).
18. RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols* (August 2001).
19. RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling Churn in a DHT. In *Proceedings of the USENIX 2004 Annual Technical Conference* (June 2004).
20. RIZZO, L. Dummynet and Forward Error Correction. In *Proceedings of the USENIX 1998 Annual Technical Conference (FREENIX Track)* (June 1998).
21. ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms* (November 2001).
22. SAROIU, S., GUMMADI, K. P., AND GRIBBLE, S. D. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems 9* (2003), 170–184.
23. SHENOY, P., AND VIN, H. M. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *Proceedings of the 1998 ACM SIGMETRICS Conference* (June 1998), pp. 44–55.
24. STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols* (September 2001).
25. SUBRAMANIAN, L., STOICA, I., BALAKRISHNAN, H., AND KATZ, R. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation* (March 2004).
26. VAHDAT, A., YOCUM, K., WALSH, K., MAHADEVAN, P., KOSTIC, D., CHASE, J., AND BECKER, D. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation* (December 2002).
27. VOGELS, W. TestZilla: a Framework for the Testing of Large-Scale Distributed Systems. Available from: `http://www.cs.cornell.edu/vogels/TestZilla/default.htm`.
28. WALDSPURGER, C. A., AND WEIHL, W. E. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation* (1994), pp. 1–11.

29. Wawrzoniak, M., Peterson, L., and Roscoe, T. Sophia: An Information Plane for Networked Systems. In *Proceedings of the 2nd Workshop on Hot Topics in Networks* (November 2003).
30. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation* (December 2002).
31. Zhao, B. Y., Kubiatowicz, J. D., and Joseph, A. D. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Tech. Rep. CSD-01-1141, University of California, Berkeley, Computer Science Division, 2001.

Testing Mobile and Distributed Systems: Method and Experimentation

Patrice Laurençot and Sébastien Salva

LIMOS, Université de Clermont-Ferrand,
Campus des Cézeaux, BP 10125 Aubière, France
laurenco@isima.fr, sebastien.salva@iut.u-clermont1.fr

Abstract. Mobile and distributed systems are generally composed of components which interact together with input/output events by using a least a mobile network (GSM, wireless lan), and eventually others heterogeneous ones. Such systems are generally complex so they need to be tested in order to check their reliability. However, no distributed testing tool is proposed. In this paper, we propose a complete method to test such systems and an experimentation which aims to test a WAP application. From a formal specification, the testing method generates test cases and deploys them on a test architecture. This one is composed of several testers which must be synchronized for testing. For the experimentation, we have implemented: a distributed test architecture composed of several testers, a WAP architecture and a WAP application. The experimentation results show that the testing method can be used in practice.

1 Introduction

Since recent years, major progresses have been completed in the mobile network area, particularly concerning Internet and mobile networks. Nowadays, it is possible to access to various services with a mobile phone and to send, receive or search information located on different servers. All these functionalities are obtained with the development of new protocols and applications for mobile telecommunications. Such systems are becoming more and more complex to be implemented and the risk of malfunctioning is more and more important on account of the distributed algorithms used and of the deployment of components on several heterogeneous networks. Validation technics, inherited from the protocol engineering area, are solutions to ensure that a final system has no error by testing it. Different categories of tests can be found in literature. These ones are grouped into two categories:

- the verification technics, which handle a specification and try to prove its correctness (in this case the system can be seen as a white box),
- the testing technics [3, 6, 8, 18], which check various aspects: performance testing, robustness testing, and conformance testing which will be dealed with in this paper. A formal specification is generally needed as well to extract or automatically generate a set of scenario sequences (called

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 37–51, 2005.

’). By executing these test cases on the implementation under test with a tester, these methods can detect incorrectness and compare the specification behavior to the implementation one. Such methods have been widely developed in the communication protocol area.

In conformance testing, implementations are generally seen as “black boxes”, where internal structures are unknown and which are accessible only through one or several interfaces. This is the case for a lot of protocols (for example, ABR for ATM, WAP,...). Therefore, test cases are executed on the implementation by using a test architecture which can access to the implementation interfaces. With systems composed of several mobile components, the classical test architecture cannot be used [17] since these interoperable components must be tested in the same time with a distributed architecture of testers. Some test architectures of distributed systems have been proposed [4, 19, 20, 14] but none of them have been experimented and no tool is proposed.

This paper presents a practical testing method of mobile systems composed of components distributed on heterogeneous networks. This method has been completely implemented and used to test a WAP (Wireless Application Protocol) application. The main goal of this paper is to detail the method implementation and this experimentation. In a first part, we present two test architectures composed of several testers : the first one is composed of two networks, one for the mobile components and one dedicated to the testers for testing. With specific systems, it may be difficult to deploy it, so we describe a second architecture, composed of an unique mobile network on which are connected the testers and the mobile components together. In a second part, we show how we generate, from a formal specification, test cases which check only functional properties of the specification and which can be used with the previous test architectures. The main problem is to split a test case into several ones which can be deployed on a distributed test architecture. Then, we use the second one, which has been implemented in our laboratory, to test a WAP (Wireless Application Protocol [9]) application. This well-known protocol allows to access to Internet sites and data bases for embedded systems like PDA (Personal Digital Equipment) or mobile phone. We detail the components used to test the WAP protocol (servers, PDA), their accessible interfaces and the tools developed to perform the experimentation.

The paper is structured as follows. Section 2 provides an overview of the testing process. Section 3 introduces the different test architectures which can be used for testing mobile and distributed applications. Section 4 presents the method developed to generate test cases which can be executed with distributed test architectures. The implementation of the second test architecture and the experimentation on a WAP application are described in Section 5. Finally, we conclude in Section 6.

2 Protocol Conformance Testing

Testing consists in checking whether the implementation is consistent with the specification by stimulating the implementation and observing its behavior.

Sequences of events, called test cases, are constructed by hands or generated automatically by testing methods from formal specifications, modelled by automata, petri nets or by specific languages such as LOTOS or LDS. Usually, test cases are composed of two kinds of interactions:

- the **outputs**, which model the observation and the sending of a message from the system
- the **inputs**, which model the sending of a message to the system.

In literature, testing methods can be gathered together in two categories:

a) **the exhaustive testing methods**, which involve generation of test cases on the complete specification, execution of the test cases on the implementation and analysis of the test results. To describe the confidence degree between the specification and the implementation, a conformance relation is first defined, then test cases are given or generated from the specification to check if the relation is satisfied or not. Two categories of exhaustive methods can be found :
   - Canonical tester based methods: in this approach, the conformance relations, called implementation relation, are defined with some algebraic properties. Some conformance relations can be found in [16]. An automaton called [illegible] is computed on the global specification so that it can detect any violation of the implementation relation.
   - FSM based approaches: historically, finite state machine (FSM) have been widely used in the networks and telecommunications area to specify communicating softwares such as telecommunication protocols. An FSM transition is fired, in a deterministic way, when an input event is received from the environment. The execution of the transition may produce a possible output event toward the environment. The major work on test generation from this model consists of:
     - the specification of a system by an FSM SPEC
     - the assumption that the implementation of the system can also be described as an FSM IMP
     - the identification of the structure of SPEC on the structure of the IMP.

b) **the non exhaustive testing methods**, which test local parts of implementations [2, 5, 10]. This concept, formalized in [12], aims to check if a set of properties, called a test purpose, is satisfied on an implementation during the testing process. Checking the satisfaction of test purposes on implementation describes a conformance relation. Test purpose based approaches are oriented methods: designers or experts who have a good knowledge of the system, describe the requirements to test, which are generally the important or critical parts of the system. Sometimes protocol standards give guidelines for test selection based on test criteria. In [5], the authors propose an automatic test purpose construction. Then, either test cases are constructed manually or are generated on these requirements and on specification parts, reducing the specification exploration in comparison with exhaustive methods (reducing in the same time the test costs).

Afterwards, test cases are executed on the implementation by mean of a test architecture. This one describes the configuration in which the implementation will be experimented which includes at least the interfaces of the implementation (called PCO, [illegible]) and the tester which applies the test cases on the implementation. Test architectures can be found in [12, 17] for untimed system testing. The execution of such test cases leads the tester to emit requests to the implementation (inputs) and then to wait for answers (outputs). Depending on the observed results, the tester can deduce a final verdict for the test: [illegible] which means that the implementation conforms the specification, [illegible] which means that we cannot conclude or [illegible].

3 Test Architectures of Distributed Systems

Test architectures, suggested by the standard[12], cannot be used since different entities cooperate in the network to provide a desired service. To test such systems, we need to observe and to analyze the transit of input and output events, received or transmitted from each component. So it's necessary to introduce different Points of Control and Observation (PCO), generally at least one for each component. These PCO are designed to access to the component interface: that is they can send events to the component (by the point of control) and observe the results (by the point of observation).

Several test architectures of distributed systems have been proposed [4, 19, 20, 14]: these ones can be centralized systems where a single tester is connected to some PCO and sends or receives events from all the component interfaces. An example of centralized architecture is given in Figure 1. Such architectures are generally easier to implement since only one tester is needed. However, the PCO involves a high traffic of data which requires a specific network and which may overload the system.

So, a second category of distributed test architectures has been proposed. These ones are composed of local testers, each of them checks one component and communicates with the others ones. These communications are necessary to synchronize the testers between them and to synchronize the execution of the system components.

The local testers also produce and send local verdicts which must be analyzed by a coordinator tester to obtain a final one. To communicate, these testers can be connected to:

- a dedicated network. In this way, each local verdict can be got back as soon as this one is produced, without interfering with the system. If one local verdict is FAIL, the coordinator tester can directly stop the test after receiving it.
- the network of the system. Local verdicts cannot be sent to the coordinator tester once they are created since the network may be used by the system components. Consequently, the local verdicts are sent to the coordinator tester once the test is terminated.

These two solutions are detailed below.

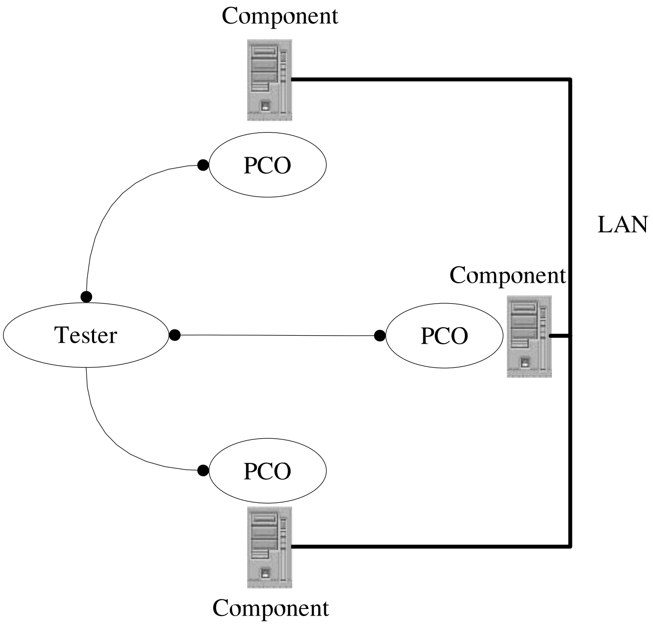


Fig. 1. A centralized test architecture

3.1 Test Architecture with a Dedicated Network

Such test architectures use a dedicated network connecting each tester with. These architectures require additional equipments, since each entity has at least two connections: one with the other components for the regular traffic, and the other connection used for the data exchanged for testing. The architecture is depicted in Figure 2.

The main advantage of this architecture is the complete independence between the regular traffic and the data exchanged between the testers. As there is no interference, we are sure that the verdict which is obtained reflects the reality. Even more, if an error occurs, the PCO which detects it, can alert the coordinator tester so the test can be stopped immediately with a FAIL verdict. However, mobile applications cannot be always tested with such architectures: a mobile terminal must have access to the two different networks simultaneously. In practice, this is not always possible or difficult to set up. For example, a mobile phone has generally only one network interface (GSM interface). A wireless equipment should have two interfaces, each one linked to a different access point. As the mobile terminal can move, we should check that the two cards stay on different networks. Therefore, this architecture is hard to implement (because of hardware constraint), but the verdict of the test can be given rapidly.

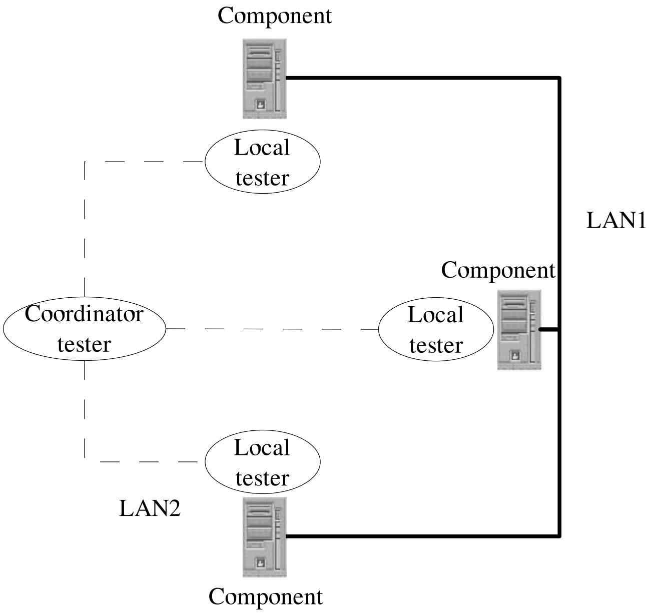


Fig. 2. Test architecture with a dedicated network

3.2 Test Architecture Using the System Network

In this case, the local testers and the coordinator one are connected directly on the system networks, as the system components. So, the regular data of the system and the specific data of the test take the same medium. To avoid collision, the local testers wait the end of the test before sending to the coordinator tester the local verdicts. This solution does not perturb the test since the results are sent after it is completed.

This architecture has the advantage to be used with most of the mobile applications, since it's the same medium which is used to transmit regular or test data. However, the test must be completely executed before obtaining a verdict even it's a FAIL one, whereas the test could be stopped immediately with the first architecture. Moreover, we must suppose that the local testers have sufficient memory to store the local verdicts.

In the next section, we introduce the method which is used to create a test case and to distribute it to the different local testers.

4 Test Methodology

Many testing methods have been proposed to generate automatically test cases from untimed specifications [7, 6, 5, 11]. To use them, specifications must be modelled with a formal language. Among the various existing ones (LOTOS, LDS, Petri Nets, automata...), we propose to use the IOSM [1, 16] (Input Output State Machine).

Definition 1 (Input Output State Machine).
$\mathcal{A}$ $< \Sigma_{\mathcal{A}}, S_{\mathcal{A}}, s^0_{\mathcal{A}}, E_{\mathcal{A}} >$

$\Sigma_{\mathcal{A}}$ $S_{\mathcal{A}}$ $s^0_{\mathcal{A}}$

$E_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times (\{?, !\} \times \Sigma_{\mathcal{A}}) \times S_{\mathcal{A}}$
!
$< s, a, s' >$ s s'
a

Furthermore, we consider that for a distributed system $\mathcal{A}$, the language $\Sigma_{\mathcal{A}}$ is the union of languages used by every components. And, for two components x and y, the two languages are disjoined, $\Sigma_{\mathcal{X}} \cap \Sigma_{\mathcal{Y}} = \varnothing$. This property guarantees that each entity, and consequently each tester, takes into account only the messages concerning it. An example of IOSM is given in Figure 6.

We propose to use a test purpose based method [5, 8] (Section 2) which generates test cases from requirements given by designers. From these requirements, called a test purpose, this method generates the test cases which aims to check whether the test purpose is satisfied or not on the implementation. However, this generation is not sufficient: in the previous test architectures, we have considered that each component of a distributed system is connected to a local tester. This implies that the test cases must be distributed on the local testers. So, a test case will consist of several "dedicated-tests", allocated to each tester. Each one will perform its dedicated-test and will communicate with the other ones to synchronize the tests of every components.

4.1 Generation of the Dedicated Test Cases for Local Testers

The algorithm, introduced below, aims to extract from a test case ω, each local test case ω_t, intended for each local tester t. Furthermore, it adds to the local test cases, some synchronization data needed to synchronize the testers between them. Synchronizations are obtained by one or several locks, modelled by data exchanged between testers and designed by ($-sync^p_j, +sync^p_j$):

- $+sync^p_j$ locks the current tester until a message of synchronization is received from the tester j.
- $-sync^p_j$ represents the sending of synchronization to the tester "j" with the number p. This one unlocks the tester "j" which can continue to execute its test case on the component until another lock or until the end of the test case.

To sum up the algorithm, it consists of dividing a test case into several dedicated ones by analyzing its symbols and by determining which tester must use them. When two successive interactions (symbols) are not destined to the same local tester, a synchronization is used: $+sync_j^p$ is added to lock the tester which must execute the second interaction. $-sync_j^p$ is added in the test case devoted of the tester which must execute the first interaction. An example of test case generation is given in the following Section.

Algorithm

```
Hypothesis: The number of testers is known and is equal to N.
Input: A test case $\omega = \gamma_1\gamma_2\gamma_3...\gamma_x$, with x the number of requests.
Output: N dedicated-test sequences, $\omega^1, \omega^2$...
BEGIN:
for k from 1 to N do
    $\omega^k \leftarrow 0$
end for
p $\leftarrow$ 1
for k from 1 to x-1 do
    Read $\gamma_k$ in $\omega$, search for the tester $t_i$ which has this alphabet
    Read $\gamma_{k+1}$ in $\omega$, search for the tester $t_j$ which has this alphabet
    If $(t_i \neq t_j)$
    /* installing coordination */
        If ($\gamma_k$ is an emission) then
            $\omega^{t_i} \leftarrow \omega^{t_i}$ + "$-sync_{t_j}^p$" + $\gamma_k$
        else $\omega^{t_i} \leftarrow \omega^{t_i}$ + $\gamma_k$ + "$-sync_{t_j}^p$"
        end if
        $\omega^{t_j} \leftarrow \omega^{t_j}$ + "$+sync_{t_i}^p$"
        p $\leftarrow p + 1$
    else $\omega^{t_i} \leftarrow \omega^{t_i} + \gamma_k$
    end if
end for
Read $\gamma_x$ in $\omega$, search for the tester $t_i$ which has this alphabet
$\omega^{t_i} \leftarrow \omega^{t_i} + \gamma_x$
END
```

Each local tester produces a local verdict: PASS if all the traces correspond to the test case, INCONCLUSIVE if the tester cannot execute the test case, or FAIL otherwise. The global test verdict, given by the coordinator tester is given by this definition:

Definition 2 (Test verdict).

[illegible] l_1, [illegible] l_n [illegible] t_1, [illegible] t_n [illegible] T [illegible]

$$T = \begin{cases} \text{[illegible]} \; \forall 1 \leq i \leq n \;, l_i = PASS \\ \text{[illegible]} \; \exists 1 \leq i \leq n \mid l_i = INCONCLUSIVE \\ \text{[illegible]} \end{cases}$$

5 Experimentation and Results

In this section, we present our experimentation and results of a WAP system test. This system is composed of a WAP architecture (WAP protocol, gateways, HTTP server, database,...) and of an application which aims to update or search information in a database, specialized in cattle diseases.

Before describing the test architecture and our implementations, we briefly expose the WAP and its requirements.

5.1 The WAP (Wireless Application Protocol) and Our WAP System

The WAP is a result of continuous work to promote industrywide specifications for technology useful in developing applications and services that operate over wireless communication networks. The aim of the WAP is to access to Internet with devices which have less powerful CPU, less memory, restricted power consumption and different input devices.

On the one hand, the WAP gathers several protocol layers which allow the access of HTTP servers and databases: the Wireless Application Environment (WAE) includes a micro-browser which permits to view the environment information. The Wireless Session Protocol (WSP) provides the application layer of the WAP with a consistent interface for two session services. The first one is connected-oriented and operates above the Wireless Transaction Protocol (WTP). The second one is connectionless and operates above a datagram service (UDP). All these layers are involved in the communication and their interactions have been described using formal methods by the Platonis project [15].

On the other hand, the WAP represents a programming model, similar to the WWW one. It defines a set of standard components that enable communication between mobile terminals and network servers, including standard naming model, content typing and standard content formats (wml language). This wml language, close to the html one can be used to construct pages accessible via a wml browser. To have a full working WAP service, a gateway is used to transform the data coming from wireless communication with a WAP encapsulation to data understandable by an HTTP server. For this article and our experimentation, we use the open source Kannel[13] gateway since its implementation respects the standard established by the WAP Forum.

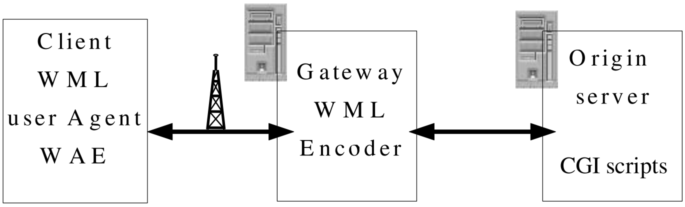


Fig. 3. WML user agent logical architecture

The WAP architecture, that we have deployed, is composed of a PDA connected to a GSM phone by an IrDA port. The PDA runs a WAP navigator, written with Embedded C++, which implements the WSP and WTP layers. With the WAP protocol, this one can access to an HTTP server via a Kannel Gateway. The HTTP server and the Kannel gateway are connected by an Ethernet network. To access to the HTTP server, the PDA must obtain an IP address, so we implement a PPP (Point to Point Protocol) server. This one is set on the same computer running the Kannel gateway in order to simplify the WAP system.

The WAP application is a "classical" Internet one: the WAP navigator proposes different wml pages which allow to request information on a database or to update it. The HTTP server contains several CGI programs which return wml pages to response at the previous requests.

5.2 Test Architecture and Testers Implementations

Since we use a GSM phone which has only one network interface, we use the second test architecture. The test architecture, devoted to our system, is illustrated in Figure 4 and described bellow.

Three testers have been implemented : two of them have the mission to detect wrong messages in the Kannel gateway and in the local network connected to the HTTP server. The third tester is a coordinator, located on the PDA. Each tester is composed of two programs: [illegible] for traffic inspection and [illegible] for giving the local verdict. The "dedicated test cases" are loaded on each PO_analysis. During the test execution, each PO_analysis compares its local test case with the frames that are stocked by PO_trace. If no error is detected, PO_analysis sends a PASS verdict at the end, if PO_trace does not respond for any reason it sends INCONCLUSIVE, otherwise it sends a FAIL one.

The tester number 1 observes the traffic received and emitted by the WAP gateway. The open source Kannel gateway was modified for installing the trace tools. The Kannel software is structured as different layers, each one implemented by a thread which communicates with the other ones by exchanging messages. Different point of observation are inserted between each layer, and a thread is

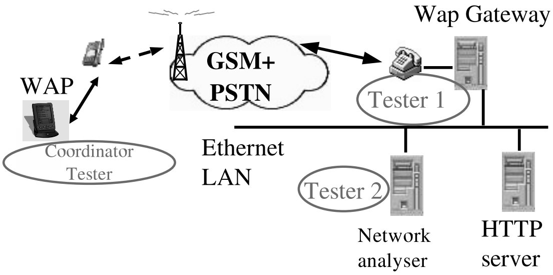
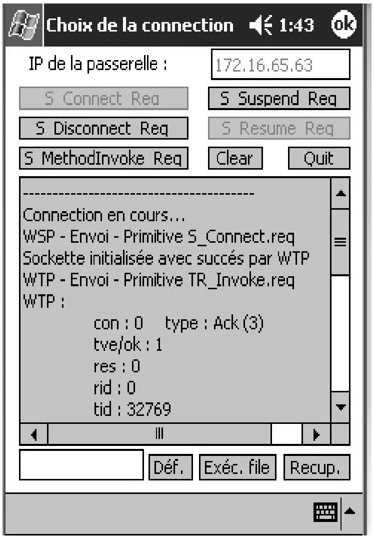


Fig. 4. Test architecture for the experimentation

added to analyze the traffic of the gateway. In fact, the modifications to install the trace tools are very small. Each time, a thread wants to send a message to another thread, the message is duplicated in a file before being emitted. The analyse thread contains a PO_trace_in which retrieves incoming traffic, a PO_trace_out which retrieves outgoing traffic and a PO_analysis which inspects the different traces and gives out the local verdict.

Since the WAP gateways are in general connected to Internet via a local network, the tester number 2 corresponds to a network analyzer, that will not perturb the network while the frame capture. For portability reasons, this analyzer was implemented in Java using [illegible]. Once all the test case is executed and inspected by PO_trace, the thread PO_analysis produces the local verdict and sends it to the coordinator tester.

The coordinator tester, located on the mobile system, must be able to send and receive different frames as well as the different local verdicts. A PDA running Windows CE is used, making it easier to program and establish a connection to GSM through a mobile phone equipped with an IrDA port. The WAP navigator, which implements the WSP and WTP layers with threads, provides also a graphical user interface that enables the load of the test cases. Figure 5 shows the graphical user interface of the PDA with the beginning of a test case. The thread PO_trace listens for all the messages received or sent by the WTP layer, while PO_analysis gives indications on the evolution of the test on the user interface, and produces the final verdict as well. if all the received local verdicts are PASS, the final verdict is PASS, otherwise it can be FAIL or INCONCLUSIVE. These softwares have been programmed with Embedded Visual C++.



IP de la passerelle : 172.16.65.63

```
----------------------------------------
Connection en cours...
WSP - Envoi - Primitive S_Connect.req
Sockette initialisée avec succés par WTP
WTP - Envoi - Primitive TR_Invoke.req
WTP :
            con : 0    type : Ack (3)
            tve/ok : 1
            res : 0
            rid : 0
            tid : 32769
```

Fig. 5. The user interface of the PDA

5.3 Test Case Generation and Experimentation Results

For the experimentation, we propose to test the “get” function of the WAP which requests and receives wml pages from http servers. This function is transcribed by the service [illegible] of the WSP layer. As we want a connected mode (which will use the WTP layer), we will have to add the S_connect.req primitive in the test purpose.

To generate test cases, we use the formal specification of the WSP layer whose a partial view is given with the IOSM of the figure 6.

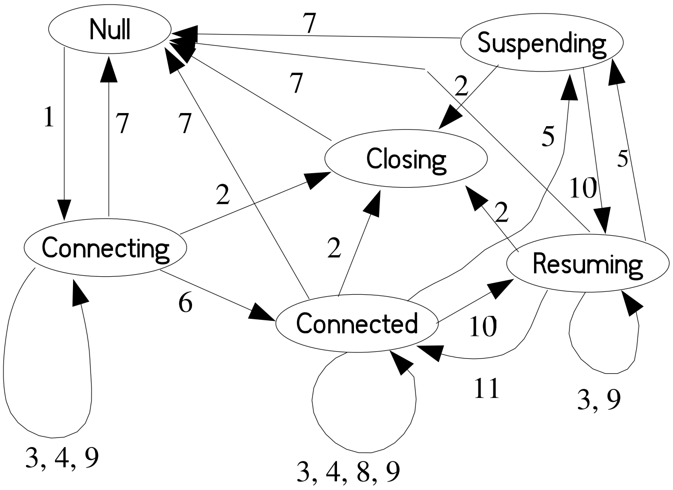


Fig. 6. Some WSP Layer Primitives

legend :	1 : ! Connect.req	5 : ! Suspend.req	9 : ? MethodAbort.ind
	2 : ! Disconnect.req	6 : ? Connect.cnf	10 : ! Resume.req
	3 : ! MethodeInvoke.req	7 : ? Disconnect.ind	11 : ? Resume.cnf
	4 : ! MethodAbord.req	8 : ? MethodeInvoke.ind	

First, we construct the test purpose $\xrightarrow{S\_connect.req}\xrightarrow{S\_MethodeInvoke.req}$ which allows to instantiate the connected mode and to ask for a wml page. With this test purpose and our description of the WSP layer, we generate a first test case by using the test purpose method TGV ([8]). This test case is composed of 19 transitions. Then, we use the algorithm described in Section 4 to create the three “dedicated-tests”. These ones are given bellow:

Coordinator Tester:
? S_connect.req + -$sync^1_{PO1}$ + ! TR_invoke.req + +$sync^2_{PO1}$ + ? TR_result.ind + ! S_connect.cnf + -$sync^3_{PO1}$ + ! TR_result.res + ? S_MethodInvoke.req + -$sync^4_{PO1}$ + ! TR_invoke.req + +$sync^{10}_{PO1}$ + ? TR_result.ind + ! S_MethodResult.ind -$sync^{11}_{PO1}$ + ! TR_result.res

Tester 1:
+$sync^1_{PCO}$ + ? TR_invoke.ind + ? TR_invoke.res + -$sync^2_{PCO}$ + ! TR_result.req +$sync^3_{PCO}$ + ? TR_result.cnf +$sync^4_{PCO}$ + ? TR_invoke.ind + ? TR_invoke.res + -$sync^5_{PO2}$ + ! TCP_connexion.req +$sync^6_{PO2}$ + ? TCP_connexion.ind + -$sync^7_{PO2}$ + ! TCP_ack.req + -$sync^8_{PO2}$ + ! TCP_data.req +$sync^9_{PO2}$ + ? TCP_data.cnf + -$sync^1_{PCO}0$ + ! TR_result.req +$sync^{11}_{PCO}$ + ? TR_result.cnf

Tester 2:
+$sync^5_{PO1}$ + ? TCP_connexion.ind -$sync^6_{PO1}$ + ! TCP_connexion.res +$sync^7_{PO1}$ + ? TCP_ack .req +$sync^8_{PO1}$ + ? TCP_data.ind + -$sync^9_{PO1}$ + ! TCP_data.res

During the first experimentations, we always obtained a FAIL verdict from the coordinator tester, located in the PDA. We searched in the PO_trace for some errors and we found that instead of receiving a TR_result.ind, the PDA received an Ack frame for a confirmation of the TID (Transaction IDentification). The TID, which is increased for each frame, is used to number all the frames of the WAP to easily detect a loss. At the beginning of a communication, if a client sends a frame with an unexpected TID to the server, this one asks for a confirmation (with an ACK) to update its TID. Consequently, the WAP navigator, executed by the PDA, sent frames with bad TID. This error was confirmed by the PO_traces of the tester located on the Kannel server. So, we correct this error on the WAP navigator. Afterwards, we have experimented once again and we have obtained a PASS verdict, which means that the mobile application can ask for an information and receive a response in the connected mode of the WAP protocol. Other tests have been completed to check different functionalities of the WAP application. All the tests have been created with the aim of testing a functionnality of the application, and so the test purposes were created by hand and their lengths were less than six primitives.

6 Conclusion

We have introduced in this paper different test architectures and a testing method which can test mobile and distributed applications. One test architecture has been completely implemented and used to test a WAP application. The experimentations show that the method and the test architecture can be used in practice to detect errors on components distributed in different heterogeneous networks. We have used the GSM network as a mobile one, but other trace tools (PO_trace) have been implemented to use wireless networks (802.11). The networks, we have considered for the test architecture, are LAN, however a perspective could be the use of Internet to connect the components of the application: in such as case, the deployment of the test architecture could not be done manually. An automatic deployment of test architectures could be planned and proposed, that is at least an automatic download and installation of the testers on the components (or stations connected to theses ones).

References

1. R. Alur and D. Dill. The theory of timed automata. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *LNCS*, pages 45–73. Springer-Verlag, 1992.
2. I. Berrada, R. Castanet, and P. Felix. A formal approach for real-time test generation. *WRTES, satellite workshop of FME symposium*, pages 5–16, 2003.
3. G.v. Bochmann, G. Das, R. Dssouli, and M. Dubuc. Fault Models in Testing. In *Proceedings of the International Workshop on Testing of Communicating Systems IWTCS'91*, 1991.
4. L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 1999.
5. R. Castanet, C. Chevrier, O. Kon, and B. Le Saec. An Adaptive Test Sequence Generation Method for the User Needs. In *Proceedings of IWPTS'95, Evry, France*, 1995.
6. A. Cavalli. Different approach to protocol ans service testing. *Proceedings of the Twelfth IFIP Workshop on Testing of Communicating Systems (IWTCS'99)*, September 1999.
7. T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.
8. J. Cl. Fernandez, C. Jard, T. Jron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *CAV'96. LNCS 1102 Springer Verlag*, 1996.
9. WAP forum. Wap specification. http://www.wapforum.org.
10. H. Fouchal, E. Petitjean, and S. Salva. Testing Timed Systems with Timed Purposes. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, RTCSA'00 (Cheju Island, South Korea),IEEE Computer Society*, pages 166–171, December 2000.
11. S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite-state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991.
12. ISO. Conformance Testing Methodology and Framework. International Standard 9646, International Organization for Standardization — Information Technology — Open Systems Interconnection, Genève, 1991.
13. kannel group. Kannel, open source wap and sms gateway. http://www.kannel.org.
14. G. Luo, R. Dssouli, G. Bochman, P. Venkatraam, and A. Ghedsami. Test generation with respect to distributed interfaces. In *Computer Standards and Interfaces*, volume 16, pages 119–132, 1994.
15. A. Mederreg, F. Zaidi, P. Combes, W. Monin, R. Castanet, M. Mackaya, and P. Laurenot. Une plate-forme de validation multi-protocoles et multi-services - rsultats d'exprimentation. *Colloque Francophone de l'ingénierie des Protocoles, CFIP*, October 2003.
16. M. Phalippou. *Relation d'implantation et hypothses de test sur des automates entres et sorties.* PhD thesis, Univ. of Bordeaux, September 1994.

17. O. Rafiq, R. Castanet, and C. Chraibi. Towards an environment for testing osi protocols. *Proc of the International Workshop on Protocol Specification, testing and Verification*, 1985.
18. S. Salva and P. Laurençot. Génération de tests temporisés orientée caractérisation d'états. *Colloque Francophone de l'ingénierie des Protocoles, CFIP*, October 2003.
19. A. Ulrich and H. Knig. Architecture for testing distributed systems. In *Proc of Inter. Workshop on testing of Communicating Systems, IWTCS'99*, 1999.
20. T. Walter, I. Schieferdecker, and J. Grabowski. Test architectures for distributed systems - state of the art and beyond. In *Testing of Communicating Systems*, 1998.

A UNITY-Based Framework Towards Component Based Systems

I.S.W.B. P y [1], T. .J. Vo [2], A. A [1], n S.D. Swi [1]

[1] Informatica Instituut, Universiteit Utrecht
wishnu@cs.uu.nl
http://www.cs.uu.nl/staff/wishnu.html
[2] Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
tanja@iti.upv.es
http://www.iti.upv.es/~tanja

Abstract. Compositionality provides the foundation of software modularity, re-usability and separate verification of software components. One of the known difficulties, when separately verifying components, is producing compositional proofs for progress properties of distributed systems. This paper offers a UNITY-based framework to model distributed applications which are built with a component based approach. The framework enables components to be abstractly specified in terms of contracts. Temporal properties are expressed and proven in the UNITY style. Compositional reasoning about components' properties, including progress, is supported. The semantical model is simple and intuitive.

Keywords: component based applications, compositionality, verification.

1 Introduction

o on n o l , li M, BA, n J v B n , l in li - ion il o o on n in y llin o ' o ion . T iff n o on n n own n on oll y o li ion , w i y n on y iff n in n lo ion . on n ly, o on n li ion n i lly i i y w o o l o i v o v i o n i o n .

V i yin lo l o y o o on n y i o li - w y no v o o o o ll i o on n . In , w v o ly on i i ion o . How v , v i yin - v ion o o o i ' i ion w il o o in i i o o- n n i nown o i l [, 5, 1, , , 9]. In o o l o in lo l o l o i , o on n will v o off on in o on- . M ly i yin - n o - on i ion o n o j ' o ion , li o x l in L, i lly no i n .

T i off UNITY- wo o i y o on n o n li ion n o in o l o i o li ion o on o i o on n .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 52–66, 2005.

2 Overview on the Model

W will i o l in BA: n on i o n . Bo o in n i i , n o i i l , w y n on in o ly. T o in n i i in y o in (o) ovi y o j [1][2]. j v il l . o on n o j , o v io on , only v l li i o n o in o ion o lv . T in o ion y v l i l i own in , n o on n i o v in i in i on [3]. An o j i o n y on i l o ll .

T wo , i in i , off o l no ion o o j , on , n li ion , n o l w llow o in lo l o- i , in l in o , o n li ion in o o i ion l w y o i ion o i o on n . n n i o o on n-on l ion. T n n l ion i n in w y in o on n-on on i n y i n o on n' o o fl xi ili y in i in o i o on n o i on (w il ill off in on i n on).

3 Preliminaries

Predicate Confinement. P i i y o o . A - i i y o v i l (w i n **conf**) i o no on in v l o ny v i l o i . A l o , i i o v i l o n x ion e, n e i on n y . W w i q **conf** o vi **conf** n q **conf** .

Actions. An i n o i , in in , n non- ini i n- i ion. An ion n o ll y n ion o niv o , no y State, o $\mathcal{P}$(State).

A ion n (l i l) i n n o ion . I i o v i l , skip i n ion o no n v i l in .

[1] This is consistent with Szyperski's definition of *object* (essentially: an object is something that has state, behavior, and encapsulation) [20], which is quite commonly accepted.

[2] We will not venture into complex features, such as inheritance and the ability to pass object reference, or to pass an entire object, through an operation call. Furthermore, our model is an abstract model: details of implementational nature, such as parameters marshaling, object deployment, and optimization of resources' utilization will not be visible in the model.

[3] This is also consistent with Szyperski's definition of *component* [20], essentially: component is a unit of composition with contractually specified interfaces and subject to composition by third parties. Our definition is stricter by saying the only knowledge we can rely on, placing ourselves as a third party, about a component is its contracts.

ion no y g --> , nin will x i g i , o wi ion v skip.

I n b ion , $\sqcup b$ i n ion i v o b. So, ($\sqcup b$) = $\cup$ b . I i o ion n $\sqcup$ i o n o ($\sqcup$: $\in$:).

W w i {var x; } o in o lo l v i l x. T nin i x in o Ho i l ollow :

$$\{\ \}\ \{\texttt{var}\ x;\ \ \}\ \{q\} \overset{d}{=} \{\ \}\ \ [x'/x]\ \{q\}$$

w $[x'/x]$ n ion o in y l in x in wi v i- l x'.

Action Refinement. W n ollowin no ion o n n ov ion —i i v i n o n on , . . in [3]. L o v i l , n l i (i in n o n inv i n). A ion b ion (o i n ion o b) wi o n , i b n i i l w v n o on v i l in , in ol ini i lly, o i i . Fo lly:

$$\vdash\ \ \sqsubseteq b$$
$$\overset{d}{=}$$
$$(\forall\ \ q :\ \ q\ \textbf{conf}\ \ : \{\ \wedge\ \}\ \ \sqcup \textsf{skip}\ \ \{q\} \Rightarrow \{\ \wedge\ \}\ b\ \{q\})$$

Notation. W will l o n o o i , n l o o l v io . Fo x l , bje = (prg :: $rogr\ m$ ops :: { er o }) n y bje on i in o wo- l n l . I $x($ $M)$ i v l o i y , n x.prg = n x.ops = M.

4 UNITY

W will o i in l UNITY o o o [6] o o x n ion. W o l v o o n w-UNITY [1] in o o o o in in i l o l n . oi , ow v , i j iv on , w n ”ol ” o o i ly o in i iv .

4.1 Programs

W will n UNITY o y l o i y :

$$rog_{\text{UNITY}} \overset{d}{=} (\textsf{acts} :: \{\quad o\ \}\ \textsf{init} ::\ \ red\ \textsf{pub} :: \{\quad r\}\ \textsf{pri} :: \{\quad r\})$$

.init i i i yin ’ o i l ini i l , .pub i o ’ li ( ) v i l , n .pri i o ’ iv (lo l) v i l . W w i .var o o .pub$\cup$ .pri. I li i ly, .init o on n y .var; .pub n .pri i join ; n o v y ion $\in$ .acts, i ol o v y , i non- y.

An x ion o UNITY o i in ni , in n ion i l non ini i lly. S l ion i w ly i , v y ion i l in ni ly o n.

W o no x l o o w i n n i ly in UNITY. UNITY ion v n ion o n i l o , w i y v l i l n w i n in no l n . In Mi ’ wo [1]: UNITY o ly o x ion o i on i n n i l o , y i yin on i ion n w i n i l o i o x .

W n o o in , w o on n i iv n ni n o n i iv v i l . So, w n o o in n Q, w now n in .pri n Q.pri o no l wi n in, iv ly, Q.var n .var. Uni n n, o x l , i v y xin n o ll iv v i l o o wi o ’ n . W will no on n o lv wi i .

o o in wo o n nnin in ll l. T vio o ll l o o i ion o n Q i o ll y $\|Q$ w i i n ollow :

Definition 4.1: Parallel Composition

$$\|Q \stackrel{d}{=} (\ .\mathsf{acts} \cup Q.\mathsf{acts}\quad .\mathsf{init} \wedge Q.\mathsf{init}\quad .\mathsf{pub} \cup Q.\mathsf{pub}\quad .\mathsf{pri} \cup Q.\mathsf{pri})$$

4.2 Properties

A i i [4] o , no y ⊢ sinv , i i ol ini i lly, n i i in in y v y ion o . A i j i n i xi on inv i n i lyin j.

Definition 4.2: Strong Invariant

$$\vdash \mathsf{sinv}\quad \stackrel{d}{=}\quad .\mathsf{init} \Rightarrow\quad \wedge\ (\forall\ :\ \in\ .\mathsf{acts} : \{\ \}\quad \{\ \})$$

To i y y n on - o o i w x n UNITY o o o [15]. W low o onv ni n .

Definition 4.3: UNITY operators

1. $|\!-\!-\quad \mathsf{unless}\ q$

$$\stackrel{d}{=}$$

$$\vdash\ \mathsf{sinv}\quad \wedge\quad q\ \mathbf{conf}\ .\mathsf{var}\ \wedge\ (\forall\ :\ \in\ .\mathsf{acts} : \{\ \wedge\ \wedge \neg q\}\quad \{\ \vee q\})$$

. $|\!-\!-\quad \mathsf{ensures}\ q$

$$\stackrel{d}{=}$$

$$|\!-\!-\quad \mathsf{unless}\ q\ \wedge\ (\exists\ :\ \in\ .\mathsf{acts} : \{\ \wedge\ \wedge \neg q\}\quad \{q\})$$

[4] We are going to use invariants to parameterize UNITY properties, in the style of Sanders [18]. Strong invariants are however used here instead of just invariants (predicates that hold through out any execution of a given program) as in [18], because the later cause a certain technical problem [16].

4.3 Refinement

W will ollowin i l no ion o n n on UNITY o .

Definition 4.4: Program Refinement and Abstraction
Fo o v i l , n i i in n o on inv i n o , w n Q i n n o (o i n o Q) ollow :

1. $|\!\!-\!\!- \sqsubseteq Q \overset{d}{=} .\mathsf{pub} \subseteq Q.\mathsf{pub} \ \wedge \ .\mathsf{pri} \subseteq Q.\mathsf{pri} \ \wedge \ Q.\mathsf{init} \Rightarrow .\mathsf{init}$
$\wedge$
$\forall b : b \in Q.\mathsf{acts} : \ \vdash \sqcup .\mathsf{acts} \sqsubseteq b$

. $\vdash \sqsubseteq Q \overset{d}{=} .\mathsf{var} \ |\!\!-\!\!- \sqsubseteq Q$

So, n inv i n o , $\vdash \sqsubseteq Q$ n v y ion o Q v , wi o v i l in , no wo n o ion o , o i j i (no o w n n ion l v l).

5 Specification of Objects and Components

In o o l o on o v ion o o o i , w n on i y o o o i n v i o on n i o o wi o nvi on n B. on i o y $\|B \vdash \mapsto q$. S o w now i o i iv n ol ly y o o-n n . I B i n o on nvi on n Q, n w n x o y will v in $\|Q$. To x i in o -onin , w n n w o x n UNITY o o , wi w i no ion o " o i iv n ol ly y " n i .

Definition 5.1: Extended UNITY operators
L n B UNITY o . W n :

1. $\triangleleft\|B \ \vdash \ \mathsf{ensures}\ q$
$\overset{d}{=}$
$\|B \ \vdash \ \mathsf{unless}\ q \ \wedge \ (\exists \ : \ \in .\mathsf{acts} : \{ \ \wedge \ \wedge \neg q\} \ \{q\})$

. $\triangleleft\|Q \ \vdash \ \mapsto q$ i n $(\lambda \ q. \triangleleft\|Q \ \vdash \ \mapsto q)$ i ll n i iv n i j n iv lo o $(\lambda \ q. \triangleleft\|Q \ \vdash \ \mathsf{ensures}\ q)$.

Now w n o i o i l o i y o o-i in on i v ion n in l o nvi on n i on o i o o wi . Mo i lly, o v y o o y o o q y , w n i in o $\triangleleft\|B$, will v w n i o o wi ny o Q n B.

Theorem 5.2: Preservation of $\mapsto$

$$\frac{\triangleleft \| B \;\; \vdash \;\; \mapsto q \quad \wedge \quad j \vdash B \sqsubseteq Q \quad \wedge \quad \Rightarrow j}{\| Q \;\; \vdash \;\; \mapsto q}$$

T l ' i i on inv i n o . How v , y no v l in i on , o x l i x o oo o i in n l . How v i i i n i on x o w inv i n j, in w n in on l ion ov y owin n n on w j.

6 Objects and Their Operations

In o o l, n . . . i UNITY o x o o o i v i l o i nvi on n . T i o i ll o o o j . T x o v i l ll o v i l . A o v i l i i : nvi on n n only in o vi o o ion ovi y o j . Any o n loy n o j y n l in i wi n y in i l n in ov i ion o in li v i l .

6.1 Semantical Model

W will n i lly o l n o j x wi l o i y :

$$bje = (\mathsf{prg} :: rog_{\mathrm{UNITY}} \; \mathsf{ops} :: \{ \; er \; o \; \})$$

w $x.\mathsf{prg}$ o l o j ' in n l o , $x.\mathsf{ops}$ i o o ion off y o j , n er o no niv o o ion o j n off .

T l o on o ov lo o y wo on o j , . . i x i n o j , $x.\mathsf{pub}$ i l o $x.\mathsf{prg.pub}$. W l o ov lo $\|$ n $\triangleleft\|$, . ., $x\|Q$ n $x.\mathsf{prg}\|Q$.

Two o j v n i y off o li v i- l n o ion . An o ion ollowin : m $me($ $r) =$ do b, w i o in o m me, r i - o ol n v l o m me, n b i n i in o ion' o y. T n o no o olli wi n o ny v i l o x. P i y v l o y n , in li y n i llow . An o ion no o lo l in o ion, o n li v i l o o j i lon o.

I x i n o j o w i m me lon o, n ll o m me i - no y $x.m$ $me(e\;z)$. I b i o y o m me, ff o ll i iv l n o:

```
atomic{var return; := e; b; z := return}
```

T x ion o n o ion i o . T i y no i n , i i , o x l , wo on l . How v , w y o in w i o ion n ly x in ll l. W will w y i i , n l v i o i l n o oo i i ili ion o o j .

Object Properties. Sin nvi on n o n o j x n only in wi i o i o ion , wo o i l nvi on n o x n - i y ollowin UNITY o :

$$x.\mathsf{env} \overset{d}{=} (\quad x.\mathsf{init} \;\; x.\mathsf{pub} \;\; \emptyset)$$

w i o ion o llin ll o i l ll o o ion in x.ops:

$$= \{\texttt{var} \quad z;\ x.m_1(\quad z)\} \parallel \ldots \parallel \{\texttt{var} \quad z;\ x.m_k(\quad z)\}$$

o ll o ion $m_1 \ldots m_k \in x$.ops. So, ny l o nvi on n Q o x i n n o x.env. Fo lly:

$$Q \text{ i o nvi on n o } x \overset{d}{=} (\forall \quad :: x.\mathsf{pub} \quad \vdash \ x.\mathsf{env} \sqsubseteq Q)$$

Any unless n ↦ o i ov n wi o $x \| x$.env n $x_{\lhd} \| x$.env - iv ly, will v w n x i o o wi ny o nvi on n .

7 Contract

A o on n i n o j o no l ll in o ion o i l o i nvi on n . In , i off i w o j o . A on i in in , o j i o li o li ny in i o i in on . A i l o o on n j li o o ion off y o j . S on n n n y in o in o ion in i , n lin o in o o i o o j . o , n nin on n o j l l , n i v i ion o x n iv .

W will ion l ion o S ion .3 o on . W n o j on o on o j x. T ni ion o ⊑ ll ow o i x will l l . T i ow l ion l ion iv v n w o in on i l l v l o o in i in ow il i (ow o x w w n o x o in).

7.1 Semantical Model

Fo lly, w will n on wi l o i y :

$$o \quad r \quad = (\mathsf{smodel} :: \quad bje \quad \mathsf{inv} :: \quad red \ \mathsf{progress} :: \{ \quad rogre \quad e \})$$

w .inv i i i yin n inv i n , n .progress i o - i ion in o ↦ q i yin o y ' on o . W i o .smodel i o- ll , w i i n o j no iv v i l [5] (o .smodel.pri = $\emptyset$). Mo ov , .inv i on inv i n o .smodel, n o ny o nvi on n o .smodel. F o , .inv o on n y .pub. Fo lly:

$$.\mathsf{inv}\ \mathbf{conf}\ .\mathsf{pub}\ \wedge\ .\mathsf{smodel}\|\ .\mathsf{env} \vdash \mathsf{sinv}\ .\mathsf{inv}$$

W will in ov lo nin o l o on o j o y l o wo on on . Fo x l , i i on , .pub no o ll ' li v i l , w i i j l o .smodel.pub.

I x i n o j , i on i no y x.contract. T l ion w n n o j n i on i n low:

Definition 7.1: Object-Contract Relation
L x n o j n = x.contract. T l ion w n x n i ollow :

1. x n .smodel v in . So x.pub = .pub n x.ops = .ops.
. T xi i :
   ( ) i on inv i n o $x\|x$.env n i i li .inv.
   ( ) .smodel i on i n ion o x. Mo i ly: $\vdash$ .smodel $\sqsubseteq$ x
   ( ) Fo v y i ion $\mapsto q$ in .progress: $x_{\lhd}\|x$.env $\vdash$ $\mapsto q$

T inv i n n ion ov i ll . . . . ˗ , . . , . . ˗ o x, n will no y x.concreteInv. No in x n v o o ion , n x.env = .env. So, ny o nvi on n o .smodel i l o o nvi on n o x. Al o no .inv i n inv i n o $x\|x$.env o j - on l ion i li xi n o inv i n o $x\|x$.env i lyin .inv. How v , .inv i no . ˗ . . ˗ inv i n o $x\|x$.env.

7.2 Inferring Object's Properties

F o o j - on l ion n o T o 5. , i ollow o o i ion o x wi ny o nvi on n Q will in in ll o o i i in R:

Corollary 7.2: Progress Commitment
L x n o j n = x.contract. L Q o nvi on n o x.

$$\frac{\mapsto q \in\ .\mathsf{progress}}{x_{\lhd}\|Q\ \ x.\mathsf{concreteInv} \vdash\ \ \mapsto q}$$

Any unless o y ov n wi o y o l in on i l o o y o l o j . Mo i ly:

[5] This is not a restriction, but more a matter of choice in defining how expressive a contract should be.

Theorem 7.3: Safety Commitment
L x n o j n $=x.$contract. L Q o nvi on n o x. T n:

$$\frac{.\mathsf{smodel} \| .\mathsf{env} \quad .\mathsf{inv} \vdash \quad \mathsf{unless}\ q}{x \| Q \quad x.\mathsf{concreteInv} \vdash \quad \mathsf{unless}\ q}$$

Contract Refinement. n o l n no ion o on n n . T i n l w n n o j o nno n n o j off in on . In i y y o n no o j w o on n . W will no wo o i i .

8 Application

An li ion on i o o o j n li n . An o j n i (i nno y li ion' nvi on n) o [6].

8.1 Semantical Model

W will n i lly o l n li ion $\mathcal{A}$ wi l o i y :

$$\overset{d}{=} \; (\,\mathsf{pubobjs} :: \{\ bjDe\ \} \quad \mathsf{priobjs} :: \{\ bjDe\ \} \quad \mathsf{clients} :: \{\ rog_{\text{UNITY}}\} \quad \mathsf{clientsinv} :: \ red \quad)$$

w $bjDe$ n o j l ion . W will no ion $\mathcal{A}$.objs o o o ll (li w ll iv) o j o $\mathcal{A}$. T i $\mathcal{A}$.clientsinv i in n o n inv i n o $\mathcal{A}$' o j wi in o ion o li n ' (iv) v i l . In i ion, v l on in , o x l on nin w ll- o n o o l; w will ow l , in w n o in o o on .

T o ll li v i l o $\mathcal{A}$, no y $\mathcal{A}$.pub, on i o li v i l o i li o j . $\mathcal{A}$' iv v i l , no y $\mathcal{A}$.pri, on i o nion o o iv v i l o i li o j , n o v i l o i iv o j n li n . T o ll $\mathcal{A}$' v i l i no y $\mathcal{A}$.var. T on o in y n li ion $\mathcal{A}$ i ollowin :

$$\mathcal{A}.\mathsf{prg} \overset{d}{=} (\|x : x \in \mathcal{A}.\mathsf{objs} : x) \;\|\; (\|Q : Q \in \mathcal{A}.\mathsf{clients} : Q)$$

[6] In CORBA a public object may be located outside the application itself. It may belong to and be controlled by another application, which may even be owned by a foreign organization. In such a setting a so-called *object broker* is used for searching the objects needed by an application and to facilitate the communication between the application and those foreign objects. This paper will however not concern itself with brokers.

T o l, o on , o w ol li ion, no y $\mathcal{A}.\mathsf{smodel}$, i nion o on o o j , o o wi li n :

Definition 8.1: Abstract Model of Application

$$\mathcal{A}.\mathsf{model} \overset{d}{=} ([\![x : x \in \mathcal{A}.\mathsf{objs} : x.\mathsf{contract.smodel}) \quad [\![\quad ([\![Q : Q \in \mathcal{A}.\mathsf{clients} : Q)$$

T wo nvi on n o n li ion n o ll y o i ll o i l ll o o ion o li o j :

Definition 8.2: Application's Abstract Environment

$$\mathcal{A}.\mathsf{env} \quad \overset{d}{=} \quad ([\![x \;\; X : x :: X \in \mathcal{A}.\mathsf{pubobjs} : x.\mathsf{env})$$

T no ion $\mathcal{A}.\mathsf{inv}$ o onj n ion o inv i n i y on in $\mathcal{A}$, n n y $\mathcal{A}.\mathsf{clientsinv}$. Si il ly, w n $\mathcal{A}.\mathsf{concreteInv}$:

Definition 8.3: Application's Invariants

$$\mathcal{A}.\mathsf{inv} \qquad\qquad \overset{d}{=} \quad (\bigwedge x : x \in \mathcal{A}.\mathsf{objs} : x.\mathsf{contract.inv}) \;\wedge\; \mathcal{A}.\mathsf{clientsinv}$$

$$\mathcal{A}.\mathsf{concreteInv} \quad \overset{d}{=} \quad (\bigwedge x : x \in \mathcal{A}.\mathsf{objs} : x.\mathsf{concreteInv}) \;\wedge\; \mathcal{A}.\mathsf{clientsinv}$$

8.2 Constraints

In o o ov l w n l o in o i o n li - ion, w n o i o o on in on i n i l o l. L $\mathcal{A}$ n li ion:

1. [CA1] o j in $\mathcal{A}$ i own ni n . So, o ny wo i in o j x n in $\mathcal{A}$, $x.\mathsf{var} \cap \;.\mathsf{var} = \emptyset$.

. [CA2] A li n n only in wi n o j o i o ion . F - o , li n n only o in l o ion in on o i . In o wo , wi o v y o j x in $\mathcal{A}$, li n o l o nvi on n o x.

3. [CA3] T only li in o ion li n o i o ll li v i l o o j in $\mathcal{A}$.

. [CA4] $\mathcal{A}.\mathsf{clientsinv}$ i in in y $\mathcal{A}$' o l. No i on- in i li $\mathcal{A}.\mathsf{clientsinv}$ n only i y v i l nown o li n .

8.3 Inferring an Application's Properties

A y o y ov n wi o o l o n li ion will x n o li ion i l , n ny o nvi on n .

Theorem 8.4: Safety by Abstract Model

$$\frac{\mathcal{A}.\mathsf{model} [\![\mathcal{A}.\mathsf{env} \;\;\; \mathcal{A}.\mathsf{inv} \;\vdash \qquad \mathsf{unless}\ q}{\mathcal{A}.\mathsf{prg} [\![\mathcal{A}.\mathsf{env} \;\;\; \mathcal{A}.\mathsf{concreteInv} \;\vdash \qquad \mathsf{unless}\ q}$$

Any o o y o i in on o ny o j in n li ion, will v y li ion, n y i nvi on n .

Theorem 8.5: Progress by Contract
L x n o j in $\mathcal{A}$ n $= x$.contract.

$$\frac{\mapsto q \ \in \ \text{.progress}}{\mathcal{A}\text{.prg}_{\lhd} \| \mathcal{A}\text{.env} \ \ \mathcal{A}\text{.concreteInv} \ \vdash \ \ \mapsto q}$$

T n x o w i o y li n n v y li ion n i nvi on n .

Theorem 8.6: Client Progress

$$\frac{\mathcal{A}\text{.model} \| \mathcal{A}\text{.env} \ \ \mathcal{A}\text{.inv} \ \vdash \ \ \mapsto q}{\text{.prg}_{\lhd} \| \mathcal{A}\text{.env} \ \mathcal{A}\text{.concreteInv} \ \vdash \ \ \mapsto q}$$

No o . n .6 fl w w n ion innin o ion . T on w n o j ' on , i. . o in o ion w in o l o o j , o x n iv i v i ion .

9 Example

on i VotingService li ion in Fi 1, i on li n superviseVoting, n o j SimpleVotingSystem llow o n

```
application votingService
  public  v :: SimpleVotingSystem ;
          d :: SimpleCalendar
  client superviseVoting
    private   closingDate :: Date = 01/01/2005 ;
              today       :: Date = 00/00/0000
    action    d.getDate(today)   []  today>=closingDate --> v.count()


contract SimpleCalendar
  smodel
    public   current :: Date
    init     true
    action   current := current + unittime

  operation  getDate(&today::Date) = do today:=current
  inv        current>=0
  progress   !D. true |--> current>=D
```

Fig. 1. An example of a simple application to do electronic voting and a contract of a simple calendar object

vo , n SimpleCalendar o j o o n . T l o ow on o SimpleCalendar.

Al o on o SimpleVotingSystem i no own, i in i wo o ion : vote n count. T i o n vo o o j . In o in vo oll in v i l votes. T on will lo vo in n o n vo . A v i l isOpen i o in i w o no vo in i ill o n. I i v i l n fli o l , n vote will no ny n w vo . iv n i i ion, on n x ollowin o y. Fo ny v l v:

$$\mathsf{v}\|\mathsf{x.env}\quad \mathsf{true}\ \vdash\ \neg\mathtt{v.isOpen} \wedge v \notin \mathtt{v.votes}\ \ \mathsf{unless}\ \ \mathsf{false} \qquad (1)$$

In o wo , on vo in o i lo , no n w vo will . No o y i n unless o y ov v‖x.env: i will v in o o i ion o v wi ny o nvi on n o v. S o w n o in i o y o w ol li ion, i. . w w n o v i y:

$$\mathsf{app.prg}\|\mathsf{app.env}\quad \vdash\ \neg\mathtt{v.isOpen} \wedge v \notin \mathtt{v.votes}\ \ \mathsf{unless}\ \ \mathsf{false} \qquad (\ )$$

w app = VotingService n app.concreteInv. By T o . i i - i n o ow ollowin :

$$\mathsf{app.model}\|\mathsf{app.env}\ \ j\ \vdash\ \neg\mathtt{v.isOpen} \wedge v \notin \mathtt{v.votes}\ \ \mathsf{unless}\ \ \mathsf{false} \qquad (3)$$

w j = app.inv. T only ion in app.model‖app.env n viol i o y i o ion vote, i n in o votes. How v , i n only o o i isOpen i true, w i i no in unless o y ov . H n ( ) i v li .

Now, o no ll in o in vo v li vo . T o j v will in- n lly l v li vo n in li validvotes. on i ollowin o i ion: i l y o n 100 v li , n v n lly v l o v i l accept will o true. T i in i vo in l o i iv ly.

$$\mathsf{app.prg}_{\triangleleft}\|\mathsf{app.env}\quad \vdash\ \mathtt{length\ v.validvotes}\quad \mathsf{100} \mapsto \mathtt{v.accept} \qquad (\ )$$

To ow ( ) i i i n (y n i ivi y o ↦) o ow ollowin o i wi o app.prg ◁‖ app.env n inv i n :

$$\mathsf{true}\ \mapsto\ \mathtt{d.current} \geq \mathtt{closingdate} \qquad (5)$$

$$\mathtt{d.current} \geq \mathtt{closingdate}\ \ \mapsto\ \ \mathtt{afterClosingDate} \qquad (6)$$

$$\mathtt{length\ v.validvotes}\quad \mathsf{100\ unless\ false} \qquad (\ )$$

$$\mathtt{length\ v.validvotes}\quad \mathsf{100}\ \wedge\ \mathtt{afterClosingDate}\ \mapsto\ \ \mathtt{v.accept} \qquad (\ )$$

w :

$$\mathtt{afterClosingDate}$$
$$=$$
$$\mathtt{d.current} \geq \mathtt{closingdate}\ \wedge\ \mathtt{today} \geq \mathtt{closingdate}$$

on i (5). Sin i o i li y SimpleCalendar o j o app, w T o .5 o li ion l v l o y o n o j l v l on . By o i o ow o y i i in progress- o o j d, w i i in (on in Fi 1).

on i (6). W x i o o li y li n . Uin T o .6, w n o ow o , wi o app.model◁‖app.env n inv i n *j*. T i n ov n ily, in i o i li y ion d.getDate(today) o li n . on n ly, i i y o ov ollowin ensures o y, w i i li ↦:

$$\begin{array}{l} \mathsf{app.model}_{\triangleleft}\|\mathsf{app.env} \quad \mathsf{j} \\ \quad \vdash \\ \quad \mathtt{d.current} \geq \mathtt{closingdate} \quad \mathsf{ensures} \quad \mathtt{afterClosingDate} \end{array} \tag{9}$$

T o o ov ( ) i i il o o ( ), n o ( ) i il o o (6).

10 Deploying an Application as an Object

Sin n li ion $\mathcal{A}$ i n i lly j o , i n loy n o j y w in i . Fo i w n o n n in , n o j li n iv v i l , ini i l on i ion, n o ion . vi n ly, w n only x o ll o o li o j o $\mathcal{A}$. T o , l nvi on n o w $\mathcal{A}$ will no v wo $\mathcal{A}$.env. on n ly, o i in in o o S ion .3, w i $\mathcal{A}$.env, will v y w in .

11 Related Work

V io o wo xi . T on in [13] i on Z n o on o li in i l ion w n o on n , i in o UML l i . In [1] wo i i li on Ho i l iion o in vio l o i . I i i l o l wi o on n v no in n l o o i own , n n nno , on i own, no o l o i . I in n l o , i yin o l o i wo l i o xili y v i l o o o j ’ io y. B oy’ wo [ ] il in i o y v i l o i lo i . T wo i i lly ilo o lin wi o on n yn oni wi nn l . I y ow v oo il i o on n x n in oion o o ion ll in . wo i o i l in l on x .

An i o n o o wo i o n o o i y (in o) vio o on o o nvi on n .

I n o o i onv ni n . n n n y ny o , . . [3, 10, 11, 1, 19, 1].

W vo o UNITY n lyin o y, o i i li i y n i xio i yl . I yi l n i l o l w i i i l n in i iv . T o UNITY n lyin o y o o o on n i n n o o y o in [10, 11, 1]. How v , w now, o wo i UNITY wo off in o l no ion o o j , on , n li ion. W l o w n o n ion S o Mi [1]. I i n x ll n o j o i n l n n lo i l y on o o UNITY. wo n n o on n o i n x n ion o S .

12 Conclusion

W v off o l wo o o o on n o o il i i li ion . T wo off o l no ion o o j , on , n li ion , n o l w o o o i ion lly in o l o i o n li ion.

T n lyin o y i in UNITY yl , w i o o o x l LTL i l x iv . How v , o UNITY' xio i yl , i yi l n i l o l w i i i l n in i iv .

A ll x i n wi i l vo in y [1] o ow on in o wo n ly n ly y ' o on n ' o l o i . U in l w ovi y wo w w l o o o i ion lly in in in o i o y .

W v o n o w no ion o n n , w i iv o o o v lo o i o on n ' il in on . I i o i l o on no ion o n n , o x l in [, 3], in x n in i o on n i i on will o x n iv . wo i i l o lin wi y in w i o on n yn oni y o ion ll . I i l i l o l wi in y , o wi y w o on n i i yn oni ion in o o ol .

W li v o wo i wo . F ivi i in l i li ion o l-wo l x l in o o o i l ili y.

References

1. M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.
2. M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.
3. R.J.R. Back and J. Von Wright. Refinement calculus, part I: Sequential nondeterministic programs. *Lecture Notes of Computer Science*, 430:42–66, 1989.
4. M. Broy. Multi-view modelling of software sytems. In Hung Dang Van and Zhiming Liu, editors, *Proceedings of the Workshop on Formal Aspects of Component Software (FACS)*, 2003. Also as UNU/IIST Report no. 284, available on-line at `www.iist.unu.edu/newrh/III/1/page.html`.

5. K. Chandy and M. Charpentier. An experiment in program composition and proof. *Formal Methods in System Design*, 20(1):7–21, 2002.
6. K.M. Chandy and J. Misra. *Parallel Program Design – A Foundation.* Addison-Wesley Publishing Company, Inc., 1988.
7. K.M. Chandy and B.A. Sanders. Reasoning about program composition. Technical Report 96-035, University of Florida, 1996.
8. K.M. Chandy and B.A. Sanders. Reasoning about program composition. Draft. Presently available via: `www.cise.ufl.edu/∼sanders/pubs`, 2000.
9. M. Charpentier and K. Chandy. Theorems about composition. *Lecture Notes of Computer Science*, 1837:167–186, 2000.
10. P. Collette. Composition of assumption-commitment specifications in a UNITY style. *Science of Computer Programming*, 23:107–125, December 1994.
11. P. Collette and E. Knapp. Logical foundations for compositional verification and development of concurrent programs in UNITY. *Lecture Notes of Computer Science*, 936:353 – 367, 1995.
12. He Jifeng, Lui Zhiming, and Li Xiaoshan. A contract-oriented approach to CBP. In Hung Dang Van and Zhiming Liu, editors, *Proceedings of the Workshop on Formal Aspects of Component Software (FACS)*, 2003. Also as UNU/IIST Report no. 284, available on-line at `www.iist.unu.edu/newrh/III/1/page.html`.
13. Soon-Kyeong Kim and David Carrington. A formal mapping between UML models and Object-Z specifications. *Lecture Notes in Computer Science*, 1878:2–??, 2000.
14. J. Misra. *A Discipline of Multiprogramming.* Springer-Verlag, 2001.
15. I. S. W. B. Prasetya. *Mechanically Supported Design of Self-stabilizing Algorithms.* PhD thesis, Inst. of Information and Comp. Science, Utrecht Univ., 1995. Download: `www.cs.uu.nl/library/docs/theses.html`.
16. I.S.W.B. Prasetya. Error in the UNITY substitution rule for subscripted operators. *Formal Aspects of Computing*, 6:466–470, 1994.
17. I.S.W.B. Prasetya, T.E.J. Vos, A. Azurat, and S.D.. Swierstra. A unity-based framework towards component based systems. Technical Report UU-CS-2003-043, Inst. of Information and Comp. Science, Utrecht Univ., 2003. Download: `www.cs.uu.nl/staff/wishnu.html`.
18. B.A. Sanders. Eliminating the substitution axiom from UNITY logic. *Formal Aspects of Computing*, 3(2):189–205, 1991.
19. N. Shankar. Lazy compositional verification. *Lecture Notes of Computer Science*, 1536:541–564, 1999.
20. C. Szyperski. *Component Software, Beyond Object-Oriented Programming.* Addison-Wesley, 1998.
21. R.T. Udink. *Program Refinement in UNITY-like Environments.* PhD thesis, Inst. of Information and Computer Sci., Utrecht University, 1995. Downloadable from `www.cs.uu.nl`.
22. T.E.J. Vos. *UNITY in Diversity: A Stratified Approach to the Verification of Distributed Algorithms.* PhD thesis, Inst. of Information and Computer Sci., Utrecht University, 2000. Download: `www.cs.uu.nl`.
23. T.E.J. Vos, S.D. Swierstra, and I.S.W.B Prasetya. Yet another program refinement relation. In *International Workshop on Refinement of Critical Systems: Methods, Tools and Experience*, 2002.

Searching for a Black Hole in Tree Networks

J y owi [1,*], D i Kow l i[2,3,**],
i i M o [1,***], n An j P l [1,†]

[1] Département d'informatique, Université du Québec en Outaouais,
Hull, Québec J8X 3X7, Canada
{jurek, evripidi, pelc}@uqo.ca

[2] Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
darek@mpi-sb.mpg.de

[3] Instytut Informatyki, Uniwersytet Warszawski,
Banacha 2, 02-097 Warszawa, Poland

Abstract. A black hole is a highly harmful stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We consider the task of locating a black hole in a (partially) synchronous tree network, assuming an upper bound on the time of any edge traversal by an agent. The minimum number of agents capable to identify a black hole is two. For a given tree and given starting node we are interested in the fastest possible black hole search by two agents. For arbitrary trees we give a 5/3-approximation algorithm for this problem. We give optimal black hole search algorithms for two "extreme" classes of trees: the class of lines and the class of trees in which any internal node (including the root which is the starting node) has at least 2 children.

Keywords: algorithm, black hole, mobile agent, tree.

1 Introduction

1.1 The Background and the Problem

S i y o o il n wo in in n wo nvi on n i n i o n i w i iv n ly owin n ion. P o in n o " o ", i. ., l i o in no o n wo , o l o n o in o , i. ., no o n wo , o n n ' [, 9]. V io o o o in o il n in li io o v n i , . ., in [5, 6, , , 9, 10].

[*] Research supported in part by NSERC grant.

[**] Research supported in part by grants from KBN (4T11C04425) and UE DELIS.

[***] This work was done during this author's stay at the Research Chair in Distributed Computing of the Université du Québec en Outaouais, as a postdoctoral fellow.

[†] Research supported in part by NSERC grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 67–80, 2005.

In i w on i o il o o i l ly l n , ll
[1, ,3,]. A l ol i ion y o i in in no o
n wo n oyin ll o il n vi i in no , wi o l vin ny
. Sin n nno v n in nni il on y vi i l ol ,
only w y o o ion in o i i n i yin o il no
n voi in vi i in i . H n w lin wi i o lo in
l ol : in i o on l ol in n wo , l
on vivin n n lo ion o l ol i i xi , o n w
i no l ol , o wi . T only w y o lo l ol i
o vi i i y l on n , n , o v in [ ], l wo n
n y o on o o lo l ol n viv . T o o
w n o n i ini o i l o o ,
i. ., , n y o no , nown o .

In [1, ,3,] i o i n l ol w x n iv ly i in
ny y o n wo . T n lyin ion in w
n wo i o lly yn ono , i. ., w il v y v l y o il
n ni i , i no o n on i i . In i in
i w o v , in o o olv o l , n wo -
onn , in i l l ol i in i l in . T i i ,
in yn ono n wo i i i o i l o i in i l ol o
“ low”lin in i n o i . H n only w y o lo l ol i o vi i
ll o no n l n y . (In i l , i i i o i l o
n w ion o w l ol lly xi in n wo , n
[1, ,3,] wo n ion i x ly on l ol n
w o lo i .)

To lly yn ono n wo ly o in i . n (o i ly
l) o n on i o v in ny y n n n
li . H n i i in in o y l ol in i lly
yn ono n wo . Wi o lo o n li y, i o n on
v l i n no li o 1 w i yi l ollowin ni ion o
i o l ol : i i xi i n y
, i. . i n wo - lo ion o l ol (o w n i
o no xi in n wo), in ll v l i 1.
i lly yn ono n io i n o o l
o in o l ol . Now i i o i l o i -o ni
o lo l ol in ny , wi only wo n , ollow : n
o lon o nnin . I y no v, on n
o o j n no n n , w il o n w i v. I
i n no n , o on viv n now
lo ion o l ol . wi , j n no i nown o n
o n n ov o i . T i i in v i n o i
in [ ] o inin i wi i -o ni l ol
i l in ny . H n i i now no i ili y i
i n y o l ol , n n i vo o i o l .

Sin o nyn wo , l ol n on in only o i nnin , olvin o l o l ol on n l . H n in i w i n ion o l ol in n wo in wo n , n o o l i o o li i in ini i . l ly, in ny , o i n l ol n o o in in nnin o , n n li ion o o o l o i y n wo in n i o n n in in o n i .

T i o l ol o l i in i o i o l xi y o l o i o in . W il w n ov o iv n in on i in o n wo n in no , n i in l o n o i ni n y wo n , on i i o o in y l o i . In o wo , i o i i o w l in n i o l xi y o l o i i i o in in .

on in l ol o i y n o o o ivi l. In i l , ollowin o l in o n. Do xi olyno i l i l o i w i , iv n n in no in , o l ol wo in in o o i l i o i in ? N v l , w ow o o l o n iv 5/3- oxi ion l o i o n l .

1.2 Our Results

Fo i y w iv 5/3- oxi ion l o i o l ol o l . Mo i ly, iv n n in no in , o l o i o l ol w o i i o 5/3 o o o i l i o i in .

W iv o i l l ol l o i o wo “ x ” l o : l o lin n l o in w i ny in n l no (in l in oo w i i in no) l il n. Mo i ly, o v y in in iv l l o i o l ol w o i i o o i l o i in .

All o l o i wo in i lin in i o in .

2 Model and Terminology

W on i oo no w i i in no o o n , n i o (i no l ol). No ion o il , n , n n n n o , n wi o i oo . A n v i in l l . T y n o ni only w n y (n no , . ., y l vin no). W i o on l ol in n wo . T i i no w i oy ny n vi i in i . A l ol ( ) o in ( ) i i o n o v l (ov) o o wo n , wi ollowin o i .

– ov on i ni .
– U on o l ion o i l on vivin n , i. ., n n no vi i l ol , n i n i now lo ion o l ol o now i no l ol in . T vivin n n o .

T i o l ol i n o i ni n il o l ion o , in wo - lo ion o l ol (o i n , w i v i wo). I i y o wo o iv n o w n i no l ol in n wo o w n l ol i l nvi i no , o yi l in i . A i ll o iv n in i i i i o o i l o i in .

Fo ny o w n ollowin :

– , i no n ov y lon i (ini i l o v y),
– , i i inin n now i no l ol in i n o i , o y now w i n o i l ol .

No in w n in , n y n i n nown no x-lo . T i i w n n n nown n j v y n n .

Any BHS- v ollowin o y: ni n o , l on n y liv n ll x lo (i o on l ol , o on l ol n o n , ll x lo). T o BHS- i o x lo . A innin o BHS- x lo i o y i y. W y o in no *v* w n n no *v* n x n in o ion w i x lo i o y. No *v* i ll

In ny o BHS- , n n n v n o w i in no . Al o wo n n . I in o , n x lo i o y i n x lo i o y . in . T n o o BHS- w n wo on iv in i ll

3 Preliminary Results

Lemma 1.

H n in BHS- , n n x lo only in ollowin w y: n n v i n n in i l . W i o o no (in l n v ni in l ol) o x lo .

Lemma 2. on

T o n n nown o l x lo in n x only i i i j n o x lo i o y. T x lo i o y in only l in oin .

Lemma 3.

W n o in w i x ly on i x lo . Si il ly, w n o in w i x ly wo x lo . In vi w o L 3, v y i i 1- o - .

Lemma 4. v $v =$ v -1

H n n n w i v n n x lo n o x-lo i o y in o o o o in oin . A o oll y o L 1, n i ny o BHS- x lo i o y i onn .

A no i ll o x lo i o y i i i in i n o o n x lo n o n n x lo .

A w y o x lo in x ly on in i ollowin : on o n w l o x lo i o y o i li i , w il o n w l o x lo i o y o , v n n nown n n o . I w o n li i o x lo i o y n (u) i n n nown ow no v, w n ollowin o :

probe(v)**:** on n v (u) (w i i ow no v) n n o no o o n w o w i . I y o no + n l ol n o n .

W l o n o wo n o l ollow o x lo wo n w in . S o wo n i no m . L $_1 \ldots _i$ li i o x lo i o y . o n nown w i o l x lo in ollowin o in i n o no o $\{ _1 \ldots _i\}$. L wo l n nown o x lo ion $(k \; _k)$ n $(\; _l)$, $_k \; _l \in \{ _1 \ldots _i\}$ (o i ly $_k = \; _l$). W no m lon o k . T ni ion o o i ollowin :

split(k, **):** n o n v o no m o no k n n ow no $_l$. T o v o no m o no n n ow no $_k$. L d (k) no n o in o no k o no . I y o no $+ d$ (k) n l ol n o n .

4 Black Hole Search in a Line

In i ion w on n o i l l ol l o i o lin , wi lin i o l xi y. A lin i $= (\quad)$, w $= \{0 \ldots \quad \}$

n $= \{[\quad + 1] : \quad = 0\ 1 \ldots \quad - 1\}$. 0 n ll n oin o lin . T in no i no y , w il n b no i n w n n n oin o lin , wi $\leq b$, n $+ b =$. W b 0, o wi lin on i o in l no . W ll i ion o ow lo n oin n o i ion.

Theorem 1.

- $- \quad - \quad = 0$
- $\sum_{i=1}^{a} \quad 1 \leq \quad = b \leq 5$
- $- \quad - 6 \quad = 1 \quad b$
- $- \quad - 10 \quad = \quad b \quad = 3 \quad b$
- $- \quad - \quad = \quad b \quad = 5 \quad b \quad \geq 6$

W will now iv n o i l l o i o olv l ol o l o lin (i. . n l o i w i o BHS- o ny lin). S o o n i no *m*. T l o i o *robe*, n ollowin on :

```
– walk(k): o n o 1 ow no k.
– walk-and-probe(v):
  while o i ion o n i no j n o no v do
        w l (v);
          o (v)
– return( ):
  repeat w l ( ) until ll inin n
```

T i -l v l i ion o Al o i Lin i ollowin :

- **case** $= 0$**:** wo n x lo lin y o in l o n n
- **case** $1 \leq \quad = b \leq 5$**:** wo n x lo lin y li
- **case** $= 1 \quad b$**:** wo n o li n n x lo o lin y o in l n n
- **case** $= \quad b$**:** wo n o li , n x lo ll l o x on y o in , n n lly x lo l wo y li
- **case** $3 \leq \quad b$ **or** ≥ 5**:** wo n o wo li , n x lo ll l o x on y o in . T y x lo l l o wi n i o y li n n lly x lo inin (i ny) w i i o y o in n n

T i o l ion o l o i i iv n Al o i 1. T i o l xi y o l o i i lin .

Theorem 2.

T oo o l o i ion o i o l o n will in ll v ion o .

Algorithm 1. Al o i Lin

```
case $a = 0$
        probe(0);
        walk-and-probe(0);
case $1 \le a = b \le 5$
        for $i := 1$ to $a$
            split($s - i$, $s + i$);
case $a = 1 < b$
        split($s - 1$, $s + 1$);
        walk-and-probe(0);
case $a = 2 < b$
        split($s - 1$, $s + 1$);
        walk-and-probe(1);
        split(0, $s + 2$);
case $a = 3 < b$
        split($s - 1$, $s + 1$);
        split($s - 2$, $s + 2$);
        walk($s - 1$);
        walk-and-probe(1);
        split(0, $s + 3$);
case $4 \le a < b$ OR $a \ge 6$
        split($s - 1$, $s + 1$);
        split($s - 2$, $s + 2$);
        walk($s - 1$);
        walk-and-probe(1);
        split(0, $s + 3$);
        walk($s + 2$);
        walk-and-probe(n);
return($s$)
```

5 Black Hole Search in a Tree

In i ion w y o l o l ol in . on i oo in no . I e i n , $e = (u\ v)$ n v i il o u. L $e = (u\ v)$ n o . on i ollowin olo in w i i ion o o . T i i ion will in n ly i o o l o i .

- i n olo o e i no v l wo n n ,
- i n n olo o e i v i l n x ly on o ollowin ol : $u =$ o (u) i (w i n o u),
- i n l olo o e i i non o ov o i

L $e = (u\ v)$ n $e' = (v\ z)$ wo l v i ni il o u n z i l n ni il o v. W ll o wo . T o ll n o l wi no u i ll .

Lemma 5. 6 6

By L 1 ny o v i y on n o o x lo . In i l n n v l . on i $e = (u\ v)$. L n o n n o no v. In vi w o L 1 n , i in ny x lo ion e i v lw y y only on n n l $\geq$ i ion l v l i (n n o v e wo i o v y n n o v). I i l on x lo ion o e w i v y o n n l i ion l v l o e i o x lo ion o wi no v (o n v e n n). T o l ini n o v l i 6.

A n o l n v in ollowin w y . v l i o x lo ion o o n . I in ny x lo ion o , i i v lw y y only on n n l i ion l v l on i n i . I i l on x lo ion o w n i i v y o n n l 6 i ion l v l on i n i (o n v , n on o x lo low n n lly y n). T o o l ini n o v l on n i 6.

Lemma 6. 3 1 3b b

5.1 An Optimal Algorithm for a Family of Trees

on i ily $\mathcal{T}$ o oo wi ollowin o y: ny in n l no o in $\mathcal{T}$ (in l in oo) l il n. T in $\mathcal{T}$ will ll .

L y wi oo n l u n in n l no o . T $v = H(u)$ o u i n il v o u (v) oo v (w i i l o y) xi i on ll oo il n o u. T il $v' =$ (u) o u i n il v' o u (v') oo v' ini i on ll oo in il o u. Ti o n i ily. No i $H(u)$ n (u) n o o ll no u in lin i .

T i -l v l i ion o Al o i B y-T i ollowin . L m in oin o wo n (ini i lly $m =$).

– x lo ny i o n nown $(m\ x)$, $(m$ $)$ wi no m y x in o $(x$ $)$, l vin $(m$ $(m))$ l .

– I i on n nown wi no m (w i $(m$ $(m)))$ x lo i o wi no n nown (i ny) in in o . I $(m$ $(m))$ i l n nown in , x lo i y x in o *robe*((m)).

– I ll wi no m x lo , x lo i il ly o ny n nown in i n o il n o m n o n o o m.

B low w iv i o l ion o l o i .

Algorithm. Bushy-Tree

```
  special-explore($s$)
```

Procedure special-explore(v)

```
  for every pair of unknown edges $(v, x), (v, y)$ with upper node $v$ do
    split($x, y$), so that edge $(v, L(v))$ is explored last
  end for
  if every edge is explored then
    repeat walk($s$) until (all remaining agents are at $s$)
  else
    case 1: every edge incident to $v$ has been explored
            $next$ := relocate($v$);
            special-explore($next$);
    case 2: there is an unknown edge $(v, z)$ incident to $v$
    (* must be $z = L(v)$ *)
            explore-only-child($v, next$);
            special-explore($next$);
  end if
```

F n ion lo (v) in n no v w o n i n n n w lo ion o wo n . I i n n nown in i n o il o v n n o o il . wi wo n o o n o v.

Function relocate(v)

```
  case 1.1: ∃ an unknown edge incident to $w \in children(v)$
          walk($w$);
          relocate := $w$
  case 1.2: every edge incident to any child of $v$ is explored
          let $t$ be the parent of $v$;
          walk($t$);
          relocate := $t$
```

P o x lo -only- il $(v$ *ex*) in n no v w o n i n n n w in oin x lo ion o $(v$ $(v))$. T i ion o o i ollowin :

- I i n n nown in i n o il w o v, $w \neq$ (v), n n x lo $(w$ $H(w))$ o wi $(v$ $(v))$ y $(H(w)$ $(v))$. T n w in oin i w.
- I v y in i n o ny il w o v, iff n o (v), i x lo n $(v$ $(v))$ i no l n nown in , n n n o o v wi n nown w o no i n n o ; n x lo $(D($ $)$ $H(D($ $)))$ (w $D($ $)$ i lo - n n o wi in i n n nown), o wi $(v$ $(v))$, y $(H(D($ $))$ $(v))$; n w in oin i $D($ $)$.
- I $(v$ $(v))$ i l n nown in n x lo i y llin *robe*((v)); n w in oin i v.

Procedure explore-only-child(v, *next*)

```
case 2.1: there is an unknown edge incident to $w \in children(v)$, $w \neq L(v)$
        split($L(v)$,$H(w)$);
        $next := w$
case 2.2: every edge incident to any $w \in children(v)$, $w \neq L(v)$ is explored
(* $L(v)$ must be a leaf *)
        case 2.2.1: there are at least 2 unknown edges left
            let $a$ be the deepest ancestor of $v$ such that:
            $D(a)$ := the closest descendant of $a$ with incident unknown edges;
            split($H(D(a))$, $L(v)$);
            $next := D(a)$
        case 2.2.2: there is only 1 unknown edge left
            probe($L(v)$);
            $next := v$
```

No i ll o (x o i ly l on i n o i o) x lo y llin o li . v in ny y , only n . By ni ion, in v y $e_r = (u_r$ $v_r)$, no v_r l wo il n n v y l o i n n oin o n $e_g = (u_g$ $v_g)$. Al o u_g l wo il n.

Sin ll v l $H(u)$ n (u) n o in lin i i i y o i o l xi y o Al o i B y-T i lin .

Theorem 3.

Sketch of the proof: T o y Al o i B y-T v ny 6 i n ny n i . Mo ov v y i - (i. . wo n v in ll l), x o i ly l (in w n n o i o), n no n w i in ny - .

5.2 An Approximation Algorithm for Trees

In i ion w iv n oxi ion l o i wi io $\frac{5}{3}$ o l ol o l , wo in o i y (i. . n l o i w i o BHS- w o i i o 5/3 o o o i l i , o v y in).

T i -l v l i ion o Al o i T i ollowin . L v in oin o wo n (ini i lly $v =$); wi no v x lo y llin o li n il i ll x lo o i o on inin n nown in i n o v, w i i x lo y llin o o ; i i o ny il o v. T i o l ion o l o i i iv n low. A o o li n o i n ion lo n in vio ion. T i - o l xi y o Al o i T i lin .

```
Algorithm. Tree
  explore($s$)
```

```
Procedure explore($v$)
  for every pair of unknown edges $(v, x), (v, y)$ incident to $v$ do
    split($x, y$);
  end for
  if there is only one remaining unknown edge $(v, z)$ incident to $v$ then
    probe($z$);
  end if
  if every edge is explored then
    repeat walk($s$) until both agents are at $s$
  else
    $next$ := relocate($v$);
    explore($next$)
  end if
```

Lemma 7. u d u β u, ρ u u $d +$ $\beta +$ ρ d $d + 1 +$ $\beta +$ ρ d

Theorem 4. $\frac{5}{3}$

I on i o in l , n io i on . wi , o k no u_1 u_2 ... u_k $\forall u_i \exists v_j$ $(e_{ij} = (u_i$ $v_j))$ i , n o n l in n o l . In ny , $\forall u_i \neq$ u_i l wo n n , n $(u_i'$ $u_i)$ i . T l $k-1$ in . L d_i: $= 1 \ldots k$ own o u_i. S o d_i: $= 1 \ldots$ i o n d_i: $=$ $+1 \ldots k$ i v n. L β_i n o n o l wi no u_i, ρ_i n o wi no u_i n $_i$ n o n wi no u_i. W v $d_i = \beta_i + \rho_i +$ $_i$.

A o in o L 6, ny BHS- n l $3\beta_i + 3\rho_i +$ $_i$ i ni on v l o ll , n n n o l wi no u_i. H n in vi w o L io w n i o o n o i l i o :

$$\frac{\sum_{i=1}^{l}(d_i + 1 + \ \beta_i + \ \rho_i) + \sum_{i=l+1}^{k}(d_i + \ \beta_i + \ \rho_i)}{\sum_{i=1}^{k}(3\beta_i + 3\rho_i + \ _i)} = \frac{\sum_{i=1}^{k}(5\beta_i + 3\rho_i + \ _i) +}{\sum_{i=1}^{k}(3\beta_i + 3\rho_i + \ _i)}$$

T ov io i $\leq \frac{5}{3}$ w n $3 \leq 6\sum_{i=1}^{k}\rho_i +$ $\sum_{i=1}^{k}$ $_i$. Sin $\sum_{i=1}^{k}\rho_i \geq k-1$, i io i low o l o $\frac{5}{3}$ w n

$$6(k-1) + \sum_{i=1}^{k} \ _i \geq 3 \tag{1}$$

I $k-1 \geq$ (i. . i l on no o v n own) n in li y (1) i .

I $k-1$ i n $= k$. T i i i ion w n v y v x u_i no low . I $k \geq$, in li y (1) ill ol . I $k=1$ n i no ($u_1 =$). A lon l wo n , in li y (1) i . wi on o ollowin ol :

- T on i o lo o β_1 n o l w β_1 i v n, n on n . In i o l n o in i o . H n , in ny BHS- l on x lo in 1- . W ov ny BHS- o n l $3\beta_1 +$ i ni o ll v l . A o in o L 5 o l n o v l n i l $6\beta_1 +$. A l o v l on in 1- n i l i ni . T o i n in i i l $\frac{6\beta_1}{2} +$ $= 3\beta_1 +$.

  A o in o L , o y Al o i T $d_1 +$ $1 +$ $\beta_1 = 5\beta_1 +$ i ni . T io i o $\frac{5\beta_1+2}{3\beta_1+2} \leq \frac{5}{3}$.
- T on i o lo o β_1 n o l w β_1 i o . I $\beta_1 = 1$ n io i on . wi w ov ny BHS- o n in i l $3\beta_1 + 1$ i ni o ll v l .
  - I i n in n w i n v y o n in n o l n o v l in n i . T o in vi w o L 5, o l n o v l i l $6(\beta_1 - 1) +$ n i n i l $\frac{6\beta_1+2}{2} = 3\beta_1 + 1$.

- wi , i i l on n x lo in 1- n o l n o v l on in i o $6\beta_1 -$ y L 5, w il v l on in 1- w i i i ni . T o i n i l $\frac{6\beta_1-2}{2} +$ $= 3\beta_1 + 1$.
- T inin i v y i x lo in - n i no w i n v y o n in . Sin n o in n i o , - ϕ in w i n o n i x lo o wi low o no n . T i n o i i l i ni in o n nno v . In vi w o L 5 o l n o v l in v y x ϕ i l $6(\beta_1 -$ $) +$ $+$ (i n on w i only v l on n n on w i only v l on). H n i n in i i l $\frac{6\beta_1-6}{2} +$ $= 3\beta_1 + 1$.

A o in o L , i o o y Al o i T i $d_1 + 1 +$ $\beta_1 = 5\beta_1 + 1$ i ni . T in ll io i o $\frac{5\beta_1+1}{3\beta_1+1} \leq \frac{5}{3}$.

No i xi ily o in w i oxi ion io i v y Al o i T i x ly 5/3. T i ily in l ll w i on i o n v n n β o n o l . A o in o L , i o o y Al o i T i $\beta +$ $\beta = 5\beta$ o , w il BHS- o i i x ly 3β i ni (o x l , ll x lo wo y wo y llin o li n n ll low x lo in w y).

6 Conclusion

W n l o i o l ol o l on . Fo i y w v 5/3- oxi ion l o i , n o wo l o (lin n ll o w o in n l no v l il n) w v o i l l o i , i. ., o o on in o o i l l ol o ny in in l . T i o l xi y o ll o l o i i lin in i o in .

I in o n i xi olyno i l i l o i o on l ol o n i y . Mo n lly, w o no now i o l i olyno i l o i y . W onj n w o l ion i n iv . H n i in in o n oo oxi ion l o i o l ol o l on i y . I o l no ivi l , o in lon ny nnin o in w l - n - o n nin o in no , ovi - oxi ion l o i o i o l .

References

1. S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, N. Santoro, Black hole search by mobile agents in hypercubes and related networks, Proc. of Symposium on Principles of Distributed Systems (OPODIS 2002), 171-182.
2. S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile agents searching for a black hole in an anonymous ring, Proc. of 15th International Symposium on Distributed Computing, (DISC 2001), 166-179.
3. S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Searching for a black hole in arbitrary networks: Optimal Mobile Agents Protocols, Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), 153-161.
4. S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Multiple agents rendezvous on a ring in spite of a black hole, Proc. Symposium on Principles of Distributed Systems (OPODIS 2003).
5. F. Hohl, Time limited black box security: Protecting mobile agents from malicious hosts, Proc. Conf. on Mobile Agent Security (1998), LNCS 1419, 92-113.
6. F. Hohl, A framework to protect mobile agents by using reference states, Proc. 20th Int. Conf. on Distributed Computing Systems (ICDCS 2000), 410-417.
7. S. Ng, K. Cheung, Protecting mobile agents against malicious hosts by intention of spreading, Proc. Int. Conf. on Parallel and Distributed Processing and Applications (PDPTA'99), 725-729.
8. T. Sander, C.F. Tschudin, Protecting mobile agents against malicious hosts, Proc. Conf. on Mobile Agent Security (1998), LNCS 1419, 44-60.
9. K. Schelderup, J. Ines, Mobile agent security – issues and directions, Proc. 6th Int. Conf. on Intelligence and Services in Networks, LNCS 1597 (1999), 155-167.
10. J. Vitek, G. Castagna, Mobile computations and hostile hosts, in: Mobile Objects, D. Tsichritzis, Ed., University of Geneva, 1999, 241-261.

Fast Localized Delaunay Triangulation*

Fili A jo n L ı o i

Universidade de Lisboa, Departamento de Informática,
Faculdade de Ciências, Campo Grande,
Edifício C6, 1749-016 Lisboa, Portugal
{filipius, ler}@di.fc.ul.pt

Abstract. A localized Delaunay triangulation owns the following interesting properties in a wireless *ad hoc* setting: it can be built with localized information, the communication cost imposed by control information is limited and it supports geographical routing algorithms that offer guaranteed convergence. This paper presents a localized algorithm that builds a graph called planar localized Delaunay triangulation, *PLDel*, known to be a good spanner of the unit disk graph, *UDG*. Unlike previous work, our algorithm builds *PLDel* in a single communication step, maintaining a communication cost of $O(n \log n)$, which is within a constant of the optimum. This represents a significant practical improvement over previous algorithms with similar theoretical bounds. Furthermore, the small cost of our algorithm makes feasible to use *PLDel* in real systems, instead of the Gabriel or the Relative Neighborhood graphs, which are not good spanners of *UDG*.

Keywords: Wireless *ad hoc* networks, Location-based routing schemes, Delaunay triangulation.

1 Introduction

Wi l n wo n wo w no o ni wi n i o wi in o n in wi l lin . No o wi l n wo y i lly o on i n v l iv ly w o y n n y o . I i o ly i o n o ly on o in wi ll n o ni ion ov . T i i n n y o in , w no only in in in o ion o o no wi in li i n i o oo . n o n , o o i n y, o in o l , i. ., ny o n y o l o i lon n o . How v , K n ov no lo li n - o i iv [9]. S ill, lo li o in n n onv n , w il i vin o i iv l n in o .

* This work was partially supported by LaSIGE and by the FCT project INDIQoS POSI/CHS/41473/2001 via POSI and FEDER funds.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 81–93, 2005.

n w y o i vin o i iv o in i o il lo l D l n y T i- n l ion [ ]. Un o n ly, il in i no vi l ol ion o o in o l in wi l n wo , :) y lon n o ni ion n ;) i nno il lo lly n o , o - ni ion o i oo i . H n , o o i o il l n (i. ., wi o in ion o) n o i l (( )), in D l n y i n l ion , in lo li ion. T oin o vin n i o o in l o i i v oo o o n o n , w il l n i y i n y o n onv n .

In li , v l l o i il D l n y i n l- ion o o in o , . ., [1 , 13, 6, 10]. T l o i in [13] il - o lo l D l n y i n l ion only in l o o wi in o ni ion n o no ; [1], [6] n [10] il n , wi lo l o ni ion o o (lo), (^{2}) n (^{2}), iv ly. W il [6] n [10] no o i l, [1] involv o ni ion o il n l , w i y o i i iv in i l y . H n , in i , w i ov on wo o Li . [1], y n in n l o i i on i ly i l n y il (*De*), wi y o i o ni ion o , wi j in l o ni ion (w n o ni ion io i o n in n n ivin on o o w i no lly l).

T o , o l o i i w ll i o wi l nvi on n o ol- lowin on :) i i v y i n i i j on o ni ion ;) i i li l o yn i n yn ono in (S ion 6);) i i lo li , only i in no o iv in o ion o y i n i o , i in o ni ion o wi in ll on n o o i (in on o (lo) i in n -no n wo i n y no); *v*) i i no o o only on n n - o n i o in v ; *v*) n on in o vin l n i y, i il wi oo n i y (S ion 5).

T o i ollow . Fo l - on in n w ovi o ov vi w o n y on in S ion . In S ion 3 w ovi v y on l wo on wi l n wo n D l n y i n l ion . In S ion w i o l o i n ov i o n . In S ion 5, w x i n lly v l o l o i . T li ion o l o i in yn i in i i in S ion 6. Fin lly, S ion on l .

2 Preliminaries

W no n in i own o i ion n o i ion o i n i o . iv n o no in wo i n ion l , w o l wi l n wo ni i , *DG*( ), w i i o i o ll no n ll onn in i o no o w o i n i o 1, i. ., in i o l, wo no n *B* i n i o (o i ly) i n only i || B|| $\leq$ 1. No n B k- o n i o i y n o in k o w o . T o o i , w will ollowin no ion: i n l n y no , B n i n $\triangle$ B ; n n l (π) w n B n n i in n ly n $\angle B$ o $\angle$ B; i l w o i i n y wo no n B i n $d($ $B)$; i i l n y no , B n i n $\bigcirc$ B .

T (GG) i o i o ll B $d($ $B)$ o no on in ny o no o . T o GG ll . T (R G) i o i o ll B i no no o w i || || || B|| n ||B || || B|| (i. ., no , nno i l n o ly lo o n B n n B o o). I o l no R G i o GG. T (D) o no , n De ( ), i o i yin " y i l " o y: B lon o i n l ion i n only i i i l on inin n B, no on inin ny o no . An i o n o y o De ( ) will o o , i i l o i n l o no on in ny no o . Un DG o l, o l D l n y i n l ion y no xi , o y lon n 1 n o , w o De ( ) = De ( ) $\cap$ DG( ) in .

In i w will ni ion o o in [1] o k , De $^{(k)}$( ). De $^{(k)}$( ) i o i o wo y o (no lon n 1):) ll o GG; n) o ll i n l B o w i no no in i $\bigcirc$ B l y , B o in k o w o . Li [1] ov De $^{(k)}$( ) i l n o $k \geq$, y in o $k = 1$. De ( ) [1 , 10] i n l n o i o ll i n l o De $^{(1)}$( ), in in i n l o no lon o De $^{(2)}$( ). Mo ov , Li [1] ov De ( ) i ($\sqrt{3}\pi$)/9- nn o DG( ) n De $^{(k)}$( ) $\supseteq$ De ( ). H n De ( ) n De $^{(k)}$( ), o ll k, l o ($\sqrt{3}\pi$)/9- nn o DG( ).

3 Related Work

In li , w n n v l l o i il D l n y i n l ion , . . [11, , 16]. i l in o l o i llow D l n y i n l ion o o in n in n l w y [1, 1], n w no iv l o no o o ion o n i i n l ion.

In [1], Li o o n l o i o il o l non-lo li D l n y i n l ion v n ov l y n wo on o o IP. How v , i li ion o i l o i o o o l x in o wi l nvi on n i no o i l in D l n y n i o y no l o o ni i i i n i n 1. In on x o wi l n - wo , o i o in l o i li y n o v iv wi n ion in li [, 1]: l o i o yl n y i v x ll n o n in n o v n in wi ( ) , on D l n y i n l ion , own y x i n l l o [1 ,10]. Un- o n ly, l o i no n o onv . W n y il, on o l n iv o in l o i , l o i on i - n l w i n o onv lon i l- n . T i o ion o y o i o in w o o in [3] n l x lo in o o ol ll y-P i S l o in (PS) [ ]. To x l n o non- l n , *R G*, *GG* o v i ion o D l n y i n l ion [19, 5, , 13], y . A n i y i i o n o i v oo o in o n , ny o v o on in in i , o oo nn o *DG*( ) [6, 0, 10, 1]. So o o [6, 10, 1] on D l n y i n l ion , i n l- o i n o il oo nn o *DG*( ). o [6] i n l ion l o i il l n ll (*RDG*). *RDG* i on in *De* ( ). o ni ion o o i l o i i (^{2}) In [1], n n l o i il *De* ( ) (l o o *De* ( )), wi o- ni ion o o (lo). L n n W n-Jin [10] l o il *De* ( ) wi i o ni ion o ((^{2})). In [13], Li n l o i il o *De* ( ) on 1 n - o n i o in o ion. Al o i l o i i l , i l n n ny o vio n i i n l w y oo nn o *DG*( ). l o i i ov l o [1]. Al o y o i o ni ion o o o l o i i , n ly (lo), o l- o i i on o ni ion , w il [1] i o ni ion . T , o l o i onv . F o , o l i- n lin o o o l o i i ll , w will ow in v l ion ion, in FLDT no n only o D l n y i n l ion in i in l o ni ion (i i y no i n).

4 Triangulation Algorithm

In i ion w n n w F Lo li D l n y T i n l ion (FLDT) l o i il *De* ( ) .

4.1 Description

T FLDT l o i i n li , i o no ly on ny n li o- on n , n lo li , in no only i o nowl o o no in i - o n i o oo . T l o i il i n l ion n o in w n ny i o no lon *DG*( ) i onn . T l o i on i o ollowin lo i l :

1. The *neighbor discovery* step. T o o i i o llow no o i ov i n i o . Fo o l i y, w i n n ly l o i in on x o x in , w ll no now i n i o . T i ion o o o l o i in on x o yn i

in (y i x n o BEACON) i o on o
S ion 6.

2. The *triangulation* step. T o o i i ol no o-
n v i oi ni o lvn Dl ny in l ion . B
on in o ion oll in ni o i ov y , no
lo lly o Dl ny in l ion. Fo onv ni n o x o i ion, w
in o i , . . . , $\triangle_P(Q R)$ ol i, o in
o in l ion o y no , i n l $\triangle$ QR o l xi . , .
. , $\triangle$ QR will l o w n in o i no i l
no . W n , . . . , $\triangle_P(Q R)$ ol , i $\angle Q R \geq \pi/3$, n o
TRIANGULATE $\triangle$ QR o ll no wi in n .

T o o $\pi/3$ on i ion i o n no no will i o
n 6 TRIANGULATE y i own ini i iv (in [1]). Sin no -
i ion l n in ollowin , o l o ni ion o o
FLDT i (lo). In i , on n involv in i o n i ll,
, w ow in S ion 5, no nno n l n 6 o no
in v .

3. The *sanity* step. T o o i i o l ni o no
li in in on i n D l n y i n l ion . T y o o y o in i n-
l ion o lo lly wi i n l ion o y i n i o
in S , v i y TRIANGULATE . No y o in
TRIANGULATE , no y l n o n w no no i
i n i o . T i i i ion l in o ion will n v n w D l n y
i n l ion , i n l ion o wi i n i o . How v ,
TRIANGULATE y inv li o o i n l ion o in
S . T i y n i :) Q o R o TRIANGULATE
wi o no inv li $\triangle$ QR, i. ., $\in \bigcirc$ QR, o) o no W
n TRIANGULATE wi n in in i n l WXZ, w i
X o Z inv li $\triangle$ QR, i. ., $X \in \bigcirc$ QR o $Z \in \bigcirc$ QR.) n
no only in in i i i n i o no w o o
no inv li i , w il) voi xi n o in ion [1].

4. The *Gabriel edges* step. T o o i i o o
ll i in i l . wi , i lw y in o , i l
Q o w i no i , . . . , $\triangle_P(Q R)$ ol (. ., wi in
o l in S 3) wo l no in l y . T i will in n i y o
, w il in ( ) (no i l lw y lon
o D l n y i n l ion n n in lo lly wi o i ion l
x n o in o ion).

Optimization. To i li y o l o i , ll TRIANGULATE o l
n in in l on ol . □

W n o in FLDT wi vio ol ion [1 , 10] on no i
i li i y o o l o i o o wo in i , w l ov o-

[1] Note that case *i*) can also prevent some intersections.

in S ion . . Fi , o o l n in TRIANGULATE , lon , o on o j i n l ion o o y n i o in i own TRIANGULATE (n vi -v), i. ., i no n o i li . T i in i il on o v ion wo D l n y n i o o no n o on o i $\triangle$ QR. I n ol no Q n R i wo l no o o n o o . T n n l i i , in , o n wo no n Q lw y on w Q o l xi (L). S on , i no , Q n R w on ly xi n o $\triangle$ QR, in y $\triangle WXZ$, on o no o $\triangle WXZ$ i in i ◯ QR, n , Q n R will li n o TRIANGULATE on $\triangle WXZ$, o in i $\triangle$ QR o l i l n o ly , Q, n R (L 5).

4.2 FLDT Builds *PLDel*(*V*) in a Single Communication Step

In i S ion w ow , in l o ni ion , o l o i il *De* ( ). To i , w on ollow n n n y oo w . T i n l ion o o l o i i - o $De^{(1)}$( ). nly 3 o l o i ov o : i o i n l i no lon o $De^{(1)}$( ) in l , n o ll in in i n l i no lon o $De^{(2)}$( ). T o , il y FLDT i o $De^{(1)}$( ) (L 3), w i i l n (L 5). In , i i *De* ( ) (T - o 1).

In oo w n wo i i n lin - (, no lo). T i ion i o o l i y. In S ion 6 w i ow lo y lin n y l o i in i l yn i in (w no n join o l v). No l o in oo w no o no o- i l (i n io n ivi lly in i l i - in l).

Lemma 1. DG( ) B D

W no i B in D n i $d($ $B)$ in l , now o , B n D. Sin B in D, $d($ $B)$ n $d($ $D)$ v ov l in (on y v n on in o) n n i ollow l on o no (. .,) i in i i l n y o i o no (. . $d($ $B)$), ovin L . □

Lemma 2. $\triangle_A(B$ $)$ B B, B TRIANGULATE $D \in$ ◯ B

o Fi 1. Sin non- i l B xi , in i $d($ $B)$ (. . [10]). In i , B nno xi B i

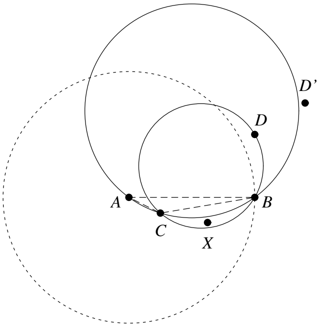


Fig. 1. A and B do not agree on $\triangle ABC$

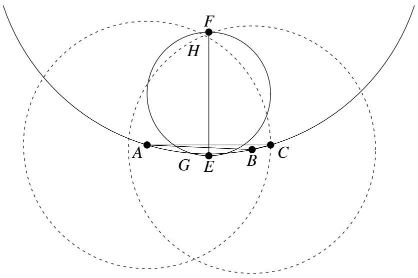


Fig. 2. A, B and C wrongly agree on $\triangle ABC$

$\triangle_B(X\ D)$ ol B o o no X n D n XD in B (- w.l.o. . X n on i o B, o i ly wi $X =$). $D \in \bigcirc\ B$, o wi $\bigcirc BXD$, wo l on in w i wo l on- i ion (D' in o i $d($ $B)$ n lo o B n o : $\bigcirc D'B$ in $\bigcirc\ B$ B n , o $X =$ i on in ; i $X \neq$, $\bigcirc D'BX$ in $\bigcirc D'B$ B n D', on inin o $\bigcirc D'B$ on in). Sin , $\angle XBD$ $\angle$ BD $\pi/3$, B will n in o ion o D in i TRIANGULATE . □

Corollary 1. B $D \in \bigcirc\ B$

Lemma 3. $\triangle_A(B\ \ )$ D B $D \in \bigcirc\ B$

I B nno xi B, oo ollow o L . H n , w will o on w B xi n B. T w i o on n i o $\in d($ $B)$ o w i $\triangle$ B ol n B, o no on i L . A now $\triangle_A(B\ \ )$ ol , w il $\triangle_B(\ \ )$ o no ($\triangle_B(\ \ D)$ ol in , wi n D lyin on i o B). Sin $D \notin d($ $B)$ [10] w n i i l o i $D \in \bigcirc\ B$ wi $||D\ \ ||$ 1 o $||B\ \ ||$ 1 (o n o i l n o ly). T l lon o no on - i L n , in o , in $||\ \ D||$ 1, $\angle$ BD $\pi/3$ n n , B will n TRIANGULATE on $\triangle$ BD, in wi $\triangle_A(B\ \ )$ o l . T L ollow . □

Lemma 4. B B $\triangle$ B B $\in d($ $B)$

iv n L , only no o ivi l in o ov i non- i l B nno l y n in in y B o vi -v 3 o l o i . H n , no l B, $\triangle_A(B$) o no ol ny o o o in in wi no $D \in \bigcirc\ B$, w i i no i n i o o , B o . In i , y L - 1 n 3, v iv in o ion o D o o on n i o o , B n n $\triangle\ B$ will no ol ny o no , B o . □

A on n o L 3, n l i o $De^{(1)}($) (w i y no l n). T ollowin L v o n no in- ion i o i l .

Lemma 5.

o Fi [1 , 10]. A $\triangle\ B$ ol , B n n $\triangle\ B$ in F (B n). I o n on in in wi B, w.l.o. . F n ini n l $\angle F$, wi $F \in \bigcirc\ B$ (nno n n in in F' i $F' \notin \bigcirc\ B$, , in , ny i l on inin n F' wo l v o in l l on o no , B o nown y). By L , F xi only i F i i l o i o i $\triangle\ FG$ ol n F n o l o i (w.l.o. . G i l o F). In l , i G o GF wo l l o in B n . Sin y y o i F n ll n l $\angle F$ i GF. By L 1, in i , G wi in o ni ion n o , B n . $\angle\ FB\ \ \pi/3 \Rightarrow \angle\ FG\ \ \pi/3$, w i n (o o B n) will lw y li n o o TRIANGULATE wi F (o o G) n will li in w on B ( ).

Now, on i w F i i l . T n, o no G, o i ly $G =$ o w i $\triangle_E(F\ G)$ ol . By y o i B n G o no in . I $\angle F\ G\ \ \pi/3$ n TRIANGULATE n L ollow . wi , n w ivi ion in i n : G xi G n G o no xi G. In , $\triangle_G(\ \ H)$ ol n H y , in , no F. Fo on i il o on iv n o , $H \in \bigcirc\ B$. $\angle\ HB\ \ \pi/3 \Rightarrow \angle GH\ \ \pi/3$. Sin G now o F, $H \neq F \Rightarrow ||H\ \ ||\ \ 1 \Rightarrow \angle HG\ \ \pi/3$. T i n i G o o o will n TRIANGULATE wi in o ion o F o $H \in \bigcirc\ B$. In on , i i n l ion o y G o no in l non- i l G n, y L , o o $X \in d(\ \ G)$, G will n in o ion o $H \in \bigcirc\ XG \Rightarrow H \in \bigcirc\ B$ ov B. W G xi o no in G, y L 1 n o oll y 1 , B n will o o in in wi no $H \in \bigcirc\ B$, wi in $\triangle\ B$ o l . □

W now in no n n i TRIANGULATE in n- n ly o o in in l o ni ion , y L 3 n 5 n o on x l in o , i ollow FLDT il o

De ( ). How v , w ill n o ov i i no o i l o o $B \in$ *De* $^{(1)}$( ) o in o ly l o nno n n o o o in in $F \notin$ *De* $^{(1)}$( ).

Theorem 1.
De ( )

o Fi . I $B \in$ *De* $^{(1)}$( ) i l y xi n o $F \notin$ *De* $^{(1)}$( ) i nno i l , i l i lw y o . H n , $\exists$ $\in d($ $B)|$ $\triangle$ B ol n B. How v , w.l.o. . $F \in \bigcirc$ B . Sin $F \notin$ *De* $^{(1)}$( ), i i no i l n $\exists K_1 \in d($ $F)$ (no own), K_1 o K_1F in B (no $||K_1$ $||$ $||$ $F||$ n $||K_1F||$ $||$ $F||$). iv n $B \notin d($ $F)$ n $B \notin$ $d(K_1$ $)$ i in ion i wi K_1 ($d(K_1$ $F)$ i in ion i wi K_1F), i ollow v n i in in $\notin$ *De* $^{(1)}$( ), w n in iv ly onin n il w n on in in $\in$ *De* $^{(1)}$( ). H n , v n i B i l o o $F \notin$ *De* $^{(1)}$( ), i o o $\in$ *De* $^{(1)}$( ) wo l l i i ly l B. T o ollow . □

5 Evaluation

In i ion, w o) o in o n in o ollowin : *R G*, *GG*, *De* , *DG* n *D* n) i n lin o o FLDT v l o i o [1]. Fi 3 ill in n wo o 100 no . W v PS o in l o i [ ] in ll , x *DG*, w i i no l n . In *DG* w v y o in l o i . l o *D* i only o v n , , w v i o , i n l ion i no o i l in wi l nvi on n . Sin no n i y i l i on o n o o in l o i , in o x i n , w v i i v i l n o no (w n 1 0 n 600) in i o x i (.5 i o ni ion n). T o l no i n i y nno i ily , i onn o olo i wo l l wi i o ili y. n o n , in in no n i y will n *DG*, y o in will onv wi in in ly i o ili y n , nli inin , will o o .

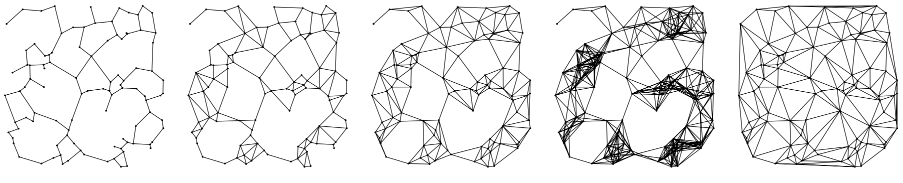

(a) *RNG* (b) *GG* (c) *PLDel* (d) *UDG* (e) *DT*

Fig. 3. Example of graphs

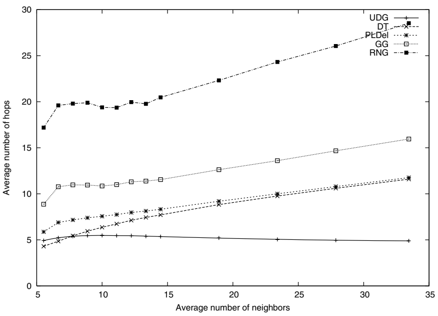


Fig. 4. Average number of hops

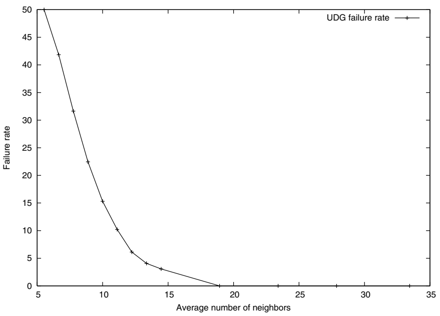


Fig. 5. Failure rate in the *UDG*

Fi ow v l n in n o o (o w y i no il), w il Fi 5 i n o il o y o in l o i in *DG* . Bo v n ion o v n o n i o o no [2]. F o , i i i vi n w n no n i y i i , no n o n *DG*, nl o y i n i n no o no w n o in in ll i n i o . In i , *De* y oo o ion, no n o in in only on n n o n i o in v . n o n , w n no n i y , *De* i ni ly l oi , i i v o n on l o i n o in onv n . Sin o i ili y o y o in il lw y xi , no ow l no n i y i , i y l o oo i o in in wo in o y: *DG* n *De* . T oin i o y in *DG* w n v o i l o o n on n wi o i - n l l o i n o *De* in y il . S ol ion v n o in o livio o no n i y. I i l o in in o o v n o o o in in *De* i y i lly i lo o n in *D* , o i n i i , w ll o , i no w n no n i i ll, in , *D* lon , vin ny o .

To o l o v l ion, w i in Fi 6 v n o n i o nno n y no , in l o i o Li . n in o own l o i . No w n v i n l i nno n , wo no o n (n in no i no o n). T l o i o Li . l o n o n- no n i l , w i o n only on no (in, n in no i no o n). W n n o no nno n ili in o l o i n i y in , n o l o i nno n - oxi ly w n 5. n i w no o n i i o in . F o , w il o l o i n in l o ni ion , l o- i o Li . n . T o , w li v l ow o l o i il *De* v y i n ly.

[2] For a node whose communication (unit) disk is entirely inside the simulation square.

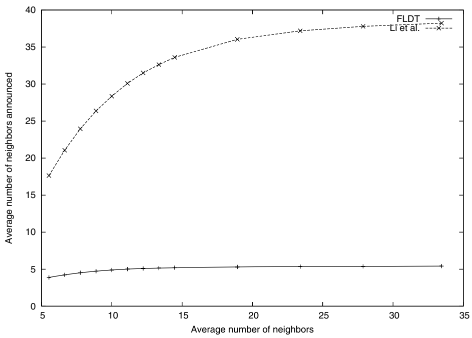


Fig. 6. Average number of neighbors announced by each node

6 Application in Dynamic Settings

So ,w v i x ion o o l o i in i in ,w no now ll i n i o . W now i li ion o o l o i in yn i in .

T li ion o ny il in l o i in yn i in i o l n y ni o i ov n w no n o / il o xi in no . In n o i i i l n ion, onni o y n on y i l n lin l y nolo y. How v , in li (o in n , [1 ,10]) i i lly no io i lly x n BEACON . W wo l li o i o l o i i i l ly w ll i o in , TRIANGULATE n ily i y o (o v n l) BEACON . T o , w n BEACON i , o l o i n i l n wi no i ion l , o in x ly o i iv wi o i l o l iv N i o oo , w i no oo nn o *DG*.

Al o, o o i li i y, w v nn l in o x o iion (i. ., no lo). How v , in yn i in , BEACON v o x n io i lly. T i n , no i ion l o in o n o x n , o l o i y n i io illy TRIANGULATE n l l *De* n o io . T o , v n i lin lo y, i n own , lon lin i (, i i n in ni ly o n y o n i n iv in ni ly o n y i iv [15]), ny n w no will v n lly i i in i nl ion.

7 Conclusions

o in o o ol o wi l n wo y n o in ln n lo li D l n y i n l ion o i v oo o in o n ,

w il , i , n in onv n . T o , in i w
n n w l o i , FLDT, o il w ll- nown ll *De* .
x i n l l ow *De* n i o i
DG, w n no n i y i ll, o o l n y n
o in onv n o ll no n i i .

FLDT o ni ion o o (lo), w i i wi in on n
o o i l n i in l o ni ion (nli vio wo ,
i o ni ion). W v l o own i n lin
o o FLDT i ll n o vio o , o ll
n o on ol . F o , in yn i in i
x n o on , o l o i i no o n
l o i o il v y i l in i n *GG* o *R G*. T o ,
o i i n y, o l o i i o i l l v n in lo ion-
wi l n wo .

Acknowledgments

T o n l o An oni Lo o l l o n on li
v ion o i .

References

1. Jean-Daniel Boissonnat and Monique Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112(2):339–354, 1993.
2. Prosenjit Bose and Pat Morin. Online routing in triangulations. In *10th Annual Internation Symposium on Algorithms and Computation (ISAAC)*, 1999.
3. Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in *ad hoc* wireless networks. In *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 48–55, 1999.
4. S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, (2):153–174, 1987.
5. K. Gabriel and R. Sokal. A new statistichal approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.
6. Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric spanners for routing in mobile networks. In *2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01)*, 2001.
7. Brad Karp and H. T. Kung. GPRS: Greedy perimeter stateless routing for wireless networks. In *ACM/IEEE International Conference on Mobile Computing and Networking*, 2000.
8. E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *11th Canadian Conference on Computation Geometry (CCCG 99)*, 1999.
9. Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'02)*, 2002.

10. Luan Lan and Hsu Wen-Jing. Localized Delaunay triangulation for topological construction and routing on manets. In *2nd ACM Workshop on Principles of Mobile Computing (POMC'02)*, 2002.
11. Der-Tsai Lee and Bruce J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.
12. Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.
13. Xiang-Yang Li, Ivan Stojmenovic, and Yu Wang. Partial delaunay triangulation and degree limited localized bluetooth scatternet formation. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):350–361, April 2004.
14. J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with Delaunay triangulation overlays. Technical Report CS-2001-26, University of Virginia, Department of Computer Science, Charlottesville, VA 22904, 5 2001.
15. N. Lynch. Distributed algorithms. In *Data Link Protocols*, chapter 16, pages 691–732. Morgan-Kaufmann, 1996.
16. F. P. Preparata and M. I. Shamos. *Computational geometry: An introduction.* Springer-Verlag, New York, 1985.
17. R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1977.
18. Ivan Stojmenovic. Position-based routing in ad hoc networks. *IEEE Communications Magazine*, July 2002.
19. G. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 4(12):261–268, 1980.
20. Yu Wang and Xiang-Yang Li. Geometric spanners for wireless ad hoc networks. In *The 22nd IEEE International Conference on Distributed Computing Systems*, 2002.

Robust Topology Control Protocols

S o , K vin Lilli , S v P n i ,
n S i P j

The University of Iowa, Iowa City, IA 52242-1419, USA
{ghosh, lillis, spandit, sriram}@cs.uiowa.edu

Abstract. Topology control protocols attempt to reduce the energy consumption of nodes in an ad-hoc wireless network while maintaining sufficient network connectivity. Topology control protocols with various features have been proposed, but they all lack robustness and are extremely sensitive to faulty information from neighbors. For example, the XTC protocol (R. Wattenhofer and A. Zollinger, XTC: A practical topology control algorithm for ad-hoc networks, *WMAN 2004*) can be forced to construct a disconnected network even if two nodes in the network receive slightly faulty distance information from one neighbor each. A key step in most localized topology control protocols is one in which each node establishes a total ordering on its set of neighbors based on information received from them. In this paper, we propose a metric for *robustness* of localized topology control protocols and define an r-robust topology control protocol as one that returns a correct output network even when its neighborhood orderings have been modified by up to $r-1$ adjacent swaps by a malicious adversary. We then modify XTC in a simple manner to derive a family of r-robust protocols for any $r > 1$. The price we pay for increased robustness is in terms of decreased network sparsity; however we can bound this decrease and we show that in transforming XTC from a 1-robust protocol (which it trivially is) into an r-robust protocol, the maximum vertex degree of the output network increases by a factor of $O(\sqrt{r})$. An extremely pleasant side-effect of our design is that the output network is both $\Omega(\sqrt{r})$-edge connected and $\Omega(\sqrt{r})$-vertex connected provided the input network is. Thus ensuring robustness of the protocol seems to give fault-tolerance of the output for free. Our r-robust version of XTC is almost as simple and practical as XTC and like XTC it only involves 2 rounds of communication between a node and its neighbors.

Keywords: Ad-hoc wireless networks, fault-tolerance, k-connectivity, robustness, topology control protocols.

1 Introduction

A - o wi l n wo on i o ono o vi o no o ni in
wi o y io. Ty i lly, o no o iny ow
o n i i o in n on in on o n o n y no

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 94–109, 2005.

n o o ni in wi o no . o o ol o ow on ion o no in o o in li o n- wo . Ty i lly, n y i y no o n i o no in l i lly wi i n w n n . A on- n , ow on ion i i ni n ly i o o w o o n o in i no , i n w n on iv no in i ll. To olo y on ol o o ol oo n- i ion ow l v l o no o no o ni wi j w n y no . in n i ion ow l v l l o olli ion n - o v n y y in n o n i ion . How v , lo l oi o n i ion ow l v l o no o in n wo o olo y i in lo l o i onn ivi y n n o l i l o w n i o no . T wo i y o l o o olo y on ol: (i) in n i ion ow l v l o v n y n (ii) in- inin onn ivi y n n n y o o o in o in i n y, l ly in onfli wi o . Any i o y ol ion o o olo y on ol o l n o i y i l y.

L $G = ($ $)$ no - o n wo wi v x no in o no n no in o o ni ion lin . L : $\to \mathbf{R}^+$ o n ion o i non-n iv l o o $e \in$. Fo v x $u \in$, l (u) no n i o o u in G. D in o o o olo y on ol o o ol , v x $u \in$ oo $_P(u) \subseteq$ (u) o v i o n i o. L in $_P$ no o i $\{(u\ v) \mid u \in$ $v \in$ $_P(u)\}$, w n vi w o o i nnin $G_P = ($ $_P)$ o G. Ty i lly, i i i G_P i y ollowin o i .

Symmetry. I $v \in$ $_P(u)$ n $u \in$ $_P(v)$. A oin o y [9, 11], wi o y y v n i l o ovi in n A K in on o - iv n o i o . Sy y i li G_P n vi w n n i . T i o o o o o i in y y, i i no o y i v y i l o i o . In - i in o i o i , w G_P i n i .

Sparseness. T i o y i y i lly n i $|$ $_P| =$ $(|$ $|)$. n, on o y, o o n i i . T i o y i o ll v i u, $|$ $(u)| \leq$ o o on n . B . l. [ ] oin o n i o n o n low in n , n w il i y in n “ v ” n , i i no in n l. [ ] l o n on l ni ion o i o in n n on y, in i ion o (o n l n iv o) n , i G_P ini i i in n i .

Connectivity. G_P i i o onn , ovi G i onn . n, on v ion o onn ivi y k- onn ivi y o k- v x onn ivi y (o k 1) i . T on v ion o on- n ivi y i ly G_P l i l o o in w n i o v i n i o l - ol n o lin o v x il .

Spanner Property. Fo ny i o v i u n v, l $(u\ v)$ (iv ly, $_P(u\ v))$ no o o w n u n v in G (- iv ly, G_P). T n, nn o y i xi n o on n $_P(u\ v) \leq \cdot\ (u\ v)$ o ll i o v i $u\ v \in$. I on n xi , n G_P i ll o G.

L y i lly, in o o i l n i y o G_P l o - i . I G_P i l n , n o i o in l o i **GOAFR**$^+$ ovi i n o in in n wo [3].

In i ion o o , o in G y v o i non-n iv l . T o o n i lly i in o , l o i l n . n i i v i o in G in o i . In i , o n $\{u\ v\}$, no $|uv|$ i l o i n w n u n v in . I v i o G in li n , n G i ll . A i l o li n i . G i ni i i i v i in li n l n n o ny i o v i u n v, $\{u\ v\}$ i n o G iff $|uv| \leq 1$. A n ion in [11], ni i lly o o l n - o n wo w ll n wo no l in n no l n n v l (no li) n i ion ow n i o o i n nn , i , n nn n i in wi i n i l ow in v y i ion o l n . T o o n $\{u\ v\}$, $(u\ v)$, i y i lly o no o n o n y on n oin o o x n in o o o ni wi o n oin . Fo li n , i i on l o $(u\ v) = |uv|^\alpha$ o o $\alpha \geq$. I G_P, o o o olo y on ol o o ol i nn o y wi o o , n i i ll n . I G_P i nn o y wi o l n , n G_P i ll .

V io o olo y on ol o o ol v n o o , n in o o ov n ion o i . H w n ion wo o o ol o ovi on n . W n n Li [10] v o o lo l o o ol o on ion o y i , o n , l n nn o n wo o l y ni i . W will ll i WL o o ol. W n o n Zollin [11] v o o i l o- o ol ll XT on y i , o n , l n n wo o n wo o l y ni i . In i ion, XT n y - i , onn n wo v n o in n wo v i y l n . In vo o WL o o ol i i o o ol i n o - n nn , w XT ovi no n . [11] o n x i n o o o XT y oo nn in " v ." In vo o XT o o ol i i x i li i y n o i onn v n w n in i no ni i . T i i li i y o i o XT v n w n in on w i no i i i no - i n ion l l n n v n w n o l in in.

In i , w y oin in o xi in o o ol
o o olo y on ol o l , in l in WL o o ol n XT , l
n x ly n i iv o l y in o ion o n i o . Fo
x l , w ow in S ion , n wo on y XT y n
o in i onn v n w n wo no iv l y i n in o ion
o on n i o . In y in WL o o ol, XT , n o
o o ol on o o ol i in [5], no u o
o l o in $\prec_u$ on i n i o oo (u). In WL o o ol $\prec_u$ i
on o v i in (u), in XT $\prec_u$ i on “ li y” o
lin w n u n v x in (u), n in on o o ol $\prec_u$
i on n l . In , o in o ion o n i o i i i l
o o n o n i o oo o in n o i i l o
o n o o o ol i l .

F l y o j in o l in o ion o n i o i o on in
- o wi l n wo . An x l o o o ol wo o ly in
n o noi y i n in o ion i n wo lo li ion o o ol in [ ].
F l y in o ion o n i o o l o i n l in n , o
l ow ly n no , o iv vin n in o
i o n i ion n n , . v n i in o ion o
n i o i no l y, i o l i ly o -o - , no y
o il . F o , i i l o o i l no y only v oxi
o i ion w n n l n only ly oxi i n in o -
ion (o x l , wo in [ ] on o in vi l oo in). Fo
ll o on , o l o in $\tilde{\prec}_u$ o y no u on i n i -
o oo (u) y iff n o l o in $\prec_u$ u i v
o , i n iv n o o l in o ion. o l i o
vi o olo y on ol o o ol wo o ly v n w n no u
o $\tilde{\prec}_u$ (n $\prec_u$), ovi $\tilde{\prec}_u$ i no oo “ w y” o
$\prec_u$. A n l o i n w n o in (o ion) o
i w n o w o j n l n i o o on o -
in o o . W n (in o lly, o now) r o o ol on
n wi n o l o o $r-1$ j n w o on ll
n i o oo o in . W i no ion i in S ion . W oin o
XT i no v n - o . W n n i l o i ion o XT
n n i in o n r- o o o ol o ny in r 0. T i w y
o in in o n i in o in i y o n -
wo . How v , w o n i . Mo i lly, in n o in XT
o 1- o o o ol (w i i i , ivi lly) o r- o o o ol, o ny
in r 1, w in xi v x o o y
o o $(\sqrt{r})$. v n wi o i ion , XT on in o x ly
i l n i l. An x ly l n i - ff o o i n i
o n wo i o $\Omega(\sqrt{r})$- onn n $\Omega(\sqrt{r})$-v x onn . In
o wo , n in o n o o o ol o ovi l - ol n
o o o .

2 XTC is Not Robust

W i ion y o in XT o o ol o [11].

```
1. Establish order $\prec_u$ over $u$'s neighbors in $G$
2. Broadcast $\prec_u$ to each neighbor in $G$; receive orders from all neighbors
3. Select topology control neighbors:
4.     $N_u := \{\ \}$; $\widetilde{N}_u := \{\ \}$
5.     while ($\prec_u$ contains unprocessed neighbors){
6.         $v :=$ least unprocessed neighbor in $\prec_u$
7.         if($\exists w \in N_u \cup \widetilde{N}_u : w \prec_v u$)
8.             $\widetilde{N}_u := \widetilde{N}_u \cup \{v\}$
9.         else
10.            $N_u := N_u \cup \{v\}$
11.    }
```

A n ion in [11], o o ol on i o in : (i) n i o o in (Lin 1), (ii) n i o o x n (Lin), n (iii) l ion (Lin 3-11). In l ion v x u i o o v o i o n i o i i v x w u n v o i lly . Mo i ly, u o v o i n i o oo i xi w $w \prec_u v$ n $w \prec_v u$. In o o ol, v i l $_u$ i o n i o u o n o in n v i l $\widetilde{\ }_u$ i o n i o u o n o o . L $_{XTC} = \{(u\ v) \mid v \in \ _u\}$ n $G_{XTC} = (\ _{XTC})$. Al o, l $\prec = \{\prec_u \mid u \in \ (G)\}$ no oll ion o n i o oo o in . No o o ol l v $\prec$ n i . T G_{XTC} i n ion, no only o in n wo G, l o o n i o oo o in $\prec$. T i n n y will i o n l n o i i w no ion $G_{XTC}(\prec)$ o no n wo on y ov o o ol. In n l, o o olo y on ol o o ol , w no ion $G_P(\prec)$ o no o o . I i ily v i $u \in \ _v$ iff $v \in \ _u$ n n $G_{XTC}(\prec)$ n o o n i .

A n ion in in o ion, XT i x ly n i iv o ll - ion in n i o oo o in . In [11], i i own i G i li n n $\prec = \{\prec_u \mid u \in \ (G)\}$, w $\prec_u$ i n

$$v \prec_u w \Leftrightarrow (|uv|, \min\{id_u, id_v\}, \max\{id_u, id_v\}) < (|uw|, \min\{id_u, id_w\}, \max\{id_u, id_w\}),$$

n $G_{XTC}(\prec)$ i y i n onn . W will ll ov n i o oo o in , o in . No in i n - o in , i only o i . W now n i l x l o -v x ni i ill l o o n o XT . W wi n i o oo o in $\prec$ n ov , y li n i n . W n on w in n i o oo o in o wo v i o o in n w n i o oo o in $\widetilde{\prec}$. W oin o $G_{XTC}(\widetilde{\prec})$ i no onn . on i ni i own in Fi 1. Fo o in on , l l n o $| b| = |d \ | = \sqrt{3}/ \ $, $| \ d| = |b \ | = 1/ \ $, n $| \ \ | = |bd| = 1$. T n

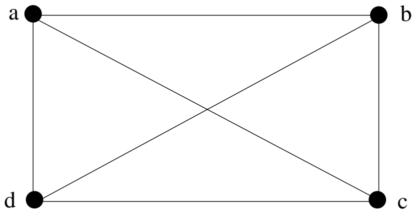


Fig. 1. A unit disk graph for showing the sensitivity of XTC to small perturbations

$$d \prec_a b \prec_a$$
$$\prec_b \quad \prec_b d$$
$$b \prec_c d \prec_c$$
$$\prec_d \quad \prec_d b$$

Now o $\tilde{\prec}_a = \prec_a$, $\tilde{\prec}_d = \prec_d$,

$$\tilde{\prec}_b \; d \; \tilde{\prec}_b$$
$$b \; \tilde{\prec}_c \quad \tilde{\prec}_c \; d$$

No $\tilde{\prec}_b$ n $\tilde{\prec}_c$ o in y w in on i o l n in $\prec_b$ n $\prec_c$. I XT i n on ni i own low wi $\tilde{\prec} =$ $\{\tilde{\prec}_a \; \tilde{\prec}_b \; \tilde{\prec}_c \; \tilde{\prec}_d\}$ n $G_{XTC}(\tilde{\prec})$ on in j wo $\{ \; d\}$ n $\{b \; \}$ n i o i onn . T o l o wo j n w w i n o onn ivi y. L in w o i y XT in i l nn in o n r- o o o ol, on n ol o l o o $r-1$ j n w on i n i o oo o in .

3 Characterizing Good Neighborhood Orderings

XT ' o n n o n i i lly n on $\prec$. S i lly, i $\prec$ i o i ly n n ollowin wo o i ol :

(i) Fo v y i n l b , $\prec_a$, $\prec_b$, n $\prec_c$ l v i , b, n n o i o in o on o $\{ \; b\}$, $\{b \; \}$, n $\{ \; \}$.
(ii) Fo v y $\_ \; (\quad \overline{})$ o G, $\prec$ v n o in o o o $(\quad \overline{})$.

P o y (i) i li $G_{XTC}(\prec)$ i i n l - , w il (ii) i li $G_{XTC}(\prec)$ i onn . V io o i o $G_{XTC}(\prec)$ ov ly in [11] i i ly ollow. H w ov n l i ion o n i o-oo o in $\prec$ n o i (i) n (ii). I will l " i n -o in " in [11] i i i ion. B o i o-n ly, ny o n l n i o oo o in l o i y o i ion. Fo x l , n i o oo o in y in in i o y

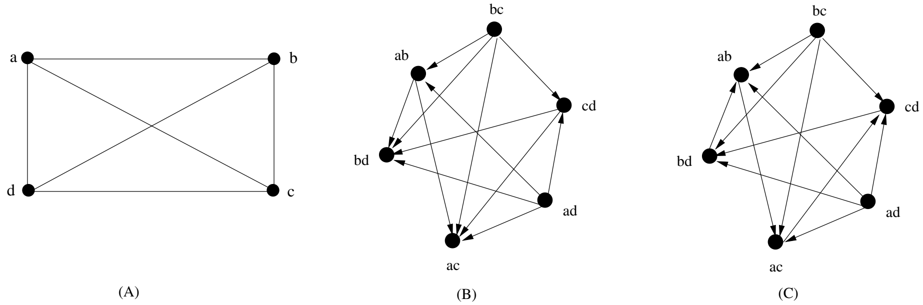


Fig. 2. On the left is the unit disk graph from Figure 1. In the middle is $L(G, \prec)$, where $\prec$ is the distance-based ordering. It is easily verified that this is acyclic. Vertices ad and bc are minimal vertices in $L(G, \prec)$. On the right is $L(G, \widetilde{\prec})$, where $\widetilde{\prec}$ is obtained from $\prec$ by swapping a and d in $\prec_b$ and $\prec_c$. Notice the cycle (ab, ac, cd, bd, ab) in $L(G, \widetilde{\prec})$. This cycle is responsible for $G_{XTC}(\widetilde{\prec})$ being disconnected

in in n l l o i y o i ion n o n o- i (i) n (ii).

T oll ion o n i o oo o in $\prec$ in in y l ion $\rightsquigarrow$ on o o G. Fo ny wo e $e' \in$ (G), $e \rightsquigarrow e'$ i e n e' o on n oin n i $e = \{u\ v\}$ n $e' = \{u\ w\}$, n $v \prec_u w$. U in i in y l ion $\rightsquigarrow$ w n n n w (i) $(G\ \prec)$ w o v x i o o G n w o o (i) i $\{(e\ e') \mid e\ e' \in\ (G)\ e \rightsquigarrow e'\}$. W ll $\prec$ i $(G\ \prec)$ i n y li . No i $(G\ \prec)$ i y li , n o i ny o $(G\ \prec)$. Al o no ny y li i n o on in l on v x wi in- (iv ly, o -) 0 n w ll v x, (iv ly,) v x. Fi ill ni ion o $\rightsquigarrow$, n $(G\ \prec)$.

Theorem 1. . G $\prec$ G $G_{XTC}(\prec)$ $\prec$

To ow $G_{XTC}(\prec)$ i i n l- , w on i n i y i n l b in G. Sin $(G\ \prec)$ i y li i i n l , y $\{\ b\}$, $\{b\ \} \rightsquigarrow \{\ b\}$ n $\{\ \} \rightsquigarrow \{\ b\}$. T i i li $\prec_b$ n $\prec_a b$. A l XT will o $\{\ b\}$ n o i n l b i no o $G_{XTC}(\prec)$. Sin oi o b w i y, $G_{XTC}(\prec)$ i i n l- .

To ow $G_{XTC}(\prec)$ i onn , w on i (¯) o G. L $_S(G)$ o $(G\ \prec)$ in y o G o in . Sin $(G\ \prec)$ i y li , o i $_S(G)$. L e ini l v x o $_S(G)$. W now ow e i in in $G_{XTC}(\prec)$. L $e = \{u\ v\}$ n o e i no in in $G_{XTC}(\prec)$. T n i v x $w \in\ (G)$ i o on n i o o u n v $w \prec_u v$ n $w \prec_v u$. Sin $\{u\ v\}$ o (¯), l on o $e_u = \{u\ w\}$ o $e_v = \{v\ w\}$ l o o . Wi o lo o n li y o e_u o ($\overline{\ }$). T o , e_u i v x in $_S(G)$. T n, y ni ion o $\rightsquigarrow$, $e_u \rightsquigarrow e$ n o e i no ini l in $(G \prec)$. T i on i o oi o e $\overline{\ }$ ini l v x in $_S(G)$.

T w v own o v y ($\overline{\ }$) o G, n in $G_{XTC}(\prec)$ o in . T i ow $G_{XTC}(\prec)$ i onn .

I i y o i n - o in i y li . L G li n n l $e = \{u v\}$ in G i l $(|uv|$ in$\{ d_u \ d_v\}$ x$\{ d_u \ d_v\})$ i in in in l xi o i o - in o ll i l . F o ni ion o i n - o in , i ollow e i ini l in $(G \prec)$. I w $(G - e \prec)$ i y li , n y in ion i ollow o i $(G \prec)$. Si il n ow ollowin l n o in l o y li .

1. T $\prec^{id}$. L v n w wo n i o o u. T n $v \prec_u^{id} w$ iff d_v d_w. A o , $\prec^{id} = \{\prec_u^{id} | \ u \in \ (G)\}$.
. T $\prec^{a}$. Fo ny i o v i u n v in G, l $\alpha(u \ v)$ no n l y lin n uv wi o i on l y wi o i in u ow $+\infty$. Fo wo n i o v n w o u, $v \prec_u^{a} w$ iff

$$(\alpha(u,v), \min\{id_u, id_v\}, \max\{id_u, id_v\}) < (\alpha(u,w), \min\{id_u, id_w\}, \max\{id_u, id_w\}).$$

o , i - o in i only w ll- n w n ll v i v (no n ily i in) i n n l - o in i only w ll n w n v i o G in li n n v i v i . T l i n o i w n n l o i on i no no o i in i n i o .

T i li ion o ov i ion o i XT o l v w ll n n wi i - o in o n l - o in in o i n - o in n o wo l ill v o i : (i) y y, (ii) onn ivi y, n (iii) in i n l - . How v , i o l no i no in i n o l ly n in i - o - in o n l - o in i no , in n l, oo i . T o y y

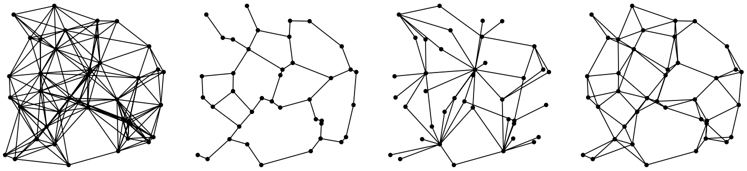


Fig. 3. The graph on the left is a unit disk graph obtained by dropping 40 points uniformly at random on a 3 × 3 grid. It contains 197 edges. The second graph from the left is the output of XTC using a distance-based ordering and it contains 47 edges. The third graph from the left is the output of XTC using an id-based ordering and it contains 55 edges. The rightmost graph is the output of k-XTC using a distance-based ordering, for $k = 2$. It contains 88 edges

n onn ivi y v , o y v o n i l . So o n in in Fi 3 (i o l) i on y XT in i - o in . Fo x l , - o in v i i i n o ov i v i v v l n n in i n on . T no o on o i on ion n n wo i v ln l o il o no . W il w no in o i - o in n l n iv o i n - o in , l in T o 1 o o i ili y o in i - o in w n i n o n i o i il (no n ily). T i y no w y o in o n o o o ol.

4 *k*-XTC: A Robust Version of XTC

In i ion, w o o ll o i ion o XT will n i in o o o o ol. T o o ol, w i w will ll k-XT i o in o XT y n in Lin o ollowin .

$$\text{i } (\exists W \subseteq {}_u \cup \widetilde{\ }_u\colon |W| = k \text{ n } \forall w \in W : w \prec_v u).$$

T i o i ion i ly n i ion o u o o v o i n i - o oo n o o no on , k o v i o u n v lly . L $G_{kXTC}(\prec)$ no o o k-XT . No XT i i ly i l o k-XT wi $k = 1$. A i l i o n o v ion o o o k-XT i ollowin .

Proposition 1. . . , k 1, . . , j, $1 \le j$. k, $G_{jXTC}(\prec)$ . . . . , . . $G_{kXTC}(\prec)$

T i o in Fi 3 ow o o k-XT o $k =$. T i o “ ” o o XT (i on o l) i o n n non- l n . A w will ow l , i i k- onn w ll k-v x onn . T o , v y v x in i l k.

W now n i y no ion o o n ollow .

Definition: L π n π' wo ion o ni , non- y . W no w n o j n w n o n o π o π' y d $(\pi\ \pi')$.

Definition: L $\prec = \{\prec_u |\ u \in\ (G)\}$ n $\widetilde{\prec} = \{\widetilde{\prec}_u |\ u \in\ (G)\}$ wo oll ion o n i o oo o in . T n w d $(\prec\ \widetilde{\prec})$ o no $\sum_u d\ (\prec_u\ \widetilde{\prec}_u)$.

Definition: A o olo y- on ol o o ol i i o r . . . . $\prec$ i $G_P(\widetilde{\prec})$ i onn o ny oll ion o n i o oo o in $\widetilde{\prec}$, w d $(\prec\ \widetilde{\prec})$. r.

In o wo , i i r- o o $\prec$, n n onn v n w n x wi oll ion o n i o oo o in i o in o $\prec$ in o $r-1$ j n w . M in “ i n ” w n o in y n o j n w ovi l n ion o n i yin v i y o i ion i v i o li v “ l ” o in on n i o . Fo x l , i v x u n i i n o n i o v n v i li n i o l in $\prec_u$. I (in o ly) i i n o v i ll n l i n , n v’ l in $\prec_u$ y ny j n w w y o i o l in $\prec_u$. W now ov in l o i . No l i ov o ny oll ion o y li n i o oo o in n no j o i n - o in . S owin k-XT i k- o i no , owin i o n n o in i n n low. T ollowin o ow o o in n r- o v ion o XT , i i i n o k-XT o $k \geq \sqrt{r}$.

Theorem 2. k $\frac{k(k+1)}{2}$ $\prec$

L G in o k-XT . L $\prec$ n i y oll - ion o n i o oo o in n l $\tilde{\prec}$ n i y oll ion o n i o oo o in . F o T o 1 n P o o i ion 1, w now $G_{kXTC}(\prec)$ i onn . W will ow i $G_{kXTC}(\tilde{\prec})$ i i onn n d $(\prec\ \tilde{\prec}) \geq k(k+1)/$. T i will i ly k-XT i $\frac{k(k+1)}{2}$- o .

W y o in $G_{kXTC}(\tilde{\prec})$ i i onn n in o no ion l onv ni n , $\tilde{H}= G_{kXTC}(\tilde{\prec})$. Sin $\tilde{H}$ i i onn i $=(\ \overline{\ })$ i no o $\tilde{H}$ o in $(\ \overline{\ })$. n o n i l on in G o in . L ( ) o in G o in . Sin $(G\ \prec)$ i y li , o $(G\ \prec)$ in y in ( ) i l o y li . In o oo w ( ) o no o $(G\ \prec)$ in y ( ).

oo i on iv n w w now i i i ion o on ion o . L $e = \{u\ v\}$ ini l in ( ). Wi o lo o n li y, o $u \in$ n $v \in \overline{\ }$. T e o no in $\tilde{H}$ n i n only n i W o k v i o ll $w \in W$, w i o on n i o o u n v, $w \tilde{\prec}_u v$, n $w \tilde{\prec}_v u$. L $(W_u\ W_v)$ i ion o W $W_u \subseteq$ n $W_v \subseteq \overline{\ }$. L $k_u = |W_u|$ n $k_v = |W_v|$. No $k_u + k_v = k$. Al o no o $w \in W_v$, $\{u\ w\}$ o n i il ly o $w \in W_u$, $\{v\ w\}$ o . Al o no in $\{u\ v\}$ i ini l in ( ), $v \prec_u w$ o ll $w \in W_v$ n $u \prec_v w$ o ll $w \in W_u$. T , w v (i) o ll $w \in W_v$, $w \tilde{\prec}_u v$ n $v \prec_u w$ n (ii) o ll $w \in W_u$, $w \tilde{\prec}_v u$ n $u \prec_v w$. S Fi o n x l . I (i) i li d $(\prec_u\ \tilde{\prec}_u) \geq k_v$ n i (ii) i li d $(\prec_v\ \tilde{\prec}_v) \geq k_u$. T in li i o i ly d $(\prec\ \tilde{\prec}) \geq k$.

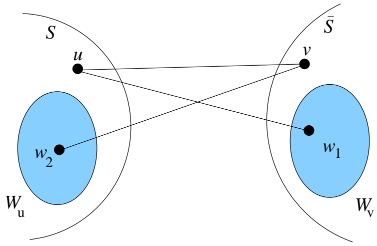


Fig. 4. The edges $\{u, w_1\}$ and $\{v, w_2\}$ cross the cut $(S, \overline{S})$. Furthermore, $v \prec_u w_1$ and $w_1 \overset{\sim}{\prec}_u v$. Also, $u \prec_v w_2$ and $w_2 \overset{\sim}{\prec} u$

Remark: A lly, o in on n l i . v n i w w n o n o $\prec_u$ in o n o in $\prec'_u$ $w \prec'_u v$ o ll $w \in W_v$, $\prec_u$ n $\prec'_u$ in i wi o in o ll o i o l n , i wo l l k_v j n w . In o wo , $\prec'_u$ i lon w y w n $\prec_u$ n $\overset{\sim}{\prec}_u$ n j in o $\prec'_u$ o $\prec_u$ l k_v j n w . in o $\overset{\sim}{\prec}_u$ o $\prec'_u$ y i ion l j n w n w o n o ly in i ion o on ion o . Si il n o “ i n ” w n $\prec_v$ n $\overset{\sim}{\prec}_v$.

T oi o $e = \{u\ v\}$ i ov , n i ion o o on ion o . L $B_1 = \{e\}$ n l $_1 = \{u\ v\}$. T $_1$ n n oin o in B_1. To o in ion y o i w n i ion l no ion. Fo ny X o v i , l $d\ _X(\prec_u\ \overset{\sim}{\prec}_u)$ ini n o j n w w n o on $\prec_u$ o v y l n $v \in X \cap\ (u)$ i in l iv o i ion in $\prec_u$ in $\overset{\sim}{\prec}_u$. Mo i ly, $d\ _X(\prec_u\ \overset{\sim}{\prec}_u) =$ in$_{\prec'_u}$ $d\ (\prec_u\ \prec'_u)$, w in o ion i ov ll $\prec'_u$ o ny $v \in X \cap\ (u)$ n o ny $w \in\ (u)$, $v \prec_u w \Leftrightarrow v \prec'_u w$. H i ll x l o ill i ni ion.

Example. D n ion $\pi = (5\ 3\ 1)$ n $\pi' = (1\ 3\ 5)$. I i y o $d\ (\pi\ \pi') = 10$. Now l $X = \{1\ \}$. W i $d\ _X(\pi\ \pi')$? I i in y o v i y $d\ _X(\pi\ \pi') = d\ (\pi\ \pi'') = 9$, w $\pi'' = (13\ 5)$. T i i $d\ _X(\pi\ \pi')$ i n o j n w n o n o π in o ion in w i 1 o ll o l n n ll o l n x 5. T o i ion o l n 1, , n 5 x .

Fo ny oll ion $\prec$ o n i o oo o in , l $d\ _X(\prec\ \overset{\sim}{\prec}) =$ $\sum_{u\in X} d\ _X(\prec_u\ \overset{\sim}{\prec}_u)$. W l o n ollowin wo l n y o n o in on ion in o no vi j n w .

Fact 1. Fo ny $X \subseteq Y \subseteq\ (u)$, $d\ _X(\prec_u\ \overset{\sim}{\prec}_u) \le d\ _Y(\prec_u\ \overset{\sim}{\prec}_u)$.

Fact 2. L $X \subseteq Y \subseteq\ (u)$ n $x \in Y - X$. S o i $W \subseteq\ (u)$ o ll $w \in W$, $x \prec_u w$ n $w \overset{\sim}{\prec}_u x$ n $d\ _X(\prec_u\ \overset{\sim}{\prec}_u) + |W| \le$ $d\ _Y(\prec_u\ \overset{\sim}{\prec}_u)$.

in ion y o i i ollowin .

Induction hypothesis: Fo ny ≥ 1, i ion o i o , w v B_i o o ( ) no o ( ) $- B_i$ in o B_i, o y o B_i in o ( ) $- B_i$. L $_i$ o n oin o in B_i. T n d $_{V_i}(\prec \overset{\sim}{\prec}) \geq k + (k-1) + \cdots + (k- \; +1)$.

W v own n o i ion o on ion o , $|B_1| = 1$, no o ( ) $- B_1$ in o B_1, n d $_{V_1}(\prec \overset{\sim}{\prec}) \geq k$. T i i o o oo .

W now ollowin l i o ($+1$) i ion o o on- ion o . W will ov i l i l ; o now w will i ol n o l oo o in ion .

Claim: In ($+1$) i ion i i o i l o i n $e' \in$ ( ) $- B_i$ (i) e' l on n oin no in $_i$, n (ii) in- o e' in ( ) i o .

A in i l i , w o in nn i i il o n o i ion. L $e' = \{u' \; v'\}$, $u' \in$, $v' \in \overline{}$, n wi o lo o n li y, $v' \notin \; _i$. T e' i no in $\tilde{H}$ i li i W o k v i o ll $w \in W$, w i o on n i o o u n v, $w \overset{\sim}{\prec}_{u'} v'$ n $w \overset{\sim}{\prec}_{v'} u'$. U in (iv o ov l i) in- o e' in ( ) i o , w on l , in n n i il o on o i ion, xi $W_{u'} \subseteq W \cap$ n $W_{v'} \subseteq W \cap \overline{}$, $|W_{u'}| + |W_{v'}| = (k - \;)$ n

(i) o ll $w \in W_{v'}$, $w \overset{\sim}{\prec}_{u'} v'$ n $v' \prec_{u'} w$ n
(ii) o ll $w \in W_{u'}$, $w \overset{\sim}{\prec}_{v'} u'$ n $u' \prec_{v'} w'$.

L $k_{u'} = |W_{u'}|$ n $k_{v'} = |W_{v'}|$, $B_{i+1} = B_i \cup \{e'\}$, n $_{i+1}$ n oin o v i in B_{i+1}. I (i) lon wi F i li d $_{V_{i+1}}(\prec_{u'} \overset{\sim}{\prec}_{u'}) \geq d$ $_{V_i}(\prec_{u'} \overset{\sim}{\prec}_{u'}) + k_{v'}$. I (ii) i li d $_{V_{i+1}}(\prec_{v'} \overset{\sim}{\prec}_{v'}) \geq k_{u'}$. T in- li i o lon wi F 1 i ly d $_{V_{i+1}}(\prec \overset{\sim}{\prec}) \geq d$ $_{V_i}$ $(\prec \overset{\sim}{\prec}) + (k- \;)$. T i o l in ion . I w in ion n il $= k$, n w v $_k$ o v i d $_{V_k}(\prec \overset{\sim}{\prec}) \geq k(k+1)/1$. Sin $_k \subseteq$, y F 1 w v d $(\prec \overset{\sim}{\prec}) = d$ $_V(\prec \overset{\sim}{\prec}) \geq d$ $_{V_k}(\prec \overset{\sim}{\prec}) \geq k(k+1)/1$.

W now ov ov l i n xi n o e'.

Proof of Claim: L $_i$ o no in B_i, v o n oin in $_i$. on i o ( ) o in y l in $B_i \cup \; _i$. ll i $_i$. Sin ( ) i y li , $_i$ i l o y li n l e' ini l v x in $_i$. I e' i no in i n on ny v x in $_i$, n e' i l o ini l in ( ) n w on . So w e' i in i n on l on v x in $_i$. Sin , e' w i o ( ) $- B_i -$ $_i$, e' nno in i n on wo v i in $_i$, o wi e' will in $_i$. T o , w l wi in w i

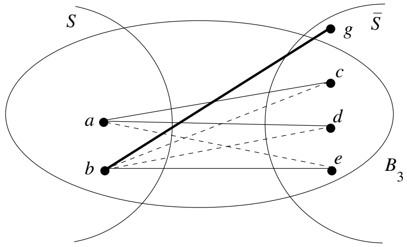


Fig. 5. This figure illustrates the proof of the Claim in the proof of Theorem 2. Here $B_3 = \{\{a, c\}, \{a, d\}, \{b, e\}\}$, $T_3 = \{\{b, d\}, \{b, c\}, \{a, e\}\}$, and $e' = \{b, g\}$. The set $V_3 = \{a, b, c, d, e\}$

e' i in i n on on v x in $_i$. Now l $e' = \{b\ g\}$ n o $b \in$ $_i$ n $g \notin$ $_i$. Fi 5 ill i ion o $= 3$. S o x in B_i in i n on b n in $_i$ in i n on b. In Fi 5, $x = 1$ n $=$. T in- o e' in ( ) i o o n ov y $x+$ n in $x \leq |B_i| =$, w o n + . Now no o v y $\{b\ b'\}$ in $_i$, i n in B_i in i n on b' o no ny n oin wi $\{\ b\}$. In o wo , o v y e'' o $_i$ $e'' \rightsquigarrow e'$ i ni f in B_i $f \not\rightsquigarrow e'$. T i iv o n o on in- o e'.

An x ly l n i - ff o o i n o k-XT i l - ol n o G_{kXTC}. W ov in ollowin wo o i G i k- onn (iv ly, k-v x onn) n G_{kXTC} i l o k- onn (- iv ly, k-v x onn). Lo li o o ol o on in l - ol n nnin in [1, 6], k-XT i i l n . F o , k-XT ovi o n , o n in in i ni i , n l o v k- onn ivi y o i ily in wi i y l n . Al o no ollowin wo o ov o ny y li oll ion o n i o oo o in , no j o i n - o in .

Theorem 3. $\prec$, $G_{kXTC}(\prec)$ k G k

S o G i k- onn , G_{kXTC} i no . Fo ny $=$ ( ) o , l ( ) no in G o in n i il ly, l $_{kXTC}($ $)$ no o G_{kXTC} o in . Sin G i k- onn , G_{kXTC} i no , i $=$ ( ) o | ( )| $\geq k$ n | $_{kXTC}($ $)|$ k. L o $(G\ \prec)$ in y ( ) $-$ $_{kXTC}($ $)$. No i non- y n in $(G\ \prec)$ i y li , o i . L $e = \{u\ v\}$ ini l v x in . Sin $\{u\ v\} \in$ $-$ $_{kXTC}$, xi v x W, $|W| = k$, o ll $w \in W$, w i o on n i o o u n v n

$$w \prec_u v \quad \text{n} \quad w \prec_v u. \tag{1}$$

L $W_1 = W \cap$ n $W_2 = W \cap \overline{}$. T n, $\chi = \{\{u\ x\} \mid x \in W_2\} \cup \{\{\ v\} \mid\ \in W_1\}$ i o (G) o o ( ). No $|\chi| = k$ n o no ll in χ n lon o ${}_{kXTC}(\ )$. L $\{\ b\} \in \chi -\ {}_{kXTC}(\ )$. T $\{\ b\} \in (\ ) -\ {}_{kXTC}(\ )$ n i o v x in . No $\{\ b\}$ i i in i n on u o in i n on v. Wi o lo o n li y, $= u$. T n, o (1) i ollow $b \prec_u v$. T i n $\{\ b\} \rightsquigarrow \{u\ v\}$, on i in $e = \{u\ v\}$ i ini l in .

Theorem 4. $\prec$, $G_{kXTC}(\prec)$ k G k

S o G i k-v x onn n G_{kXTC} i no . Sin G_{kXTC} i no k-v x onn , xi $' \subseteq$ $|\ '| = k-1$ n $G'_{kXTC} = G_{kXTC} -\ '$ i i onn . Sin G i k-v x onn , $G' = G -\ '$ i onn . Sin G'_{kXTC} i i onn , xi $= (\ \overline{}\ )$ o $-\ '$ no in G'_{kXTC} o . How v , in G' i onn , xi non- y o ${}_C$ in G' o . L o $(G\ \prec)$ in y ${}_C$. L $e = \{u\ v\}$ ini l v x in ${}_C$. Wi o lo o n li y o $u \in$ n $v \in \overline{}$. Sin no in G'_{kXTC} o , e i no in G_{kXTC}. H n , xi $W \subseteq$, $|W| = k$, o ll $w \in W$, w i o on n i o o o u n v, n $w \prec_u v$ n $w \prec_v u$. Sin $|W| = k$ n $|\ '| = k-1$, xi v x $w \in W -\ '$. T o , $\underline{w}$ i v x in G' n in G'_{kXTC}. Wi o lo o n li y $w \in \overline{}$. T o , $\{u\ w\}$ o n lon o ${}_C$. F o , in $w \prec_u v$, $\{u\ w\} \rightsquigarrow \{u\ v\}$ on i in $\{u\ v\}$ i ini l in .

W $\Delta(G)$ o no xi o v x in G. W now ow n o o n 6 [11] on $\Delta(G_{XTC})$ i G i ni i i ov l nly o k-XT , ivin n o n o $6k$ on $\Delta(G_{kXTC})$. No n i i o i n - o in , n o no y ov o n i y y li o in . In , n ion o o o i - o in , no ll y li o in will i y i o n l .

Theorem 5. G $\prec$ $\Delta(G_{kXTC}(\prec)) \leq 6k$

To ov i o , w ow $k+1$ j n in G_{kXTC} nno n lo n n l l n $\frac{\pi}{3}$. Mo i ly, v x u $k+1$ n i o $v_0\ v_1\ \ldots\ v_k$ in G_{kXTC}, li in o n lo wi o - in o i y n i o v_0. F $\angle v_0 u v_k$ $\frac{\pi}{3}$. Fi 6 ill i ion.

S o on n i o $v_0\ v_1\ \ldots\ v_k$, n i o v_i o o , $0 \leq\ \leq k$, i on i l y k-XT . Sin v_i i on i l w v

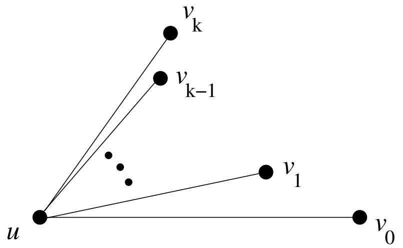


Fig. 6. The neighbors $v_0, v_1, \ldots, v_k$ of u. For the proof we suppose that $\angle v_0uv_k < \pi/3$

$v_j \prec_u v_i$ o ll $j \neq$, $0 \leq j \leq k$. Sin $\prec$ i i n - o in , i
i li $|uv_j| \leq |uv_i|$, o ll $j \neq$, $0 \leq j \leq k$.

Now on i i n l uv_iv_j, $j \neq$, $0 \leq j \leq k$. Sin $|uv_j| \leq |uv_i|$,
uv_j i no lon o i n l . Al o in $\angle v_juv_i$ $\frac{\pi}{3}$, lin
n v_iv_j i i ly o n l on o o wo lin -
n in i n l , n ly uv_i n uv_j. o inin i wi
$|uv_j| \leq |uv_i|$, w v $|v_iv_j|$ $|uv_i|$, i lyin $v_j \prec_{v_i} u$. T , w v
$v_j \prec_u v_i$ n $v_j \prec_{v_i} u$ o ll $j \neq$, $0 \leq j \leq k$. T i n $\{u\ v_i\}$
will no in l in G_{kXTC}, on i in v_i i n i o o u
in G_{kXTC}.

5 Future Directions

T nn o i o G_{kXTC} in n x lo n v l in-
in ion on o l . Fo x l , k in G_{kXTC} o
o n n w x i o o nn o G. n o l
i iv n ny ≥ 1 n ni i G, w i $k = k(\ )$
G_{kXTC} i - nn o G. In i ion, on o l o on n o ni
i (o o in y i i in oin ni o ly n o in
o n l n ion) n inv i nn o i o G_{kXTC} in i
in . T i wo l l o n n ly i lly ovin on l ion,
x i n lly iv in [11], G_{XTC} i oo nn o n o ni
i .

Ano i ion w in in in i l xin ion
G i ni i n inv i in k-XT o o o n l
l o . Fo x l , [ ] n o l
y l i i “ lo no o li y o n xi in n wo .” W
in in inv i in o n o k-XT o i ni i
n o o li i n li ion o ni i .

Acknowledgment. W n T H n o ill in in i ion on l
wo l n o n - o wi l n wo .

References

1. M. Bahramgiri, M. Hajiaghayi, and V. S. Mirrokni. Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks. In *Proceedings of the 11th IEEE International Conference on Computer Communications and Networks (IC3N)*, pages 392–398, 2002.
2. M. Burkhart, P. von Rickenbach, R. Wattenhofer, and A. Zollinger. Does topology control reduce interference? In *Proceedings of the 4th ACM International Symposium on Mobile Ad-Hoc Networking and Computing (MOBIHOC)*, 2003.
3. F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proceedings of the 22nd ACM Symposium on the Principles of Distributed Computing (PODC)*, 2003.
4. F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs. In *DIAL-POMC 2003*, 2003.
5. L. Li, J. Halpern, P. Bahl, Y. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 264–273, 2001.
6. N. Li and J.C. Hou. FLSS: A fault-tolerant topology control algorithm for wireless networks. In *Proceedings of MOBICOM*, 2004.
7. D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *SenSys 2004*, 2004.
8. T. Moscibroda, R. O'Dell amd M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *DIAL-POMC 2004*, 2004.
9. R. Prakash. Unidirectional links prove costly in wireless ad-hoc networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIAL-M)*, 1999.
10. Y. Wang and X. Y. Li. Localized construction of bounded degree planar spanner for wireless ad hoc networks. In *Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing*, pages 59–68, 2003.
11. R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad-hoc networks. In *Proceedings of the 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN 04)*, 2004.

A Scheme Encouraging Mobile Nodes to Forward Packets via Multiple Wireless Links Aggregating System Between the Internet and Mobile Ad Hoc Networks

Yosuke Ito[1], Hiroshi Mineno[2], and Susumu Ishihara[3]

[1] Graduate School of Science and Technology, Shizuoka University,
3-5-1, Johoku, Hamamatsu, Shizuoka, 432-8561, Japan
Phone/Fax:+81-53-478-1265,
ito@ishilab.net

[2] Faculty of Information, Shizuoka University
mineno@cs.inf.shizuoka.ac.jp

[3] Faculty of Engineering, Shizuoka University
ishihara@ishilab.net

Abstract. We have proposed a system that achieves high-speed and high-quality communication between mobile nodes and the Internet by using multiple network interfaces of multiple mobile nodes. In this system, adjacent mobile nodes connect to each other with short-range high-speed links and establish temporary networks. A mobile node in a temporary network simultaneously uses multiple links owned by the nodes in the network when it communicates with nodes outside the network. In this system, a part of data packets for one node have to be relayed by the other nodes in the temporary network. However, other nodes might not relay data packets unless they receive some profit from their contribution. In this report, we introduce credits as an incentive to network nodes to relay packets. We propose a method that provides secure credit exchanges between nodes relaying packets and a node requesting the relays, and the method provides a trusted third party that assists those nodes exchanging credits.

Keywords: mobile computing, multiple paths, mobile IP, cooperation, incentive of forwarding, accounting, fairness, SHAKE, ad hoc network.

1 Introduction

In wireless communication environment, users demand to connect to the Internet comfortably at any time and place. In a previous report [1], we proposed SHAKE (a procedure for SHAring multiple paths to create a cluster networK Environment) to enable high-speed, reliable communication with multiple network interfaces for a temporal group of mobile devices. In SHAKE, mobile devices gathering in particular location establish a temporary network (we call this network an *alliance*) by establishing a short-range high-speed wireless link (e.g., wireless LAN). When a mobile device in an alliance accesses the Internet under a situation where the node has to use a slow link (e.g., 2G, 3G cellular), it uses not only its link to the Internet but also the links between

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 110–123, 2005.

the other mobile devices in that location and the Internet. This improves the data transmission speed, reliability and connectivity of the communication between the mobile devices and the Internet.

In SHAKE, nodes must assist other nodes by using their own external link to relay traffic. If nodes refuse a relay connection because they have to use their own CPU power, memory, and battery to relay traffic for other nodes, communicating by using the SHAKE will be impossible. To solve this problem, we propose a scheme that uses credits as an incentive to encourage nodes to relay traffic for other nodes.

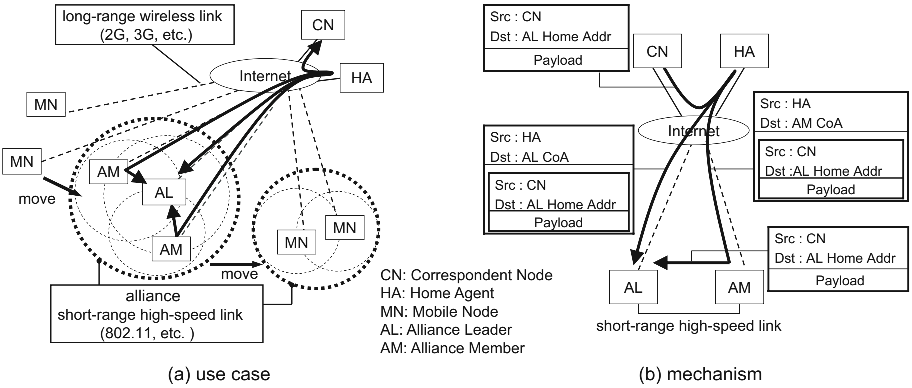
The rest of the paper is organized as follows. In Section 2, we review the SHAKE architecture, the issues, and the related work. In Section 3, we present architecture for motivating mobile nodes to perform relays. In Section 4, we discuss robustness and overhead of the proposed scheme. Section 5 summarizes this paper.

2 SHAKE

In this section, we provide an overview of SHAKE. In SHAKE, mobile nodes establish an alliance to enhance communication speed between them and the Internet. A node relaying data packets for another node in an alliance is described an Alliance Member (AM), and a node requesting the relay of data packets to AMs is described an Alliance Leader (AL). When an AL communicates outside of the alliance, it distributes traffic not only to its own external link but also to those of the AMs.

2.1 Mobile IP SHAKE

Mobile IP SHAKE [1] is an implementation of SHAKE on the IP layer. We assume the use of Mobile IP SHAKE in this paper. To establish SHAKE on the IP layer, a node that distributes traffic on the path between the correspondent node (CN) of an AL and the alliance including that AL is necessary. If no node distributes packets sent from the CN to the destination node (AL), the CN has to know all the addresses of the nodes in the alliance (AM). This is not ideal because it is not functionally practical for



(a) use case

(b) mechanism

Fig. 1. Mobile IP SHAKE

ordinary hosts on the Internet to know the addresses of all AMs for a short time session. Mobile IP SHAKE exploits a feature that assures that the packets from the CN to a Mobile Node (MN) always go through the Home Agent (HA) of an MN on the Mobile IPv4 mechanism unless route optimization is used, and introduces a traffic-dispersion mechanisms into the HA. For this reason, Mobile IP SHAKE does not require any special mechanism for CNs.

Figure 1 provides an overview of Mobile IP SHAKE. An AL registers an AM's care-of address (CoA) as well as the AL's own CoA to the HA of the AL in advance. When the HA forwards packets sent from the CN, it encapsulates the packets and distributes them not only to the AL but also to the AMs. The AMs decapsulate the transmitted packets and forward them to the AL through the links in the alliance. When packets are transmitted from the inside of an alliance to the external link, the AL encapsulates and distributes packets to each AM. Then, each AM forwards the packets to the destination node or the AL's HA as in the reverse tunneling technology used in Mobile IPv4.

In the following section, we describe transmission from a node in an alliance to outside the cluster as '*uplink*', and transmission from outside the alliance into the alliance as '*downlink*'.

2.2 Issues in Using SHAKE

In SHAKE, AMs have to offer CPU resources, battery power, and link bandwidth to the AL. For this reason, AMs may refuse to relay packets for the AL unless mutual trust exists between the AL and the AMs or unless some reward is promised. Therefore, we introduce a mechanism for motivating AMs to relay packets for the AL by granting rewards to the AMs. We deal with this issue in this paper. Adding to this, the management of heterogeneous mobile nodes in the alliance and traffic distribution are important issues. These issues have been discussed in [2].

2.3 Related Work

The issues of cooperation of mobile nodes for packet forwarding have been investigated in ad hoc networks and multi-hop cellular networks. In [5, 7], reputation mechanisms for ad hoc networks were proposed. In [10], Eidenbenz et al. proposed game theory approach in ad hoc networks. Golle et al. analyzed the incentives in peer-to-peer networks [9].

Our approach for our special architecture SHAKE is credit-based mechanism. Credit-based mechanism is used in ad hoc networks [3, 6, 4], and in multi-hop cellular networks [11, 8]. In [3], Buttyan and Hubaux proposed virtual currency called *nuglets*. The sender of a packet loads *nuglets* on the packet, and the intermediate nodes acquire some *nuglets* from that packet by forwarding it. In [6], they proposed an improved mechanism. In [3, 6], to ensure the payment of the correct amount of *nuglets* to each node, tamper-proof hardware is used. Our system does not need any tamper-proof hardware at any node. Zhong et al. proposed a method relying on a central authority that collects receipts from the forwarding nodes [4]. In this method, intermediate nodes send receipts after forwarding data messages. Then, the central authority charges the source nodes and rewards the forwarding nodes based on the receipts.

The following are differences between these credit-based methods and our method. First, in [4, 11], authors use cryptographic functions based on public key cryptography, whereas our solution is based on symmetric key cryptography requiring less computation load. Secondly, some of above credit-based approaches do not solve a case in which the destination of a packet pays the reward. When SHAKE is used, an AL has to grant AMs the rewards in both cases when the AL is the transmission source and the destination, because the AL relies on AMs in both cases. So we designed a mechanism adapted to the both cases. Thirdly, these approaches assume only a rational malicious node that attempts to cheat if the expected benefit of doing so is greater than the expected benefit of acting honestly. In other words, they do not take care of the offenders for pleasure. We suppose that the existence of such offenders is one of serious problems. In addition to the cases that malicious nodes attack the system intentionally, cracked computers might attack other hosts unintentionally. This leads to collapse the systems and to loss service provider's confidence. The fourth difference is that above credit-based methods can not distinguish unintentional packet losses from packet drop of malicious node, and can not solve contradiction of charging arisen from packet losses. We also address this problem.

3 A Scheme Encouraging Mobile Nodes to Forward Packets on SHAKE

To encourage Alliance Members (AMs) to relay packets for an Alliance Leader (AL) in SHAKE, we introduce an incentive for AMs. We propose a method of using credit as the incentive. Each AM receives credit in compensation for the relay. Therefore, if an AL wants to send packets via an AM, the AL needs to pay credit for the AM. The amount of credit is proportional to the size of the packet. We assume that the credit can be converted into real money or can grant privilege to users in provider services. If a node wants to get more credit, the node can get by paying its debit or buy them using real money, or be remunerated by forwarding others' data traffic.

We introduce a trusted third party to maintain users' credit account, and we call this party a Credit Server (CS). We assume that the CS and the Home Agent (HA) are completely reliable and do not coalesce with other hosts. From a practical standpoint, HAs will be managed by ISP or carrier if Mobile IP is used for mobile phone. Because of this, it is considered to be reasonable that the HAs are completely reliable. The CS is the authority for managing credit, and the CS rewards AMs that have forwarded packets reliably and charges the ALs. The CS charges and rewards for the relay of packets forwarded successfully. We use Forward Reports (FRs) from an AM and a HA for judging whether packets have been successfully forwarded. Between the HA and the CS, and between the AM and the CS, the FRs are assumed not to be modified by a third party by using secure session like IPsec.

In the following discussion, we deal with the following malicious attacks.

- Forgery of credit:
  Individual nodes may illegitimately try to increase their own credit.

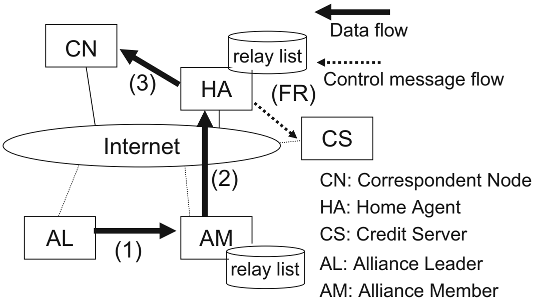
- Free riding (AL's refusal to pay to CS):
  An AL may claim that it did not initiate some communication despite being helped by AMs. The CS has to refuse these kinds of claims.
- AM's false charge for rewards:
  An AM may charge credit by sending a false FR to the CS. The CS has to refuse such kinds of charges.

3.1 Forwarding Uplink Packets

Overview. Figure 2 illustrates the flow of data packets and control messages for crediting procedures in uplink. In Section 2.1, in uplink on Mobile IP SHAKE, we pointed out that both the transitions of passing through HA and of not passing through HA could be used. However in this paper, we assume that packets from an AL to a CN are forwarded by the HA. The purpose of this is that we intend to enable the HA to confirm that AMs forward packets with certainty.

In uplink, the packets from an AL are delivered to the CN via an AM and an HA except packets sent directly from that AL's own external link. When the HA forwards a certain amount of packets via the AM, it generates a FR and sends it to the CS. We suppose that the HA sends the FRs to the CS via TCP for reliable transmission. When the CS receives the FRs from the HA, it judges whether each packet has been successfully forwarded. After this operation, the CS pays the reward to the AM and charges it to the AL for the successfully forwarded packets. This CS's payment is supposed to be levied as ISP or other service charges. When no FR is received from the HA, the CS judges that forwarding has failed, and does not charge or reward credit.

Protocol in Detail. In this section, we present details of the uplink protocol. The packets from an AL to a CN are distributed to a communication path via an AM (AL → AM → HA → CN) and another communication path using the AL's own external link (AL



relay list (=Forward Report (FR))

charge	reward	seID	seq	length
AL	AM	30	221	1500
AL	AM	30	222	1500

(1) $(\boldsymbol{payload}, \boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{MAC}_{K_{HA\text{-}AL}}(\boldsymbol{payload}, \boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}))$

(2) $(\boldsymbol{payload}, \boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{MAC}_{K_{HA\text{-}AM}}(\boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{MAC}_{K_{HA\text{-}AL}}))$

(3) $(\boldsymbol{payload})$

Fig. 2. Uplink procedure

$\rightarrow$ HA $\rightarrow$ CN). Charging or rewarding credits is not processed for packets delivered directly from an AL to the HA rather than via an AM. Hereafter, we explain the protocol relating to the crediting procedure on the communication path via the AM.

To authenticate the sending node and the forwarding node of a packet, we use a message authentication code (MAC). In our proposal, an AL sends a packet with a session ID (*seID*), sequence number (*seq*), length (*length*) and its MAC. After an AM receives the packet, it forwards the packet to the HA with a new MAC computed with the MAC included in the received packet. The HA verifies the MAC in the received packet. The *seq* is used to resist replay attacks. The *length* is used for charging at the CS.

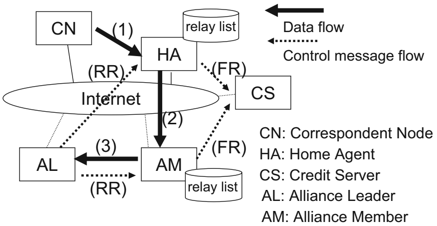
Symmetric session keys (K_{HA-AM}, K_{HA-AL}) must be established in advance through a suitable key exchange protocol between an HA and an AM via an AL, and between the HA and an AL, respectively. Hereafter, $MAC_{K_{HA-AM}}$, $MAC_{K_{HA-AL}}$ denote MACs, which are the keyed cryptographic hash values computed with the session key between the HA of the AL and an AM, and between the HA and the AL, respectively. Moreover, we assume that HAs have a *relay list* including a list of packets relayed for the AL. This *relay list* is used for generating FR for multiple relayed packets.

We explain the crediting procedure on Mobile IP SHAKE as described in Figure 2.

1. An AL generates the *seID* of the session, and distributes the packets to the AMs with their *seID*, *seq*, *length*, $MAC_{K_{HA-AL}}$. $MAC_{K_{HA-AL}}$ is the keyed cryptographic hash value of the content of the packet (i.e. *seID*, *seq*, *length*, *payload*) (Figure 2(1)). The session key K_{HA-AL} is used for computing $MAC_{K_{HA-AL}}$.
2. An AM receives the packet from the AL. It checks that the sequence number has not already been used. If the packet is not duplicated, the AM computes a new MAC with the received MAC and K_{HA-AM}, and forwards the packet to the HA adding the $MAC_{K_{HA-AM}}(seID\ seq\ length\ MAC_{K_{HA-AL}})$ instead of the received MAC (Figure 2(2)).
3. The HA verifies whether the value of MAC added to the packet is correct by comparing it with the keyed cryptographic hash value using K_{HA-AL} and K_{HA-AM} stored in the HA. If it is not correct, the packet is dropped. Otherwise, the HA checks that the sequence number has not already been used. If the packet is not duplicated, the HA forwards the data packet to the CN (Figure 2(3)). After forwarding the packet, the HA adds the entry including the *seID*, *seq* and *length* of each packet to its *relay list*.
4. The HA sends a FR based on each *relay list* to the CS periodically or when the number of unsent entries of *relay list* reaches the upper limit (Figure 2).
5. The CS charges and rewards credit according to the amount of the packet reported from the HA.

3.2 Forwarding Downlink Packets

Overview. Figure 3 shows the flow of the data packets and the control messages in the crediting procedures in downlink. A data packet is delivered from the CN to an AL via the HA of the AL and an AM. When an AL receives a certain amount of packets



(1) $(\boldsymbol{payload})$

(2) $(\boldsymbol{payload}, \boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{MAC}_{K1_{HA\text{-}AL}}(X))$

(3) $(\boldsymbol{payload}, \boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{MAC}_{K1_{HA\text{-}AL}}(X))$

$(X) = (\boldsymbol{seID}, \boldsymbol{seq}, \boldsymbol{length}, \boldsymbol{payload})$

relay list

charge	reward	seID	seq	length	$MAC_{K_{HA\text{-}AM}}(X)$
AL	AM	50	121	1500	sdfiou3...ew
AL	AM	50	122	1500	eew3w...5er

Receive Report (RR)

charge	reward	seID	seq	length	$MAC_{K2_{HA\text{-}AL}}(X)$
AL	AM	50	121	1500	sdfiou3...ew
AL	AM	50	122	1500	eew3w...5er

Forward Report (FR)

charge	reward	seID	seq	length	$MAC_{K_{HA\text{-}AM}}(X)$	$MAC_{K2_{HA\text{-}AL}}(X)$
AL	AM	50	121	1500	sd3asfsaf...dsf	sdfiou3...ew
AL	AM	50	122	1500	dsafasf23...5d	eew3w...5er

Fig. 3. Downlink procedure

forwarded by the AM, the AL sends a Receive Report (RR) of the forwarded packets to the HA and the AM. The RR is a list that contains the *seID*, *seq*, *length*, and $MAC_{K_{HA-AL}}$ of each packet. This RR is essential for confirming the success of forwarding. The RRs, as well as the FRs, are supposed to be sent through TCP connections. We assume that an AM and a HA require a $MAC_{K_{HA-AL}}$ contained in the RR to generate a FR. The AM and the HA generate their FR based on both the RR and their *relay list*, and send it to the CS. The CS compares the FRs from both the AM and the HA, and confirms that the packets were actually forwarded by the AM and the HA. If the FR is correctly collated, the charging and rewarding procedure is performed.

Protocol in Detail. In this section, we explain the details of the protocol in downlink according to Fig. 3. Symmetric session keys (K_{HA-AM}, K_{HA-AL}) are established in advance through a suitable key exchange protocol between the HA of the AL and an AM via the AL, and between the HA and the AL, respectively. Between the HA and the AL, two symmetric session keys are established. We name them $K1_{HA-AL}$, $K2_{HA-AL}$, respectively. One is used for authentication in communication, and the other is used for RRs.

1. The CN transmits the data packets destined for an AL (Figure 3(1)).
2. When the HA of the AL forwards the data packets to the AM, the HA generates the *seID* of the session, and attaches the *seID*, *seq*, *length* and $MAC_{K1_{HA-AL}}$ of the received packet (Figure 3(2)). After forwarding the packet, the HA adds an entry that consists of the *seID*, *seq*, *length* and the calculation result of $MAC_{K_{HA-AM}}(seID\ \ seq\ \ length\ \ payload)$ to its *relay list*.
3. The AM receives the packet and then checks whether the sequence number has not already been used. The AM forwards the packet to the AL. After forwarding the packet, the AM adds an entry that consists of the *seID*, *seq*, *length*, and $MAC_{K_{HA-AM}}$ of the forwarded packet to its *relay list*.
4. When the AL receives the packet forwarded by the AM, the AL verifies whether the value of $MAC_{K1_{HA-AL}}$ added to the packet is correct. If the verification is successful, the AL adds an entry including the *seID*, *seq*, *length*, and the calculation result of $MAC_{K2_{HA-AL}}(seID\ \ seq\ \ length\ \ payload)$ of the packet to its RR. When the AL

receives a certain amount of packets, the AL sends a RR to the HA and the AM. The HA and the AM generate FRs based on the RR and the *relay list* maintained by themselves, and send them to the CS. The content of the FRs is a list including the entries of the set of *seID*, *seq*, *length*, $MAC_{K_{HA-AM}}$ and $MAC_{K2_{HA-AL}}$ of forwarded packets as in Fig. 3.

5. The CS compares the entries in the FRs of the AM and the HA, and judges whether the packets were successfully forwarded. The CS charges and rewards credits according to the amount of correctly forwarded packets.

3.3 Mechanisms for the CS/HA to Resist Dishonest Claims

In order to resist dishonest claims by ALs and AMs, the CS and the HA perform the following procedures.

CS Operation. We assume that some CSs exist in the Internet. The CSs maintain a list of malicious nodes by mutually exchanging information or using a centralized information server. Specifically, the CSs record the nodes that refused to pay credits or falsely charged for rewards, and this information is shared by CSs. We assume that this information can be referred to all nodes when an alliance is established, and thus can be used to evaluate whether nodes are suitable to be included in an alliance. In addition, when the wrong MACs are submitted to the CS, the CS sends error messages containing the wrong MACs to the AL and the AM, which announces to the AL and the AM that the wrong MACs are sent. The error messages are assumed not to be modified by a third party by using secure session like IPsec.

HA Operation. In downlink, ALs are supposed to sent the RR to the HA and the AM if in fact they have received a packet from the CN via the HA and the AM. However the RR may not be sent from the AL in the following two cases. One is when the AL does not send the RR intentionally. The other is when some accidents occur on the link between the HA and AL via the AM, and the data packets from the CN do not reach the AL via the HA and the AM. In either case, we assume that the HA stops distributing packets to the route from which an RR was not delivered for a certain amount of time, and the HA distributes packets to other AMs' routes. Moreover, in the same way as CS, the HA sends error messages to the AL and the AM if wrong MACs are sent in uplink.

4 Analysis

In this section, we analyze the robustness and overhead of our proposed method.

4.1 Robustness Against Attacks of Malicious ALs

Here we consider the robustness against malicious attacks by Alliance Leaders (ALs).

Refusal of Payment by AL

*Dishonset Act.*An AL may refuse a payment claim from the Credit Server (CS) although it was actually supported by Alliance Members (AMs).

*Solution.*If the AL refuses payment, the CS records that the AL refused the payment. Because this recorded information is publicly open to the other nodes, the AL cannot maintain the confidence of other nodes using SHAKE afterwards. The payment refusal of the AL is prevented because payment refusal becomes disadvantageous when using SHAKE.

Transmission of Incorrect MAC by AL

Dishonest Act. In uplink, in order to escape charges, an AL might transmit a false MAC to the AM and the HA.

Solution. In uplink, the AL transmits $MAC_{K_{HA-AL}}$ with each packet. The MAC can be verified in the HA although it cannot be verified in the AM, because the MAC is made from the session key between the AL and the HA. If the HA's verification of the MAC is unsuccessful, the HA will drop the packet. Transmission of an incorrect MAC results in the packet undelivered to the CN. Therefore, the ALs will not transmit incorrect MACs.

Undelivered RR

Dishonest Act. In downlink transmission, an AL may not submit a Receive Report (RR) although it received packets via the AM accurately.

Solution. The HA will stop distributing packets to any route from which an RR is not submitted as described in Section 3.3. If an AL maliciously refuses to submit RRs, the HA will stop delivering packets to routes from which RRs are not submitted, and so the route will not be used. Therefore, all ALs will submit the RR faithfully if they want to use the route effectively.

Incorrect RR Submission

Dishonest Act. In two cases, incorrect RR may be submitted from an AL to refuse charging. One is that the AL submits incorrect RRs both to the HA and the AMs. The other is that the AL submits an incorrect RR either to the HA or the AM. The RRs can be verified in the HA though they cannot be verified in the AM, because the MACs contained in each entry of the RRs are made from the session key between the AL and the HA. Therefore, a problem exists when the AL submits a correct RR to the HA and an incorrect RR to the AM.

Solution. The Forward Report (FR) is supposed to contain the MAC generated by AL in the RR. If the CS cannot collate the FRs from the HA and the AM correctly, the CS sends error messages to the AL and the AM as described in Section 3.3. If the AM receives the error message, it stops forwarding of packets for the AL.

4.2 Robustness Against Malicious Attacks by AMs

In this section, we consider the ability of the proposal method to resist the malicious attacks of AMs.

Dishonest Rewards

Dishonest Act. An AM may charge for a reward for packets that it did not forward.

Solution. In uplink, if an AM wants to be rewarded, it must actually forward packets to the HA. The HA sends a FR to the CS for the only packets that arrived at the HA. Thus the AM cannot receive a reward for packets that it has not forwarded.

In downlink, the AM must send a FR to the CS to be rewarded for the forwarding of packets. When the AM generates the FR, the AM cannot generate the required MAC for the FR by itself because it needs the MAC computed for the forwarded packet with the session key between the HA and the AL owned by HA and AL. The MAC is included in a RR from the AL. Therefore, in downlink, an AM can generate a FR only when it has actually forwarded packets for the AL to the CN and received the corresponding RR from the AL.

Packet Drop in Forwarding
Dishonest Act. AM may intentionally drop packets for AL. *Solution.* An AM can easily drop packets intentionally. However, if a packet does not reach its forwarding destination node, the AM cannot be remunerated. If an AM drops only a few packets, the influence on the communication performance is a little, and this is common in mobile environment. Thus, any special operation is not performed. If the packet loss continues, the HA and the AL stop distributing packets to the route that is dropping packets as described in Section 3.3. Therefore, this dishonest act is insignificant.

Modification of MAC Generated by AL
Dishonest Act. In uplink, an AM forwards packets including MAC generated by the AL. In downlink, an AM receives a RR from the AL, then the AM sends a FR containing MAC generated by the AL and included in the RR. The AM can modify the AL's MACs, which intentionally damages the reputation of the AL.

Solution. If the CS and the HA receive wrong MACs, they send error messages to the AM and the AL as described in Section 3.3. If the AL receives the error message, it stops the distribution of packets to the AM and breaks the alliance with the AM.

4.3 Robustness Against Malicious Attacks by a Third Party

We assume that a HA is completely trustworthy. All dishonest behavior resulting from a conspiracy can be prevented by the tact that all packets must pass through the HA. For instance, if the HA cannot identify a third party that colludes with an AM or an AL, the HA does not forward packets and does not send a FR to the CS. Therefore, problems do not occur.

4.4 Robustness Against Malfunction Caused by Lost Packets on Links

In transmission on wireless links, packet loss may occur unexpectedly. We point out losses of data packets, and do not discuss losses of the RR and FR that are supposed to via TCP flow. We consider instances of both the uplink and downlink, and discuss the charging and rewarding rather than the influence on communication performance.

Uplink (AL → AM → HA → CN)

- Packet loss between AL and AM
  If the packet destined for the CN does not arrive at the AM, crediting procedure is not performed, and therefore problems do not occur.
- Packet loss between AM and HA
  In this case, the packet destined for the CN does not reach the HA even if the AM

actually forwards the packet to the HA. If the packet does not reach the HA, the AM cannot be remunerated though it was actually willing to forward the packet. We assume that the AM can be remunerated only when packet forwarding has succeeded. Contradictions related to rewards do not occur.

- Packet loss between HA and CN
  In this case, a packet from an AL to the CN does not reach the CN. However, an AM has in fact successfully forwarded the packet to the HA, and therefore the AL should send a reward to the AM. We assume that the AL sends the reward to the AM that has forwarded the packet successfully, and that the CS can confirm the AM's forwarding via the HA's FR. Therefore, contradictions related to rewards do not happen.

Downlink (CN → HA → AM → AL)

- Packet loss between CN and HA, Packet loss between HA and AM
  The charging and rewarding of credit will not occur if a packet does not reach the AM. Neither kind of packet losses causes problems.
- Packet loss between AM and AL
  In this case, even if an AM certainly forwarded the packet to the AL, whether the packet is dropped by the AM intentionally or not cannot be distinguished. In our proposed method, if an AM's successful forwarding cannot be confirmed, the rewarding procedure is not performed. Therefore, the contradictions to the rewarding procedure do not occur.

4.5 Overhead

SHAKE is a mechanism aiming at the improvement of the communication performance by using two or more links simultaneously. To maintain the very small overhead for the crediting procedure compared with the communication performance improvement is essential.

Computation Overhead. For each packet, MAC computations and MAC verifications have to be performed at the HA, AM, and AL. Cryptographic operations need energy and time to be performed. Regarding energy consumption, the energy required to perform the computation is negligible compared with the energy required to perform the transmission [12]. Moreover, the time required to compute the cryptographic hash function is also efficient. [13] shows numerical examples of speed benchmarks for some of the most commonly used cryptographic algorithms. For example, when being run on a Pentium 4 2.1 GHz processor under Windows XP SP 1.386, a MAC computation with HMAC/MD5 algorithm can be performed at 1.6 Gbps. According to this value, the MAC computation time for 1500 bytes packet would be approximately 7 microseconds. In the measurements of the Mobile IP SHAKE that we previously implemented, the time required to perform the forwarding at the HA was approximately 250 microseconds, the time required to perform the forwarding at the AM was approximately 24 microseconds. The MAC computation time (7 microseconds) is negligible compared with the forwarding time. Therefore, the overhead of MAC computations and MAC verifications at the HA, AM, and AL is acceptable.

Communication Overhead. In order to measure the communication overhead of each data packet, we have implemented a prototype of our scheme by adding an authentication header per packet on our Mobile IP SHAKE ([1]). We implemented this prototype on Linux, and realized the authentication header as a part of IP options. As mentioned in Section 3, we need seID, seq, length, and MAC for each packet. In this implementation, the MAC is computed by HMAC-MD5. In IP options format, option fields (1 byte) and option length field (1 byte) are prepared. We added *seID*(1 byte long), *length* (that is packet length, 1 byte long), *seq* (4 bytes long) and MAC (16 bytes long) to the IP options format. Additionally, we attached a 2-bytes long SPI (Security Parameter Index) field for the authentication algorithm and a 2-bytes long padding field. Thus, the total length of the authentication header is 28 bytes long.

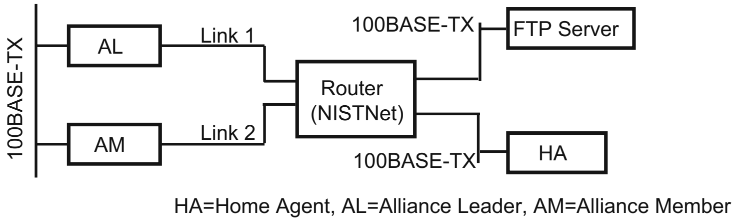
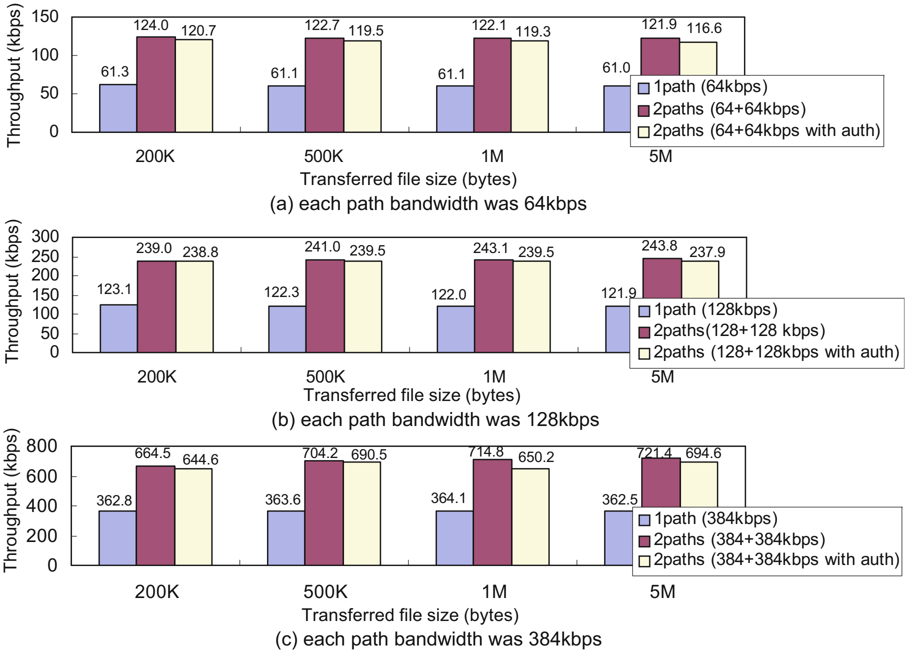


Fig. 4. Experimental network for emulating wireless network



(a) each path bandwidth was 64kbps

(b) each path bandwidth was 128kbps

(c) each path bandwidth was 384kbps

Fig. 5. Performance on an emulated network. Average of five trials

We measured the effect of the additional header to the throughput of the communication on the Mobile IP SHAKE. Figure 4 shows the network topology of the experimental network. An AL and an AM that have two fast Ethernet interfaces were connected, and they were also connected to a router with multiple network interfaces and runs NISTNET [14] network emulator. The HA of the AL and an FTP server were connected to the different network interfaces of the router. We measured the throughput when 200KB, 500KB, 1MB and 5MB files were transferred from the FTP server to the AL. The bandwidth and delay of the link between the AL and the router, and the AM and the router were set to 64, 128, 384 [kbps] and 100 [msec], respectively. The distribution ratio of AL to AM in the HA was 1:1. A case where only one AL was connected to the router was also tested for comparison.

Figure 5 shows the result of the experiments. We can see the influence to the throughput by addition of the authentication header is negligible. For example, in Fig. 5 (b), when 5MB file was transferred by normal Mobile IP SHAKE and the Mobile IP SHAKE with authentication header, the throughputs were 243.8 kbps and 237.9 kbps respectively. The ratio of 243.8 kbps to 237.9 kbps is 97.6 %. The communication overhead seems to be acceptable.

Other Overhead. In our system, the additional messages are required to establish symmetric keys between the HA and AL, and between the HA and AM. This is performed only once in a session.

The HA and AM need to send forward reports, and the AL needs to send receive reports. We assume that the reports are not sent for each packet, but sent only when the entry reaches the some degree. Thus, we consider that the influence to the communication performance will be negligible.

4.6 Use in Heterogeneous Environments

Until this point, we did not take into consideration the situations that the costs of forwarding packets are different among the nodes in the alliance, e.g., a case that an AM is a PC and the other AMs are PDA or mobile phones. The ratio of CPU power, memory and battery consumption for forwarding packets may be different among AMs. Besides, each AM may connect to different mobile carriers. The delay and bandwidth of each link and fee structure depend on the mobile carrier. Thus, in such heterogeneous environments, we have to take into account the differences and reflect them in the accounting rule of CS, and we should make the dispersion rule for HA and AL in consideration of the accounting rule and the link status of each AM.

5 Conclusion

In this paper, we addressed a problem of motivating mobile nodes to forward packets on SHAKE. To solve this problem, we proposed a method to provide rewards for nodes that forward packets. We introduced a trusted third party that functions as a credit server and manages members' credit accounts. We presented a charging/rewarding method based on Receive Reports (RRs) and Forward Reports (FRs). By using RRs and FRs, the charging/rewarding procedure works only for successfully forwarded packets, and our method can resist several kinds of dishonest attacks. Moreover, we showed that our system works even if unexpected packet losses occur on the link. We implemented the prototype of our system that deals with extension header per packet, and evaluated the communication overhead of the extension header. The result showed the overhead is acceptable for the use of SHAKE.

References

1. K. Koyama, Y. Ito, S. Ishihara, H. Mineno, “Performance evaluation of TCP on Mobile IP SHAKE,” IPSJ journal, Vol. 45, No. 10, pp. 2270–2278, 2004.
2. H. Mineno, Y. Konishi, S. Ishihara, and T. Mizuno, “Implementation of cluster control manager for multiple wireless links sharing system,” in proc. of PACRIM, 2003.
3. L. Buttyan and J.-P. Hubaux, “Enforcing Service availability in mobile ad-hoc WANs,” in proc. of MobiHoc, 2000.
4. S. Zhong, Y. R. Yang, and J. Chen, “Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad Hoc Netoworks,” in proc. of INFOCOM. IEEE, 2003.
5. S. Marti, T. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in proc. of MobiCom, 2000.
6. Levente Buttyan, Jean-Pierre Hubaux, “Stimulating cooperation in self-organizing mobile ad hoc networks,” Mobile Networks and Applications, v.8 n.5, p.579-592, 2003
7. Sonja Buchegger, Jean-Yves Le Boudec, “Performance analysis of the CONFIDANT protocol,” in proc. of MobiHoc, 2002.
8. N. Ben Salem, L. Buttyan, J. P. Hubaux, and M. Jakobsson, “A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks,” in proc. of MobiHoc, 2003.
9. P. Golle, K. Leyton-Brown, and I. Mironov, “Incentives in peer-topeer file sharing,” in proc. of the ACM Symposium on Electronic Commerce (EC’ 01) 2001, 2001.
10. Luzi Anderegg and Stephan Eidenbenz, “Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents,” in proc. of MobiCom, 2003.
11. B. Lamparter, K. Paul, and D. Westhoff, “Charging Support for Ad Hoc Stub Networks. Journal of Computer Communication,” Technology and Applications, Elsevier Science, 2003.
12. G. J. Pottie and W. J. Kaiser. Wireless Integrated Network Sensors. Communications of the ACM, May, 2000.
13. Speed Comparison of Popular Crypto Algorithms, http://www.eskimo.com/˜weidai/benchmarks.html
14. NIST Net, http://snad.ncsl.nist.gov/itg/nistnet

A Protocol for Recording Provenance in Service-Oriented Grids

P l o , Mi l L , n L Mo

School of Electronics and Computer Science,
University of Southampton, Highfield,
Southampton SO17 1BJ, United Kingdom
{pg03r, mml, l.moreau}@ecs.soton.ac.uk

Abstract. Both the scientific and business communities, which are beginning to rely on Grids as problem-solving mechanisms, have requirements in terms of provenance. The provenance of some data is the documentation of process that led to the data; its necessity is apparent in fields ranging from medicine to aerospace. To support provenance capture in Grids, we have developed an implementation-independent protocol for the recording of provenance. We describe the protocol in the context of a service-oriented architecture and formalise the entities involved using an abstract state machine or a three-dimensional state transition diagram. Using these techniques we sketch a liveness property for the system.

Keywords: recording provenance, provenance, grids, web services, lineage.

1 Introduction

A i i y oo in o ion l o no j o n-
li on ol in n , o n, n l- o o o ol n in
o liv non- ivi l li i o vi [ ]. By oo in in iv , i i
o ion l o , i n o l - l o l
i o wi yon o o lo l, o o no y . i
in v lo o n wi v i y o li ion o o in n
i n o ni i . S i n i li ion in l n ly i o o
L H on olli (l .w . n. /L /), x i n in -
i y [3] n n x n ion li . i in in
o ni y o o i i l ion, i i i in ol in-
y, n o ovi o olio o n ion in n n i l vi
(www.i . o / i).

T o ni i l o v i n in o . W n
ov n n o o o n ion o l o
. T n i y o ov n n i n in wi n o l .
Fo x l , A i n Foo n D A ini ion i
ov n n o ’ i ov y lon i in (o 50

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 124–139, 2005.

y o i). In i y, ov n n i o il o y w i i l i n , llowin i l o n . In o , i l ion o w ll o ov n n i o o 99 y i n o n i . In n n i l i in , A i n S n - xl y A i li o n in o in in ov n n o n i o o l v n y i o o (Uni S P li L w No. 10 - 0). In i in , ov n n o n o n i vi l o i ff iv n n l n ion. T j o x l o i n o ov n n in i n n in . P ov n n i i l ly i o n w n i no y i l o in o ly i n i o .

iv n n o ov n n in o ion n n o i in o nnin jo li ion , o l i y o lly y i o ni y, n ly, ow o o ov n n in i ? So o n - o ol ion v n v lo o l o ov n n o in ili y wi in on x o i i li ion . Un o n ly, i n ov n n y nno in- o . T o , in o i ili y o o on n v n ov n n o in . F o , n o o on n o o in ov n n v lo n o li ion i in ov n n o in o o - li n on o .

Ano w o n o ol ion i in ili y o ov - n n o y iff n i . v n wi v il ili y o ov n n - l o w o on n , o l o in ov n n in o ion will no i v . To i o l , n o l v lo o ow ov n n in o ion i o , n , n . S n wo l llow ov n n o o li ion , ov n n o o- n n , n i , in ov n n in o ion o i l n v l - l . In y, i y o n , o on n , n ni o o in ov n n i o l n o y i o ni y. T o o i wo , v lo n o n l i n o o ol o o in ov n n , i ow in o l .

T o i o ni ollow : S ion n o - i n ov n n y o l . T n, S ion 3 o lin i n o ov n n o in y in on x o vi -o i n i- . T y l n o o y i P ov n n o in P o o ol i in S ion . In S ion 5, o in y o li , n o li ion n , in S ion 6, o iv o i o n o i o y . Fin lly, S ion i l wo , ollow y on l ion. iv n l n o i , w i il- i wi i , Vi l ni ion (V), W S vi , n vi -o i n i (S A).

2 Requirements

W v i n i n o i n ov n n y o l o o n ini i l i n in o . T ollowin v n i n v no i l i o n in o iv in v lo n o o i n o o ol.

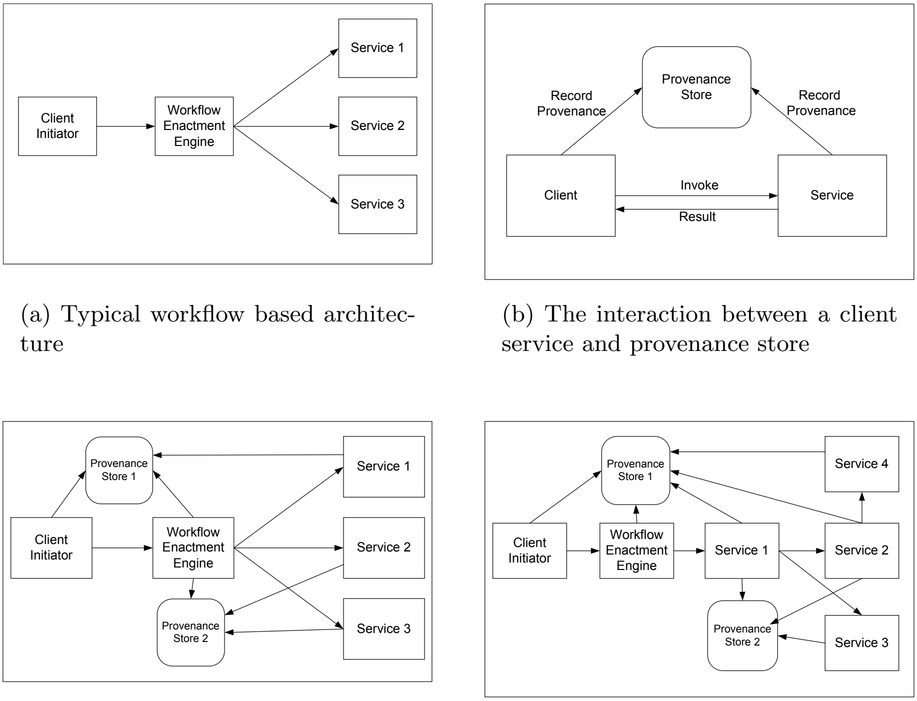
1. Verifiability. A ov n n y o l v ili y o v i y o in o o involv , i ion n i l ion i wi on no .
2. Accountability. lo ly l o v i ili y i o n ili y. An o o l o n l o i ion in o . T o , ov n n y o l o in non- i l nn ny ov n n n y n o .
3. Reproducibility. A ov n n y o l , ini , l o o n o i ly o o o ov n n i o .
4. Preservation. A ov n n y o l v ili y o in in ov n n in o ion o n x n io o i . T i i vi l o li - ion n in V on x v n V i n , ov n n will y i lly n o in in .
5. Scalability. iv n l o n o i li ion n l , in o in o o L H on olli , i i n y ov n n y l l . Ano on o l ili y i ov n n in o ion y l n o o n li ion.
6. Generality. i i n o o wi v i y o li ion , o , ov n n y o l n l no o o ov n n o v yin li ion .
7. Customisability. To llow o o li ion i o ov n n in o ion, ov n n y o l llow o o i ion. A o - o i ili y o l in l on in on y o ov n n o , i on in on w n o in n l , n n l i y o ov n n o o .

Wi i n in in , w now il o on l i o o in ov n n in S A.

3 Conceptual Architecture

Fi 1( ) ow y i l wo flow vi -o i n i . A li n ini i o invo wo flow n n n in w i , in n, invo v io - vi on wo flow i y ini i o , n lly, l i n o ini i o . In n , i n o n own in o wo y o o : li n w o invo vi n vi iv invo ion n n l .

iv n y o o n i o o o ni ion, w v i n i wo in o ov n n xi in vi -o i n i .



(a) Typical workflow based architecture

(b) The interaction between a client service and provenance store

(c) Workflow based architecture with provenance recording

(d) Architecture with provenance recording and services invoking other services

Fig. 1. Architecture diagrams

T in o ov n n i in ion ov n n . Fo o , in -
ion ov n n i o n ion o in ion w n o l
o . In S A, in ion , n n lly, li n invo in -
vi . T o , in ion ov n n n o in y o in in
n o o v io vi involv in n in l . T on
y o ov n n w v i n i i o ov n n . Fo o , o
ov n n i o n ion n only ovi y i l o
inin o o l o .

Wi in on x o in o ov n n , o i in o
i y o o , ov n n o .

Third Party Provenance Stores. W i y ov n n o y
o l llin i n o lin ov . In o v ion, l in
n o in inin ov n n on i y o n n i
li n no vi in in ov n n in o ion yon o o
ny iv n li ion n. An i ion l n o i y ov n n o
i y ovi o o l y li ion o in in ov n n .

In o o n n ow ov n n o l o o i n , w now x l in o in o o in ion ov n n .

A Triangle of Interaction. i o in ion ov n n in ollowin nn . Fo in ion w n li n n vi , oni in o n invo ion n l , y i i o i i vi w o in ion o o on ov n n o . v n o o i on i l i l o , in ion w n ll o n own o o on ‘ i n l ’ n o in ion i ov n own in Fi 1( ). T i ion i o i l o y on in only y o o , li n , vi n ov n n o , w o xi in o o o o y o i l on - o-on in ion o li n n -vi . T in ion o o i ov n y P ov n n o in P o o ol, w i w will il l in .

Uses of Interaction Provenance. T li n - vi in ion o i o in ion ov n n o o . T i in ion ov n n n o o v n o o l o . Fo x l , i vi involv in o v no n , in o vi , o in ov n n , n o invo o on in vi o in o . o in ion ov n n in l ol in o o o n o i in n o n o v i ion o o .

The case for Recording Two Views. How v , i in ion ov n n i o o o ion, o n ili y o v i ion o , in ion o on y o involv . o i own vi w o n in ion, w i , i o i , i in n o o o in n in ion. T o , in o i li n n vi i i vi w o n in ion o o on ov n n o , w i n n o on i in ion. Wi o v i ion y ov n n o , v l o l o l i , i l ly in o n nvi on n . Fo x l , o l li n only y o in in ion in ov n n o , vi wo l li n on li n o i ov n n . In , wi o i ion o ov n n o vi , wo l no vi n li n invo vi o l li n no o in ion. iv n vi n l o n l o i ion o in ov n n o , i i n l . In o y , ov n n o wo l now vi w invo vi i in o ion. T o l wo l l o xi in w vi w only y i in o ov n n o . W no i n o i i i vi w o no v n oll ion w n i , i o llow ov n n o o w n wo i i o o o n in ion.

Multiple provenance stores. Al o li n n vi i o o on ov n n o o n in ion, iff n ov n n o n o iff n in ion v n w n li n n vi .

Fi 1( ) ow y i l wo flow i wi l i l ov n n
o . T i i i l o ‘ i n l ’ n i in
Fi 1( ). n n o l i l ov n n o i li in ion o
n l oin o il . Ano n i n i o l i l
vi in in i ’ o n . T n l o
l ili y i n .

Advanced Architecture Support. A w ll o in y i l wo flow
n n i , o y o o v n i -
li on own in Fi 1( ). In i i , vi invo
o vi o o l , in on o vio i w
wo flow n n n in w only o invo in vi . In o
o in in ov n n o ov n n o , li n n o in o
o i in l ov n n o w n i n w ov n n o . Fo x l , in
Fi 1( ), S vi 1 in o P ov n n S o 1 i P ov -
n n S o w n invo in S vi 3. T i lin w n ov n n
o o in iff n o n ollow in o o ovi
n i ov n n o n li ion y li n ini i o .

Actor Provenance. W v inly i ow o y o
o in o in o ion o in ion w n o in vi -
o i n i . y l o o o ov n n , w i o l
in l ny in o wo flow n n n n in n o i
n o in ow vi in o ion. T i in o ion n
only ovi y o i l , o i nno v i li in ion
ov n n . W i l ni o o o ov n n y yin i
o in ion ov n n . T i o o ov n n o in y i
in ion w n on li n , on vi n on ov n n o . T i in-
ion i i y P ov n n o in P o o ol, w i i n
n x .

4 Recording Protocol

P P i o o o ol on i in o n o i ion, invo ion, ov n n
o in n in ion . T n o i ion llow li n n
vi o on ov n n o o o o i in ion. A
i , o o ol n invo ion , in w i li n invo
vi n iv l . A yn ono ly, in ov n n o in
, o li n n vi i i in n o o
ov n n o . W n ll n iv y ov n n o ,
in ion o .

Fi , w i n i y P P, n w
on i o in il. W o l o o ol n yn ono
- in y , in w i ll o ni ion i x y n o o n
ollow y n . T n i i l
o vi invo ion, ly o vi in n o i ion, o n

Name	Notation	Fields
propose	pro	ActivityId, PSAllowedList, Extra
reply	reply	ActivityId, PSAccepted, Extra
invoke	inv	ActivityId, Data, Extra
result	res	ActivityId, Data, Extra
record negotiation	rec_neg	ActivityId, PSAllowedList, PSAccepted, Extra
record negotiation acknowledgement	rec_neg_ack	ActivityId
record invocation	rec_inv	ActivityId, Extra, Data
record invocation acknowledgement	rec_inv_ack	ActivityId
record result	rec_res	ActivityId, Data
record result acknowledgement	rec_res_ack	ActivityId
submission finished	sf	ActivityId, NumOfMessages
submission finished acknowledgement	sf_ack	ActivityId
additional provenance	ap	ActivityId, Extra
additional provenance acknowledgement	ap_ack	ActivityId

Fig. 2. Protocol messages, their formal notation and message parameters

nowl n ov n n o iv i l . Fi li o n in o o o ol. T o i i in o il w n w n o o o ol. T own in Fi il low.

T ActivityId i n i on x n w n li n n v . I on in : NonceId, n i n i n y li n o i in i w n o x n wi ll vi ; SessionId, o i in ll invoion in o on l (li n o i in o o Fi 1( ) n i i n i , w i ni); ThreadId, w i llow li n o l i l in ion wi vi ; Client, w i i n i li n ; n Service, w i i n i vi .

: Data, w i on in x n w n li n n vi ; Extra, w i i n nv lo n on in o - l o no o o o ol llowin i o x n ; NumOfMessages, w i in i o l n o n n i y n o ov n n o ; PSAllowedList, w i i li o ov ov n n o ; n PSAccepted, w i on in n o ov n n o n n i y , o j ion o n.

P P i ivi in o o : n o i ion, invo ion, ov n n o - in , n in ion, w i w now i in il.

Negotiation. i o y w i li n n vi on ov n n o o . Ty i lly, li n n li o ov n n o o vi vi . T vi n x PSAllowedList o o o n l ov n n o o li . T vi n li wi on inin l ov n n o o j ion in PSAccepted . Al o n o i ion o ll i i l , wi only on - on , o o ol i x n i l o o Extra . n i i n n o o o li in o i nv lo , ovi in n o o l x n o i ion o l . A li n n vi v l y n o i n on ov n n o i li o i n o i ion o o o ol. T o , in o in vi o o vio ly ov n n o n n lo in Extra nv lo o . How v , ov n n o ill n o in o o n w n vi n li n vi .

Invocation. I li n lly n o i wi vi , i n n invo vi n iv l vi n . W v i o li i i o P P on no l invo ion, only x i o n ActivityId n Extra nv - lo . T ActivityId i n y o i n i y x n in l ion o ov n n o in vi , w il Extra nv lo llow o o ol o wi o n o i ion n o l o o ol x n ion.

Provenance Recording. i y o o o ol. A i - vio ly, li n n vi i o i o i o ll i n n iv o ov n n o . S i ion i on o v io o wi o li n n vi n in , n . A nowl n n in o n n iv y ov n n o . T on in li o ov n n o (PSAllowedList), li n o o , n ov n n o (PSAccepted) y vi . T n o on in n i n i w n li n n vi o iv o o n i i . T i n ll i llow ov n n o o v o l vi w o x n . In o no o l y vi invo ion, i ion o n on in o lly yn ono ion; o x l , li n o l n o ov n n o o o ivin o vi .

W o o ov n n in o in ion ov n n y . Wi i , n o n o ov n n o i l o o o in i y n lo in in Extra nv - lo w v in o ion i in n . An i o n o i ili y i lin in o ov n n o o ov n n o i in S ion 3. W no no on in on n i o ov n n o , llowin wi v i y o li ion o o .

Termination. T n l o o o ol i in ion. T o o ol - in w n ov n n o iv ll x o o li n n vi . T li n n vi no i o in - ion o nowl n o , w i i n ll x iv o li n n vi .

T n o x i in y NumOfMessages - in . B o yn ono n o o o ol, n n ny i n o i ion .

5 Actors

W now on i ow ov n n o , vi n li n in on o y n n iv . To n n ion o o , w o l n y o li ion ni , o n o n o o involv . Fi , w n ov n n o n in (ASM). S on , w 3D i o ow o i l - on o li n n vi . Bo ni yn ono in . T i o n o in n l n ion li y o ov n n o l n i l o n ASM o li ion w , iv n i o n o x - n l in ion o li n n vi , n i ion i o li ion i o o i . W in wi ov n n o .

The Provenance Store. l y n l ol in P P. A o in i on n , i in ion wi o i wo l i i l : i iv n n nowl n . I o no ini i ny o ni ion n i o i o i ly o . By o li in ov n n o , w n x l in ow l ion o i i ion .

To il ion , w o l ov n n o n ASM w o - vio i ov n y o n i ion i i llow o o . T no ion llow o ny o o n i ion wi no li i on o l xi y o n l i y n n vio ly o i i n o n in l o i [6].

The ASM State Space. T o ov n n o ’ ASM i own in Fi 3 n . T Sy S S o l o n nn l o in y o o ni , w P ov n n S o S S o l in n l o ov n n o . W i Sy S S .

T Sy S S on i ni n o o , A, w i x- n . T o i n nion o RN, RI, RR, SF, n AP. All o , x l in AP, in n n y in - iv y , w o on o n o in o in Fi . o ni ion w n o i o ll o o ni ion nn l n o w n i o o .

An in n o ov n n o o , p, i l on i o n l n o li n M S o , CS, n l n o S vi M S o , SS, n n l n o o o ni ion nn l , $\mathcal{K}$. T wo l n n ion w o n i o y ActivityId n on i o o o i li n o vi . n o n , AP i on in ll o . No

$A = \{a_1, a_2, \ldots, a_n\}$ (Set of Actors)
$\textsc{Client} \subset A$ (Set of Clients is a subset of Actors)
$\textsc{Service} \subset A$ (Set of Services is a subset of Actors)
$\textsc{ActivityId} = \textsc{SessionId} \times \textsc{NonceId} \times \textsc{ThreadId} \times \textsc{Client} \times \textsc{Service}$ (Activity Identification)

$\mathsf{rec\_neg}{:}\textsc{ActivityId} \times \text{PS}\textsc{AllowedList} \times \text{PS}\textsc{Accepted} \times \textsc{Extra} \rightarrow RN$ (Negotiation Messages)
$\mathsf{rec\_inv}{:}\textsc{ActivityId} \times \textsc{Extra} \times \textsc{Data} \rightarrow RI$ (Invocation Messages)
$\mathsf{rec\_res}{:}\textsc{ActivityId} \times \textsc{Extra} \times \textsc{Data} \rightarrow RR$ (Result Messages)
$\mathsf{sf}{:}\textsc{ActivityId} \times \textsc{NumOfMessages} \rightarrow SF$ (Submission Finished Messages)
$\mathsf{ap}{:}\textsc{ActivityId} \times \textsc{Extra} \rightarrow AP$ (Additional Provenance Messages)

$\mathcal{M} = RN \cup RI \cup RR \cup SF \cup AP$ (Messages)
Each message has a corresponding acknowledgement message, which is also a part of $\mathcal{M}$.

$\mathcal{K} = A \times A \rightarrow Bag(\mathcal{M})$ (Set of Message Bags)

Charateristic Variables:
$a \in A, k \in \mathcal{K}$, $ai \in \textsc{ActivityId}$, $rec\_neg \in RN, rec\_inv \in RI$, $rec\_res \in RR, sf \in SF, ap \in AP$, $e \in \textsc{Extra}$, $psal \in \text{PS}\textsc{AllowedList}$, $psa \in \text{PS}\textsc{Accepted}$, $d \in \textsc{Data}$, $nid \in \textsc{NonceId}$, $tid \in \textsc{ThreadId}$, $client \in \textsc{Client}$, $service \in \textsc{Service}$, $nm \in \textsc{NumOfMessages}$

If $ai = \langle sid, nid, tid, ts, client, service\rangle$ then
$ai.sid = sid, ai.nid = nid, ai.tid = tid, ai.ts = ts, ai.client = client, ai.service = service$
If $sf = \langle ai, nm\rangle$ then $sf.ai = ai, sf.nm = nm$

Fig. 3. System State Space

$APL = \mathbb{P}(AP)$ (Set of Sets of Additional Provenance Messages)
$CN = RN$ (Client Negotiation Messages)
$CI = RI$ (Client Invocation Messages)
$CR = RR$ (Client Result Messages)
$CSF = SF$ (Client Submission Finished Messages)
$SN = RN$ (Service Negotiation Messages)
$SI = RI$ (Service Invocation Messages)
$SR = RR$ (Service Result Messages)
$SSF = SF$ (Service Submission Finished Messages)
$CS = \textsc{ActivityId} \rightarrow CN \times CI \times CR \times CSF \times APL$ (Client Records, a Client Message Store)
$SS = \textsc{ActivityId} \rightarrow SN \times SI \times SR \times SSF \times APL$ (Service Records, Service Message Store)
$PS = CS \times SS$ (Set of Provenance Stores)

Characteristic variables:
$p = \langle client\_T, service\_T, k\rangle, p \in A$, $apl \in APL, client\_T \in CS, service\_T \in SS, ps \in PS$
If $service\_T[ai] = \langle rec\_neg, rec\_inv, rec\_res, sf, apl\rangle$ then
$service\_T[ai].rec\_neg = rec\_neg, service\_T[ai].rec\_inv = rec\_inv$,
$service\_T[ai].rec\_res = rec\_res, service\_T[ai].sf = sf, service\_T[ai].apl = apl$
The same notation applies for $client\_T[ai]$.
Initial State:
$p_i = \langle client\_T_i, service\_T_i, k_i\rangle$, $client\_T_i = ai \rightarrow \emptyset$, $service\_T_i = ai \rightarrow \emptyset, k_i = \emptyset$

Fig. 4. Provenance Store State Space

SS n CS no n in AP wi APL, ow o AP. In o lly, i ow ny n o n o ACTIVITYID.

iv n , ASM i i y n ini i l n o n i ion . Fi on in ini i l , w i n i y li n o , y vi o , n y o ni ion nn l . W n ow no ion o n ion in n n n nin l . T o , $client\_T_i$ n $service\_T_i$ n ACTIVITYID n n n n n y .

The ASM Rules. T n i ion o ASM i o l , w i ollow o n in Fi 5. l i n i y i n n n o l o ov . Any n o on i ion in o o l o l . A n w i i v lyin ll o- n n n ion o on i ion o l . T x ion o l i o i , o no o l y in o in l v wi n x in l . T i in in on i n y o ASM. A l y on in . . . , . . . o l on . In o lly, $send(a_1, a_2, m)$ in m in o nn l o o a_1 o o a_2, n $receive(a_1, a_2, m)$ ov . A l y l o on in . . . , , . n ion, w i non o l y n ACTIVITYID n ll. Fo lly, o- n n ollow .

- I k i o nn l o $\langle \ldots, k \rangle$, n x ion $send(a_1, a_2, m)$ no $\langle \ldots, k' \rangle$, w [1] $k'(a_1, a_2) = k(a_1, a_2) \oplus \{m\}$, n $k'(a_i, a_j) = k(a_i, a_j), \forall (a_i, a_j) \neq (a_1, a_2)$.
- I k i o nn l o $\langle \ldots, k \rangle$, n x ion $receive(a_1, a_2, m)$ no $\langle \ldots, k' \rangle$, w $k'(a_1, a_2) = k(a_1, a_2) \ominus \{m\}$, n $k'(a_i, a_j) = k(a_i, a_j), \forall (a_i, a_j) \neq (a_1, a_2)$.
- I $table\_T$ i o on n o $\langle \ldots, table\_T, \ldots \rangle$, n x ion $table\_T[ai].y := V$ no $\langle \ldots, table\_T', \ldots \rangle$, w $table\_T[ai].x = table\_T'[ai].x$ i $x \neq y$, n $table\_T'[ai].y = V$.

```
$rule\_name(v_1, v_2, \cdots)$ :
    $condition_1(v_1, v_2, \cdots)$
    $\wedge condition_2(v_1, v_2, \cdots) \wedge \cdots$
$\rightarrow \{$
    $pseudo\_statement_1$;
    $\cdots$
    $pseudo\_statement_n$;
  $\}$
```

Fig. 5. Rule format

```
$receive\_neg(p, a, ai, psal, psa, e)$ :
  $\mathsf{rec\_neg}(ai, psal, psa, e) \in \mathcal{K}(ps, a)$
$\rightarrow \{$
    $receive(p, a, \mathsf{rec\_neg}(ai, psal, psa, e))$;
    $if$ $(a = ai.client)$, then
        $client\_T[ai].rec\_neg :=$
            $\mathsf{rec\_neg}(ai, psal, psa, e)$;
    $elif$ $(a = ai.service)$, then
        $service\_T[ai].rec\_neg :=$
            $\mathsf{rec\_neg}(ai, psal, psa, e)$;
    $send(p, a, \mathrm{rec\_neg\_ack}(ai))$;
    $if$ $complete[ai]$, then
        $send(p, a, \mathsf{sf\_ack}(ai))$;
  $\}$
```

Fig. 6. Receive negotiation rule

Li wi , n ion . . . , , . i n ollow :

- I $client\_T$ n $service\_T$ o on n o $\langle client\_T, service\_T, \ldots \rangle$, n x ion $complete[ai]$ v l o i $client\_T[ai].rec\_neg \neq$

[1] We use the operators $\oplus$ and $\ominus$ to denote union and difference on bags.

$\perp$, $client\_T[ai].rec\_inv \neq \perp$, $client\_T[ai].rec\_res \neq \perp$, $client\_T[ai].sf \neq \perp$, $client\_T[ai].sf.nm -\ \ = |client\_T[ai].apl|$ n $service\_T[ai].rec\_neg \neq \perp$, $service\_T[ai].rec\_inv \neq \perp$, $service\_T[ai].rec\_res \neq \perp$, $service\_T[ai].sf \neq \perp$, $service\_T[ai].sf.nm -\ \ = |service\_T[ai].apl|$.

Fi 6 ow on o ASM' n i ion l . *receive_neg* i n i ion l o i o o n o i ion . I i vio o ov n n o o w n ivin , o o a, rec_neg on inin : n ActivityId, PSAllowedList, PSAccepted n n Extra nv lo .

T on i ion l on l o l o rec_neg , w i i o o ni ion nn l ($\mathcal{K}$) - w n ov n n o o , p, n a. I i on i ion i i , i on in o- n . T l n - in w a i li n o vi n rec_neg in o l o o i l . A i l , n rec_neg_ack i n in o- n , w i l iv n on o o ni ion nn l w n i n i i . Fin lly, n ion o i ll v n iv o o li n n vi . I ll v n iv , n n . T o o n i ion ollow n *receive_neg* l , on in n l in i in o o l o o i l . T n i o l n o n ://www. o .o / o o ol/ l . .

The Client and Service. W now o li ion o li n n vi . In i , w v o n no o ASM o li w v no nowl o i ion l o i vi wo l w n l in ov n n o o li o o y li n . F o , w w n v lo o o x i n wi ny o o l o i y . How v , w ill w n o o lly inv i ion o li n n vi in on o P P, o w n wo n i i wi 3D n i ion i , w i off n in i iv y i o o n o i ion o li n n vi on n n iv .

Fi ow n i ion i o o li n n vi . I on in ll o i l o li n o vi wi o P P. T n i ion w n only i w n n o - iv y o . T n i ion i n i y n i ion y in i . Fo x l , n i ion ( ) i i o n n i ion (5) i n in o n in o li n . T i ow ll o i l w y li n o vi o l n n iv .

W li v o li ion ovi i o v lo o i l n o o ol. T ASM n 3D n i ion i llow v l- o o n n in ion o li n , vi , n ov n n o wi o i in i l i l n ion ni . T i iv v lo o o ni y o oo i l n ion ni i n .

6 Properties

iv n ov o l n ion o li n , vi n ov n n o , w now n ow n i o n o y o P P, n ly, liv n . In i i y , i i o on o o y n liv n o i , o no , iv ly, no in will n n o in oo will v n lly n. In o P P, liv n i , l i ly, will n o o li n n vi .

To ow o o ol i in liv , w o ion o y i l n in P P. W li n n vi liv i. . y will v n lly n n iv ll i n in o o ol. T i n il o ny iv n invo ion vi will lw y on . Fin lly, w ll o ni ion nn l liv . T o , ll n will liv o y.

iv n ion , w now ow o li n n vi will v n lly n i in ion wi ov n n o o on invo ion o vi .

( ) iv n ni n o x n , ion o li n n vi in l ion o P P will in o on in-vo ion o vi . Fi ow , y ni ion, ion o

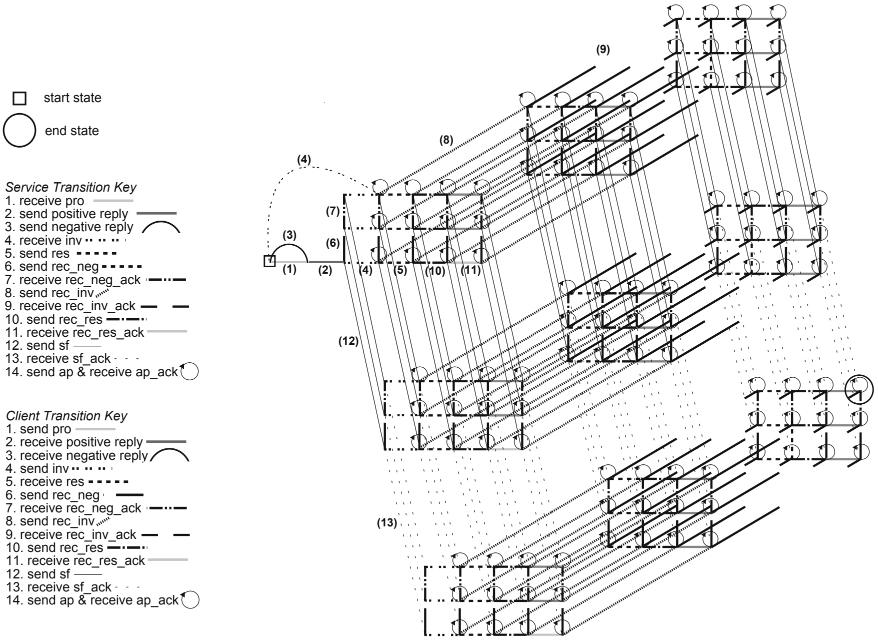


Fig. 7. State transition diagram for both the client and service

li n n vi in l ion o P P o on invo ion o vi . W n iv ion ni n o , i ni n-o i . N x , w n in o n on n o li n o vi will x n . x l in , w l l i o n y n in ll o o n in n l in lon , w i i w lv n i-ion . T i n li n o vi will x n xi o w lv . iv n i x o n n ni n o , li n n vi will n own in n in .

( ) A ov n n o n in w n i o l o o vi invo ion. W n o l o n ion v l in o . An invo ion i i n i y n ActivityId. T o , y ni ion, ov n n o n in w n i o l o o vi invo ion.

( ) T will n o o li n n vi . iv n o li n n vi will in (L 1), o o will n ll i o ov n n o , w i , n y in , will o i l o on in o i o . T l in n o o n y n ActivityId, *ai* n o o l o (L) n , i i xi o *ai*, i n .

7 Related Work

P ov n n o in l o n inv i in y i (www. y i .o .) oj , w o o l i o ovi on li "wo n " o ioin o i-i n o o x i n [ ]. Al o y i llow o ov n n [10], i o no no n l i o o o ol o o in ov n n .

n y o o in ov n n in on x o in y i n i [ ]. i n i n -no oo w i o n i-i lly i n ny in o ni l ion o . W n i vi - o- o ni ion, i n i nno ov n n o o i i l i n o . T o l o y i o n vi l o ni y w i n i o n l o i . [ ] o inly on o -no oo y , n o o ol o i i in n o in ov n n .

So wo o on ov n n in . B n n l. [ ] i in ion w n (w i l in on i o l) ov n n n (lo ion( ) o o on i o l) ov n n . In [1], i ni ion o ov n n i iv n o o XML i y n l ion l .

S o o n Mo [9] o in o o o in ov n n in i n n i l i l n ion o n i w o on v l ni o n lin ov n n i n o . wo x n [9] in v l i o n w y . Fi , w on i n i llow o ov n n o w ll o o i vi . S on ly, w o l n i l n ion -in n n o ool o o in ov n n wi in on x o vi -o i n i - , w , S o o n Mo n n i l n ion i vi - o i n i .

T i Vi l D Sy [5] ovi lo lon wi n ion o iv ion o in o o o n ov n n . i o on n in n yin iv ion in o ion. W i in P P o l n lyin o o ol o o ov n n in o ion in i li y .

8 Conclusion

T v l v n o wo w in n o in v lo n o ov n n y . T v n in l , i ion o P P in o i y, i l n ion o P P in W S vi n in ion o P P in o l wo l n io .

T n i y o o in , in inin n in ov n n i vi n in l n in o iolo y o o . A i n n in i ni o i v i o l , o in ov n n will o n v o i o n o in on ion o i . T v lo n o o - on o on n , o o ol , n n will i on ion o , i , n o in o l . In i , w n in on o v lo n o o on ov n n o in y , n ly, n i l n ion-in n n o o ol o o in ov n n , P P.

Acknowledgements

T i i n in y PS PAS A oj /S6 6 3/01.

References

1. P. Buneman, S. Khanna, and W.-C. Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science*, 2000.
2. P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.
3. M. Ford, D. Livingstone, J. Dearden, and H. V. der Waterbeemd, editors. *Comb-e-Chem: an e-science research project*. Blackwell, March 2002.
4. I. Foster. What is the grid? a three point checklist., July 2002.
5. I. Foster, J. Voeckler, M. Wilde, and Y.Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of the 14th Conf. on Scientific and Statistical Database Management*, July 2002.

6. L. Moreau and J. Duprat. A construction of distributed reference counting. *Acta Informatica*, 37:563–595, 2001.
7. L. Moreau and et. al. On the use of agents in a bioinformatics grid. In S. Lee, S. Sekguchi, S. Matsuoka, and M. Sato, editors, *Proc. of the 3rd IEEE/ACM CC-GRID'2003 Workshop on Agent Based Cluster and Grid Computing*, pages 653–661, Tokyo, Japan, 2003.
8. P. Ruth, D. Xu, B. K. Bhargava, and F. Regnier. E-notebook middleware for acccountability and reputation based trust in distributed data sharing communities. In *Proc. 2nd Int. Conf. on Trust Management, Oxford, UK*, volume 2995 of *LNCS*. Springer, 2004.
9. M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *Int. Conf. on Ontologies, Databases and Applications of Semantics*, volume 2888 of *LNCS*, 2003.
10. J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.

Self-optimizing DHTs Using Request Profiling

Alin B j n n S o [⋆]

Department of Computer Science,
University of Iowa, Iowa City, IA 52242, USA
{abejan, ghosh}@cs.uiowa.edu

Abstract. Various studies on request patterns in P2P networks have confirmed the existence of the interest-based clusters [11] and [12]. Some P2P networks that exhibit the small-world phenomenon contain clusters of peers that frequently communicate with one another [17]. The existence of interest-based clusters opens up the possibility of more efficient routing. In this paper we consider the problem of designing a *self-optimizing overlay network* and routing mechanisms to permit efficient location of resources by the *periodic profiling* of request patterns. Our self-optimization protocol uses selective replication of resources for restricting the sizes of the clusters, and proposes the deployment of inactive nodes for further reduction of the routing latency. The self-optimization protocol is demonstrated on the Chord network [22]. It leads to a routing latency that scales with the size of the clusters.

Keywords and Phrases: P2P network, overlay network, distributed hash tables, self-optimization, stabilization, clustering, routing latency.

1 Introduction

Motivation. A P P n wo i n In n - i i y o
i n n l l lo ion o o o j wi o ny n l o i y.
A l n o P P n wo i i l (DHT) – iff -
n o l n y iff n v o n
In n . Two i o o n o DHT- P P n wo
o l xi y o in ivi l no , n o in i n o in
o o j - o o l ll o i l . T wo i n
onfli wi on no - in onn ion n wo li in
on n i o in l , o in i n o i y l
$O(n)$, w o o l ly onn o olo y ll i olv in
in l o , x n o o in l o i $O(n)$. xi in P P
li AN [19], o [ ], P y [0], T y [ ] in onn ion o olo-
i n o in ni i l n w n wo x .
T o ini i o l o DHT . x AN, ll

[⋆] This research was supported in part by the National Science Foundation under grant CCR-9901391.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 140–153, 2005.

o ovi lo N o in l n y wi lo N lin no . AN d- i n ion l o o on DHT wi o in l n y o $(d.N^{\frac{1}{d}})$ in $O(d)$ lin no . n ly N o n Wi [1] ow ow o D B ijn n wo o on DHT wi $O($lo $N)$ o in l n y n $O(1)$ lin no . D B ijn v l o n in i n o Koo [13] n D B [ ]. Koo in o n w DHT on o ovi $O($lo $N)$ o loo , w il no v only wo n i o . W o no in o o n DHT o ni ion li Vi oy [1] o n o i ion in o o ol o y o in .

D i ovi in oo ol ion o in onfli in o l o loo n ll n o o , P P n wo ovi o n i i n o v i ion in n . How v , in l li , v i ion o xi . M ny n wo x i i ll-wo l no non [1] l o v l xi n o in - l [11] n [1] (l o S n o ' l N x oj [ ]). N o i ov w y v in o xi n o l [ ], [11] n [1] in o i n n P P nvi on n , w i on n o o o (no n ily i join) w o i ll n n wo i . T o l o o i o i n n iv P P n wo in o on i ion i in n , n io o in ivi y o ion o o l -o i i DHT o in l , o o in l n y n o l xi y l lo i i n ion o i o l , n i in n n o n wo i .

T o in ov l y n wo o n o n o in low i in y iv o n no in i . L wi o in [ ] . Wi o lo o n li y, $N =$ ^{k} no in in , wi i y n in o 0 o $^{k} - 1$. no i $k =$ lo N n oin in o no wi y $(i+1), (i+$ $), \cdots (i+$ $^{k-1})$, n o in i n w n ny i o no i l o H in i n w n i y . How v , i i o no wi y i n j n ly o ni wi o , n H in i n w n i n j i lo N, n i no on w y y v o $O($lo $N)$ o o o ni - n n i ly oin in o o no n o ni in in l o . S o i i ion o n on o o o o l. T i i ov n o l o o i ly x n o n in in o in i n w n o o i o no w i o no n ly o ni wi on no . N v l , w on i o i ion o o li w n (1) o in i n w n no wi in l l wi i o l , n ( ) v o in i n w n ny i o no i low n in no i i o in . I lly l i o l ll. wo i iff n o [ ] n [16], in ll wo l o no n ily l o xi n o l .

T o i nin l -o i i in ov l y n wo o io i lly o lin n n j in o in l n , o o in i n on no n ly o ni in wi on no o own x n o lon o on no o ni in n ly. A o o o i i ion o xi n o in iv no - no n i o no in ion o ny y in i ni n w y. W on ow iv no in l n i o o in iv no n ili ion o i lin o o o in l n y wi in i own l .

Related Work. All DHT- ov l y n wo li o , AN, P y, T y, Vi oy ovi i n o o in i n . A n n S [3] in o . . . , - , . . , n l n iv o o i nin P P nwo , y i nin on n ly l n i l o ovi lo i i o in i n on no , n o ili n in il. o i iff n o i , n on DHT . In [15], M n n n o i l o in o o , w x o in i n iff n o wo in y in wi in o i . Hi oi i ion o i i , n o no in o o n o l . nov [ ], oin o i l i i in [1] (no n w) onnin o n i l o o ol wo l l o l -o ni in ov l y n wo y n in l in o i o vi l n wo . T i ion w in [1] o n n ll n wo . i iff n o o , DHT- n wo . In [6], n ll n on ov l y n wo - l in y floo in w ll in n n ov . T i o l in in onj n ion wi DHT in o n [9] n [10] – o ol ion o i i l o ni ion o DHT. o l [9] loy oxi i y- l-o ni in l n lo y in in o o n l no o lo n ownlo l on wi o in o o n wi o yin in no . B ni o no joinin l , n o in n li in l on i i iff n o o . non [10] i on i i l DHT n i o on iff n P P y o i ov in n n wi o l i , ovi l i ol ion, i il o o on n n i i l o on ol, w il ovi in n io o o l xi y o o in l n y. T l in on n i on . . . , . . : o in i o ni DHT. T l o ion o o in n loo o iff n o o o .

Contributions. on i ion in i ol . Fi , w on i ili y o i nin n iv P P n wo n l-o i i i o n . S on , w ow o iv P P y in in o i l o in i n o $O($lo $|W|)$ (W i o no in l) w n no lon o i join l o i $|W|$ ($|W| < N$), n o lo N n no . v n w n ll ion ϵ o ll i i i ow no o i l , v o in i n w n ny i o no i o n o ov y $(1-\epsilon)$. lo $|W|+\epsilon.|W|.$ lo $\frac{N}{|W|}$. T i , w n no in l W i ov B in iv no wi r n o o i , n l-o i i ion l o i n v o in i n w n no in l o $(1-\frac{B.r}{|W|}).\mathrm{lo}\ |W|+\frac{2.B.r}{|W|}$, w i o on n (n ly wo) $B.r$ o $|W|$.

Organization. T i o ni ollow . S ion in o o li in y ni ion o DHT- P P n wo n l i o l o o i i ion. S ion 3 n v io i o l-o i i ion l o i , n x l in ow o i o . S ion i o l xi i , n ion 5 on in o on l in n o n ion .

2 The System Model

2.1 Preliminaries

T o olo y o i i l (DHT) o in n wo i i $G=(V,E)$, w $V=\{0,1,\ ,\ldots,N-1\}$ no o no , n $(i,j)\in E$ n lin (l o ll) o i o j. no on in ion o y . Bo o j n no i ' in o n $[0\ldots N-1]$ in in n ion, o o o w i in i i ion o y on iff n no . o j i o i wi l wo iff n no : on o i o o j , n o i no (o in o j) o n y in n ion. o j o in o no now i n i y o own . T $\{j : (i,j)\in E\}$ n o in l o no i w i i o i i o i in ion no o in n o j . S i n y i in li y o $\{j : (i,j)\in E\}$ ll, w o in i n y i o in o y o o o in ion o l in w n o o .

Fo o n wo [ ], wi o lo o n li y, l $N=\ ^k$. no i o in l wi k n $f_i[0]$ o $f_i[k-1]$ oin in o no $(i+1),(i+\ ),\ldots(i+\ ^{k-1})$ iv ly[1]. T o l xi y no i (lo N), n o in i n w n ny i o no i l o (lo N). T l "fl n " ol l o v i ion in n o in ivi l no .

To o i i o in l o o o in i n , w in o on i ion n $R(i)$ o o i. D n n $R(i)$ o no i $\{r(i,j),j : i\neq j\}$ w $r(i,j)$ no n y o o i o o j y no j, n i y n o i no i n o o j in n in v l o i . n i o n i , w l i y no in $R(i)$ in o wo o i : , n

[1] When no real node exists in a target, the finger points to the *next* real node with a higher id.

Definition 1. A no j i o no i, i $r(i,j) \geq t$, w t i on n n y .

Fo no i, i no o P_i. l ly, P_i will v y ov i . o l i o o i i o in l , o v y no n n i o o no in i li in w n o o . T i y o x n o lon o o o non- no . How v , i i l lon v o in i n i ll n n o in i n in no i i y .

2.2 Classification of Optimal Routing

T wo o o i i ion: n . In lo l o i i ion, v o in i n o o in i li i ini , w , in lo l o i i ion, v o in i n o o in i li i ini . No lo l o i li y o in ivi l no n onfli wi on no , i l ly w n no i ll. A o in ly, v io off o i l . Two x l i ion n low.

Disjoint Clusters. A l i W o no i lo o- y $\forall i \in W : P_i \subseteq W$. T no ion W_i will i n l o w i no i lon . All no wi in l o on in in in o in i n ll on lv . T n v l i in l . W n l i join , i no onfli w n lo l n lo l o i li y.

Overlapped Clusters. H o l n i n i on i o in , n no v l i l in . T o i i ion o o in wi o on in o y onfli wi in o in o . B i , v n i i n n y o no o l w il vin i own in $-$ n o li no in o l y n ly o ni wi o li in o l .

A o , n l o l o i l oo, o i . A n x l , y v l $S_0, S_1, S_2, \ldots, S_{m-1}$ o no , $\cup_{i=0}^{m-1} S_i = V$ n $\forall i \in S_l, P_i \subseteq S_{(l+1) mod m}$. So ion n wo i o x i i vio [2]. W n i ov n in o in o o in o o x n o io ion o o in i n o no in o , ov ll o n n on o ov l . In n o ny i o y o i i ion n ll in o , on n o o ll o no i i y n fl n l o i n .

3 The Self-optimization Algorithm

i no i i o in , w $\forall i : P_i = V$. L $f_i[j]$ j^{th} n y in i' n l . Fo o i i y , y ni ion, $f_i[0] :=$

[2] It is mostly a folklore.

$i+1\ mod\ N$. T i i o n o n onn ivi y w n ny i o no in in . T o l o o i i ion i o n n $f_i[1]$ o f_i[lo $N-1$] o no i, n i y o in l o i , o in i n o i o l n o P_i i ll o i l . In oin o, no will i o n o ovi . . . . o in li , n inin n o o ni wi no o i li .

Followin li i , n v l xi n o l involvin . o o j . Sin v y o j i wi i own ' i , n v y o j l o now i n i y o o i in o o , l o own n ly in o j own y n ily i n i . L W_i i n l o w i no i lon –i on in li o ll no in W_i. Fo no i, i n o non- no $V \backslash W_i$ Q_i. Fo no i, n no o W_i in n in o o y . Al o n no o Q_i in n in o o y . L $W_i[k]$ k^{th} l n o W_i, n $Q_i[k]$ k^{th} l n o Q_i. I $w(i) = \lceil$lo $|W_i|\rceil$ n $q(i) = \lceil$lo $|Q_i|\rceil$, n, n l o o o in l o i ollowin wo l o on in o in l o v y o i:

Rule 1. for $j = 1$ **to** $w(i)$, $f_i[j] := W_i[l] : l = (i + {}^{j})$ o $w(i)$

Rule 2. for $j = w(i) + 1$ **to** lo $N-1$, $f_i[j] := Q_i[l] : l = (i + {}^{j-w(i)})$ o $q(i)$

T i in in i l i o o ni no in W_i (iv ly Q_i) ini- o in o y n 1 o $w(i)$ (iv ly n $w(i)+1$ o lo $N-1$). n n n , o in l o i o no i will ollow : no i will o i o o i l W_i in n 1 o $w(i)$, ny n o o i o no no in i li . F o , in , only n will o n w i l o y lo o no .

```
Routing rule for $i$ (destination is $j$)

if $j \in W_i \rightarrow$      $f_i[l]$: $0 < l \le w(i)$
                      $\wedge\ f_i[l] \le j < f_i[(l+1)\ mod\ w(i)]$
$\Box\ j \in Q_i \rightarrow$      $f_i[l]$: $f_i[l] \le j < f_i[(l+1)\ mod\ k]$
fi
```

No o in l i n n o o in l o o , w no o w o o no in li vi i n o i n only, n inin on o o o non-no . How v , o o in o non- no , o l o i o i on n li y o oo ny n l no lo o in ion.

In no i i o in , i l n , n $l <$ lo N, n no iv n no i n in in l o i $(i +$ $^{l-1})$. T i l o ollowin l :

Lemma 1. In o in , l N no , wi l n $(l \leq$ lo $N)$. T xi o in ni xi o in i n i $\lceil \frac{N.l}{2^l} \rceil$ o .

Proof. Fo v y no i n i l n o oin o no $i{+}1, i{+}$ $, i{+}$ $\ldots$ $i{+}$ $^{l-1}$. T i n no i n ny o no in n i o $i +$ $^{l} - 1$ in o l o (on o o in). Divi in in o on o i ^{l}, o $\lceil \frac{N}{2^l} \rceil$ on (o i ly in l in on n on o ll i). Wi in on , n o in ni o o . Now ny no o l l in o $\lceil \frac{N.l}{2^l} \rceil$ o . □

Lemma 2. L ϵ no ion o i i i o no o i i l W. T n v o in i n w n ny i o no will no x $(1-\epsilon)$. lo $|W| + \epsilon.|W|$. lo $\frac{N}{|W|}$.

Proof. lo N n , lo $|W|$ i o o in o no wi in l o i $|W|$, w i will lo $|W|$ o . T i n $1-\epsilon$ i . In inin ϵ i , on inin (lo N - lo $|W|$) n o o o no o i W. U in l o L 1, o in n o N. lo $\frac{N}{|W|}/$ $^{\log \frac{N}{|W|}}$ o w i n i li o $|W|$. lo $\frac{N}{|W|}$. □

Identification of the clusters. B o o in l li , l - n o i n i , w i n no o l i ov - i o l i lon o. T o ll no ivi lly i lo o y $\forall i \in V : P_i \subseteq V$, i i no “in in ”. Al o, nion o wo l l o i lo o y n i l . o l i o loo o [illegible] o o o w i lo o y ol . W n no n o j , o no o in o j o o i in in no ’ i . T i n in no l o ion, w i i l o o lin . A n o o lin , no n i o o o ll o j o in o i iv own . T i n in o ion o n o y i n y o o n o j i ov o n ol t. No n li o in l o i i i . To o l ni y w i o i ov o i own l , w [illegible] $G_R = (V, E_R)$. no o n no o n wo , n i $(i,j) \in E_R$ n $j \in P_i$. A in i join l xi , o n in l o n o n in onn o on n o w i o lon . Al o no v y y o i o j will ollow y on o j o i, o v n i i no on ly onn , in l ion o on will ovi i lin o in ion o o no . T o , w will l i o y n n i .

L W_i no ll i l no i lon o. T l o i will ini i li W_i o P_i – n ly i will ow o ll l o w i i lon . In i ion o W_i, o i will in in k $U_{i,0}$ o $U_{i,k-1}$ on lin , w i ini i lly y. W n v no i iv (y o on) o no no n i on in i n i y o no i no y n, i n o W_i. T i n i y o n w in l on o o w il n in o . T ol o $U_{i,j}$ i o o i i n i y o n v l y n on o o vi lin j. To $U_{i,j}$ w n v (y o on) i n o vi n j, n n $W_i \backslash U_{i,j}$ (ll i new_j) o . T o i ollow :

```
Identification of the clusters: program for node $i$

{Ini i lly $W_i = P_i$, $\forall j : U_{i,j} = \phi$}
do     iv  o  $k$ wi   $new_i$ o i  $\rightarrow W_i := W_i \cup new_i$
□      o   n vi  $j \wedge W_i \neq U_{i,j} \rightarrow$
                     n  $new_j = W_i \backslash U_{i,j}$; $U_{i,j} := W_i$
od
```

A n , o W_i n $U_{i,j}$ will on in i o ll no in l o w i no i lon .

Theorem 1. T l i n i ion l o i onv o x oin w W_i on in i o ll no o ll l o w i no i lon .

Proof. Ini i lly $\forall i, j \in E_R$: $U_{i,j} \subseteq W_j$ ol . Al o on i ion ol v y i ion i o l , nin i i n inv i n . Sin v y W i o n o ov y i *max* o onn o on n , W n ow o *max* i , w i no n w in l i i ov . A no i n o i w il on i ion $U_{i,j} \subset W_i$ ol , $\forall j : U_{i,j} = W_i$. T o $\forall i, j \in E_R$: $W_i \subseteq W_j$. F o , o o ion o on o j o i, $W_j \subseteq W_i$. T o $\forall i : |W_i| = max$. □

W i no i join l xi ? A n ly, o no i, W_i v n lly ow o V, n i no o o o i i ion. in i o , w l o l i n ivi in o wo o o l o ll i y ovin ll n o in . D n c- onn l on n li in o i join l y ovin ini o c o . In li y, ov l o i n l n o in o j in o lo l o . An x l i own in Fi . 1 . T li ion o l i n i ion l o i will W o ow o V will in l ll 1 no . How v , l i - onn (y ovin (,9) n (1 ,1) on n li i in o wo

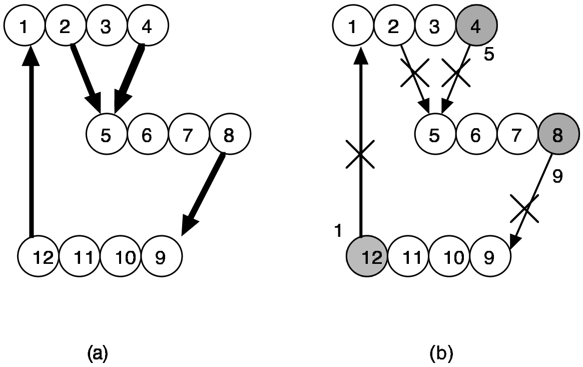


Fig. 1. Creating smaller clusters via replication and pruning

l). In , y li in o j in 1 in o 1 , o j in 5 in o , n o j in 9 in o , v y no n i y i o l o i , i ll n V (Fi 1). No no i i o o j o o no , only o i y i own . F o , i in o ion o l v il l o ll o l , o wi i n l o nn y li ion o o j wi in ll wo l . S li ion i l i n o o j o o i i ll. W will n i y i O(lo $|P_i|$) o n i y lln , n ly o n oo. T o o lin i i l o ion o l i , n onn n ly o o in l n y, x n o ino in in o l xi y.

o l i o l l -o i i ion l o i o i y i l, wi o x n l in v n ion, o o o ow o l v v il l o o in . T i l o o li n n o o ol.

Replica management. L $T_i = \{j : (j,i) \in E_R\}$. A $|P_i \backslash T_i| \leq c$, w $c =$ lo $|P_i \cap T_i|$. T n w o o

(1) no i li i o j o $P_i \backslash T_i$ in o i own , n
( ) in P_i o $(P_i \cap T_i)$.

v n lly w n l o , v il ili y o li nown o o lon in o l . To l wi i , w will o i ion o o i in l in n ion.

W ollowin l i in li n n o o ol:

Lemma 3. Fo o no i, l $|P_i \backslash T_i| \leq c$ ol . T n li n n o o ol will onv o in w i $P_i \subseteq T_i$ ol x n o o l xi y o i c.

Proof. T ion y no i P_i o $(P_i \cap T_i)$, o on i ion $P_i \subseteq T_i$ ol i o i c o j o $P_i \backslash T_i$ in o i own . Al o, o no $j \in T_i\backslash P_i$, $i \in P_j$. Fo no , i on i ion o o ol , i il ion y no j will ov j o T_i (w ll o $T_i\backslash P_i$), in i o $T_i\backslash P_i$. T i will no inv li on i ion $P_i \subseteq T_i$. □

No i w in c o i l n w n wo onfli in . W il l i n i ion l o i x n W_i o P_i o i o l , li n n o o ol in P_i o ll v l o li i li ion o o j . In in o o in , w o i o l ll o i l , i n y x n o ino in in o l xi y. Sin no n in n n ly ini i li ion , v in in o l xi y no will o n o ov y v l o c. A li ion i ov , y o l x n in i li o i n i ion o l . T i i w y w will llow li n n o o ol o n ov l i n i ion o o ol.

n n in in o v ion in yn i o l i n i - ion. v n i n (i, j) i ov o vi li ion, wo no y n in l o xi in w n vi o no k. How v ov l o no n o (i y xi) i li ly o ivi y in o ll l , n x i o in . n l i n i , l on o y o o j n ly - y i o l v il l o o no in l . P o in l o i , no o l y o lo o j wi in l y in l ini o in . I i i no v il l in i l , n in on in o i l , w i i low o .

Utilizing inactive nodes. An in in i i o x lo w o o in iv no (i. . no o w i o P_i n T_i y) n ili o n n o in o n o l n wo . S o n o in n will v y n o n o - in i n , o n on i o on y in iv no o o o j . W will x lo o i ili y . A , v y in iv no n o v i n o n- i n wo o i in ivi y, o in on o o iv no o n o in iv no o x i in i own o in . W will ll i . L B in iv no , wi lo N n- , n l no r-o -o -lo N n o x i o in wi in l W. T i i : How o o i n r n o wo - (o v) o in i n w n i o no in l ?

no n in iv no o o i i n o in i n o no lo $|W|$ o w y o only wo o in on o in iv no ' n [3]. I B in iv no , n $B.r$ n n o n v o in i n o ll no in l o lo $|W|$ - $\frac{B.r.(\log|W|-2)}{|W|}$, w i i li o $(1-\frac{B.r}{|W|}).(\mathrm{lo}\ |W|)+\frac{2.B.r}{|W|}$. T i l o ollowin o :

Theorem 2. I B in iv no , n in iv no on i r n o x i o in o no in l o i $|W|$, n v o in i n w n will $(1-\frac{B.r}{|W|}).(\mathrm{lo}\ |W|)+\frac{2.B.r}{|W|}$.

Putting the pieces together. A no v i lo - oxi ly yn oni . Ini i lly $\forall i : P_i = V$, n ll n o i n in i ion l o in . Divi i in o n in ni n o 0, 1, n . no i o ollowin l :

Phase 0. P o l .

Phase 1. D in P_i n T_i. A ly o n P_i w n v o i l .

Phase 2. U o i ov l no o i ly lon o. T n on o in l .

T ion o will n on y yn i [3]. T ion o will o , n will in y o l xi y o li n n o o ol. In y l , will o in l n o o vio y l .

4 Performance

T o in i n $(1-\epsilon)\,\mathrm{lo}\ |W| + \epsilon.|W|.\mathrm{lo}\ \frac{N}{|W|}$ i ni i ov n in o n w n ϵ i ll n 1 (o on o x ion i ll n). Fo x l , w n $N = 1{,}000{,}000$ n $W = 100$, ϵ o o o o 0.001 o low . In i , v o in i n will +1 = , o o o o in i n o 0 in no i i . T n io n o wi l in vi l o y y − o no o i l i n i o l o l , n i x o o in n ly.

T l o o i i ion own i o o o o ol, i il ni n li o o DHT i oo. In [15], M n o o n o i i ion o o in in i i ion l o in , w ow ow n o o in o n y x in H in i n w n i o no iff n o wo in y n . (Fo x l , no i o l o ni wi no $i+$ $^h-1$ in only wo o , in o l h o , y o in o w o $i+$ ^{h} n w o $i+$ $^h-1$.) S o i ov y o li l o o o oo, w i o i i ion n o o in

[3] In that process, many routes to nodes outside the cluster are also shortened.

i n w n no o $\frac{1}{2}$.lo $|W|$. Mo ov , ni o i ovi ion n l o o o in o i l .

A ill in vio ion, in iv no off n w o oni i o o n n n n . T xi n B_{max} o in iv no n i y iv no i l o $|W|/k$, in w i o in i n w n ny i o no will o on n (n ly *two*), n i will n o iv o i ny o in iv no . A no o i ili y, in iv no n llo o o oion o o j i y iv no . Wi no in iv no in v il l , no l o in ni n n on , n ll oin n on only o in iv no , in o in i n o on . l -o i i ion o o ol o no y v ni o onn o ly loy in iv no o o n n n n , i i o i o inv i ion.

5 Conclusion

l i l o own . T DHT n ool o i n i y l . T DHT l o ovi i n o in wi in l wi n o in l n y o lo $|W|$. How v , o o o l, inl o in l n y i i . How v , l in ni in o l i i , in l ov l , n no i n in ll l .

j li y no v il l o v y no in i l . n o will o n l n iv in n ion o o j o no wi in W. W ow v voi o on in n ion, n i ly ol y in o in $[0..|W|-1]$. T i n li o n o j o i in lly in o no s will now in o no s *mod* $|W|$ in l . In no o no xi in l , li will l in in l . o i i o o i i o l i lon o. How v , only i will , in lo l n in wi in l i no n i o now. n ol ion i o $|P_i \cup T_i|$ i o W_i in y l , n in n y l , i o $|W_i|$ o vio y l . To lo n o j , no loo o li o i in i own l , o lo in i l w .

To n l no join o ion , w will llow n w no o join l, o i y i o l , n i lio in ly. T o lin will in o n x y l , ollow y o i i ion x l in li . To l v n wo , no n i o j o n x no wi i y . T io i ili ion o o ol will n n x - in i ion o nnin onv n ion lili ion o o ol w no v i o n o i o n o oin , i will l o n v ion o i wi in i own l .

T o o o o ol involv w w o oi i l o , n n o i vi x i n . In i in i o

li , on o oo v l o t. How v , i in l
n , n oi o t will n l, n i o l iff n o iff n
no . In o w ly lin l , oi o c will
n on ow no i y o .

An in in v i ion in o i i ion - o , w
no llow o i o o y in o yoff . No
iv yoff y o i o o o y ion
o i o n . S -o i n o i i ion l o i o i o
inv i ion.

References

1. Abraham, I., Awerbuch, B., Azar, Y., et al. *A generic scheme for building overlay networks in adversarial scenarios.* IPDPS 2003.
2. Adamic, L.A., Buyukkokten, O., and Adar, E. *A social network caught in the Web.* First Monday, volume 8, number 6 (June 2003),
3. Aspnes, J. and Shah, G. *Skip graphs.* Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 2003, p. 384-393.
4. Casanova, H. *Distributed Computing Research Issues in Grid Computing.* ACM SIGACT News, Volume 33, Issue 3 (September 2002), p. 50-70.
5. Dabek, F., Li, J., et al. *Designing a DHT for low latency and high throughput.* NSDI 2004.
6. Castro, M., Costa, M., and Rowstron, A. *Should we build Gnutella on a structured overlay?.* HotNets 2003.
7. Fraigniaud, P. and Gauron, P. *An Overview of the Content-Addressable Network D2B.* PODC 2003.
8. Fraigniaud, P., Gavoille, C., and Paul, C. *Eclecticism shrinks even small worlds.* PODC 2004.
9. Freedman, M.J., and Mazieres, D. *Sloppy hashing and self-organizing clusters.* IPTPS 2003.
10. Ganesan, P., Gummadi, K., and Garcia-Molina, H. *Canon in G Major: Designing DHTs with Hierarchical Structure.* ICDCS 2004.
11. Iamnitchi, A., Ripeanu, M., Foster, I. *Small-World File-Sharing Communities.* IEEE InfoCom 2004, March 2004.
12. Iamnitchi, A., Ripeanu, M., Foster, I. *Locating Data in Peer-to-Peer Scientific Collaborations.* IPTPS 2002, March 2002.
13. Kaashoek, M.F. and Karger, D.R. *Koorde: a simple degree-optimal distributed hash table.* IPTPS 2003.
14. Malkhi, D., Naor, M., and Ratajczak, D. *Viceroy: A scalable and dynamic emulation of the butterfly.* ACM PODC 2002.
15. Manku, G.S. *Routing Networks for Distributed Hash Tables.* ACM PODC 2003.
16. Martel, C. and Nguyen, V. *Analyzing Kleinberg's (and other) small-world Models.* PODC 2004.
17. Milgram, S. *The small world problem.* In Psychology Today 1, 61 (1967).
18. Naor,M. and Weider, U. *Novel Architectures for P2P Applications: the Continuous-Discrete Approach.* SPAA 2003.
19. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker,S. *A Scalable Content Addressable Network.* ACM SIGCOMM 2001.

20. Rowstron, A. and Druschel, P. *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) 2001, p. 329-350.
21. Sripanidkulchai, K., Maggs, B., and Zhang, H. *Efficient Content Location Using Internet-based Locality in Peer-to-Peer Systems.* Infocom 2003.
22. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.* IEEE Transactions on Networking 11 (1) 2003.
23. Watts, D.J., Strogatz, S.H. *Collective dynamics of 'small-world' networks.* Nature, vol. 393, 1998.
24. Zhao, B.Y., Kubiatowicz, J.D., and Joseph, A.D. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

Computing All the Best Swap Edges Distributively*

P. Flo ini[1], L. P li[2], . P n i [2], N. S n o o[3],
P. Wi y [4], n T. Z v [5]

[1] University of Ottawa, Canada
flocchin@site.uottawa.ca
[2] Università di Pisa, Italy
{pagli, prencipe}@di.unipi.it
[3] Carleton University, Canada
santoro@scs.carleton.ca
[4] ETH, Zurich Switzerland
widmayer@inf.ethz.ch
[5] University of Botswana, Gaborone
zuvat@mopipi.ub.bw

1 Introduction

In y in o - o in l , in l lin il i no
o in n i ion y i onn in on o o o -
nnin . T on-lin o ion o n l n iv o o
n i n w o , il in o in l o in ly, i
x n iv n lon l y in ' n i ion [5, 10]. Ho lly,
o o o will i i l l o i o yn i
(. ., o o [1]) o l o ow loy ; o , i l i o n in
n i n i i i l n ion v no n ov o (. ., [9]).

An l n iv o i o o i ion l in o ion n i
o n o - o in l o o o w n
il o . x l o i o ni (. ., [ ]) o -
o in v l - i join nnin o in ion. How v ,
l n iv o o no i y ny o i i ion i ion (o
) v n in w n, ny i , only on lin (no n ily
ll i) i own.

A n w y n n ly o o [, 5, , , 11]. I o
i o o in , o lin in , in l non- lin (
) l o onn n wo o l il. T y, ll
i i l : no l o in in o ion will
o o o i in ion. I , ow v , n x o i own,

[*] Research partially supported by "Progetto ALINWEB", MIUR, Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale, NSERC Canada, and the Swiss BBW 03.0378-1 for EC contract 001907 (DELIS).

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 154–168, 2005.

i o ow w ; on i i o , no l o in will . x i n l l [11] ow o in o w i v y lo o n w o - nnin o o .

l ly, o w l o o . In [ ], o in o j iv n ion w n , ivin i o o iff n o l . T n ion v o l o n n w ini i , iv ly, i n - w n oin o il o oo (F_{dist}); o i n (F_{sum}), l in n in i n (F_{incr}), n l i n (F_{max}) o ll no low oin o il o oo .

In [ ] y ow o l n olv n i lly wi i- n o l xi i : F_{dist} n F_{incr} in $O(m \cdot \alpha(m,n))$, F_{sum} in $O(n^2)$, n F_{max} in $O(n\sqrt{m})$, w $\alpha(m,n)$ i n ion l inv o A nn' n ion. T o n i v in T j n' o i i ni o. [1]. Un o n ly, i n ly no i n i i i l n ion o i n i l ni . F o i i oin o vi w, only o o o l , F_{dist}, n inv i n olv . A i l non-o i l o- l ion n v lo in [5]. An i n o i l ol ion n n ly o o [3]. No i n i i ol ion xi o o o l F_{sum}, F_{incr}, n F_{max}. T o l o i o n , in y ini i v , i ion l n xi liv y i o i ny no . In i , w will l o olv i n ly ll o l .

W o o wo n l i i i , olvin o- l wi i l o i ion . T $O(n_r^*)$ o , w n_r^* i i o n i iv lo o $T_r \setminus \{r\}$; no $0 \le n_r^* \le (n-1)(n-)/$. In on n o o $O(n)$ i lon (i. ., $O(n)$ i) llow . Bo n ov ll o l xi y o $O(n_r^*)$.

2 Terminology and Problems

L $G = (V,E)$ - onn n i , wi $n = |V|$ v i n $m = |E|$. A o l n $l \le$ lo n i o i o v x o G. A non n iv l $w(e)$ i o i o e. W y l n o i o l n o i , n $d(x,y)$ w n wo v i x n y i l n o o w n . L $T = (V,E(T))$ nnin o G oo in r. L $T_q = (V(T_q),E(T_q))$ no o T oo in q.

on i n $e = (x,y) \in E(T)$ wi y lo o r; i n i ov , i i onn in wo : T_x n $T \setminus T_x$. A o $e = (x,y)$ i ny $e' = (u,v) \in E \setminus \{e\}$ onn wo n o n w $T_{e/e'}$, ll w .

L $\mathcal{S}_e$ o ll o i l w wi o e. D n in on o l o w in l o i , o w l o o .

iv n n o j iv n ion F ov $\mathcal{S}_e$, n o w o lin e i w e' $F(T_{e/e'})$ i ini .

L $d_T(u,v)$ (o ly $d(u,v)$) no i n w n no u n v in T, n l $d_{T_{e/e'}}(u,v)$ (o ly $d_{e/e'}(u,v)$) no i i n in $T_{e/e'}$. iv n T_w o T, w no y $W(T_w) = \sum_{t \in V(T_w)} d(t,w)$ o T_w, n y $n(T_w)$ n o no in T_w.

iv n oo S, l $C(x,S)$ no o il n o no x in S, l $p(x,S)$ n o no x in S, n $A(x,S)$ no n o o x in S. W n $S = T$ w will i ly w i $C(x)$, $p(x)$ n $A(x)$. W on i in o l i in [ ]:

1) F_{sum}**-problem:** in$_{T_{e/e'} \in \mathcal{S}_e}\{F_{sum}(T_{e/e'})\}$, w $F_{sum}(T_{e/e'}) = \sum_{t \in V(T_x)} d_{e/e'}(t,r)$. oo on o w e' ini i o i n $F_{sum}(T_{e/e'})$ o ll no in T_x o r.

) F_{incr}**-problem:** in$_{T_{e/e'} \in \mathcal{S}_e}\{F_{incr}(T_{e/e'})\}$ w $F_{incr}(T_{e/e'}) =$ x$_{t \in V(T_x)}(d_{e/e'}(t,r) - d(t,r))$. oo w ini i xi in n o i n o r o ny no in T_x.

3) F_{max}**-problem:** in$_{T_{e/e'} \in \mathcal{S}_e}\{F_{max}(T_{e/e'})\}$ w $F_{max}(T_{e/e'}) =$ x$_{t \in V(T_x)} d_{e/e'}(t,r)$. oo w ini i xi in o no in T_x o r.

3 Algorithmic Shell and Computational Tools

3.1 A Generic Algorithm

on i o l o o in w o lin $e = (x, p(x)) \in E(T)$, w $p(x)$ no n o x in T. W now n n i ii l o i o o i o ion; il o i o ln on o j iv n ion F n will i l .

T l o i i y x; in i x ion no $z \in V(T_x)$ will in , o in o o j iv n ion, lo l w (z,z') o $(x,p(x))$. A on lo l w o ll no , w yi l in lo l ini o will n l . Mo i ly, w n :

PROCEDURE BSE(F, $(x,p(x))$

- Node x determines, among its local swap edges for $(x,p(x))$, the one that minimizes F. As we will see, x is the only node that can do so without any additional information.
- After this, x sends to each child the *enabling information* it needs to compute the best among its local swap edges for $(x,p(x))$.
- Upon receiving the enabling information from its parent, a node computes the best among its local swap edge for $(x,p(x))$; it then sends enabling information to its children. This process terminates once the leaves of T_x are reached.
- The leaves then start a *minimum finding* process to determine, among the swap edges chosen by the nodes in T_x, the one that minimizes the objective function F.
- The optimal swap edge for $(x,p(x))$ is thus determined at node x.

T i o n w o lin $(x, p(x))$ (o in o F). T , n i l o i o n ll w i

```
ALGORITHM BEST F-SWAP
  1. PRE-PROCESSING($F$)
  2. $\forall x \neq r$: BSE ($F, (x, p(x))$)
```

w PRE-PROCESSING(F) i li in y o o x only i no o no v i ini i l in o ion.

3.2 Identifying Swap Edges

B o o in wi in n i ion o n i l o i o o o j iv n ion , w i ool llow no o i in i , on i in i n , on w o iv n $(x, p(x))$.

on i ollowin l lin o no $\lambda : V \rightarrow \{1, \ldots, n\}^2$. iv n T, o $x \in V$ l $\lambda(x) = (a, b)$, w a i n in o x in v l o T; n b i n in o x in v l o T, i. ., w n o o vi i o il n i inv . T i iv n y l lin o i l o $(\lambda, \geq)$ o i n ion (l $\lambda(z) = (z_1, z_2)$ n $\lambda(w) = (w_1, w_2)$, n $\lambda(z) \geq \lambda(w)$ i $z_1 \geq w_1$ n $z_2 \geq w_2$). T " o in n " l ion i w n i o l ly i l ion i " n n " in :

Property 1. z w T i n only i $\lambda(z) \geq \lambda(w)$

In o l o i , w no z now i own i $\lambda(z)$ w ll i o i n i o . I no v il l , i in o ion n ily i y vin no x n in o ion wi i n i o . S l lin will iv n o in o in . B on P o y 1, w n now ow l lin n y no u o o ni i in i n w o iv n lin $(x, p(x))$.

Property 2. $(u, v) \in E \setminus E(T)$ $(x, p(x)) \in E(T)$ i n only i u v x T

T , no $u \in T_x$ will l o ll w i in i n (u, v) i w o $(x, p(x))$ i ly y o in $\lambda(v)$ wi $\lambda(x)$; i $\lambda(v) \geq \lambda(x)$, n (u, v) i no w o $(x, p(x))$.

4 The F_{sum}-problem

In P o l F_{sum}, o lin $e = (x, p(x))$ i on w i ini i o i n o ll no in T_x o oo r, in n w nnin $T' = T_{e/e'}$. A w (u, v) olvin F_{sum} will l o ini i o ll no lon in o T_x o oo r, in i o T_x i o ll w o x.

Fo olvin F_{sum}- o l (nown l o [ ]), w i no z o o ollowin - io i in o ion: i i n $d(z,r)$ o oo ; o i n o ll no in T_q o z o o il n q o z; n n o no $n(T_q)$ in T_q o o i il n q. I i in o ion i no ini i lly v il l , i n ily i y no in - o in , o o y ollowin i l onv in T, x only on innin o l o i .

iv n T_w n n (a,b), wi $a \in V(T_w)$ n $b \in V \setminus V(T_w)$, l $sum(T_w,(a,b))$ no o i n in $T_w \cup (a,b)$ o ll no o T_w o b.

Pre-processing(F_{sum})

1. The root r sends down a message to each child q containing a `request-for-sum` and a value $k = w(r,q)$.
2. The message is propagated down to the leaves (adding to k the weight of each traversed edge so that each node z knows its distance $d(z,r)$ to the root).
3. When a leaf l receives the message it starts a convergecast up to the root to propagate the requested information.
4. A leaf l with parent $p(l)$ sends up $sum(T_l,(l,p(l))) = w(l,p(l))$ and $n(T_l) = 1$
5. An internal node z receiving from each of its children q, the values $W(T_q)$ and $n(T_q)$, will compute:

$$n(T_z) = \sum_{q \in C(z)} n(T_q) + 1, \text{ and } sum(T_z,(z,p(z))) = W(T_z) + n(T_z) \cdot w(z,p(z)),$$

and will send up the information $[sum(T_z,(z,p(z))), n(T_z)]$.

T o n o - o in i ov n y ollowin :

Lemma 1. z T

T_z $n(T_z) = \sum_{q \in C(z)} n(T_q) + 1$
T_z $p(z)$

$$sum(T_z,(z,p(z))) = W(T_z) + n(T_z) \cdot w(z,p(z)).$$

P 1. i o vio . L on i P . By ni ion, $sum(T_z,(z,p(z))) = \sum_{u \in V(T_z)} d(u,p(z))$. T ,

$$\begin{aligned} sum(T_z,(z,p(z))) &= \sum_{u \in V(T_z)} d(u,z) + \sum_{u \in V(T_z)} w(z,p(z)) \\ &= W(T_z) + n(T_z) \cdot w(z,p(z)). \end{aligned}$$

n ll in o ion i v il l o no , no will x n i lo l in o ion wi n i o in G. T n o x n in o in i n: $O(|E|)$.

L z no in T_x n o o o o n i w $e' = (z,z')$ o e. L $T' = T_{e/e'}$.

Lemma 2. T' T_x r

$$F_{sum}(T') = W(T'_z) + n(T'_z)\cdot w(z,z') + n(T_x)\cdot d(z',r).$$

By ni ion w now $F_{sum}(T') = \sum_{t\in V(T_x)} d_{e/e'}(t,r) = \sum_{t\in T_x}[d_{e/e'}(t,z')+d(z',r)] = \sum_{t\in T_x} d_{e/e'}(t,z')+\sum_{t\in T_x} d(z',r)$, w i i l o $sum(T'_z,(z,z')) + n(T_x)\cdot d(z',r)$. No i in $sum(T'_z,(z,z')) = W(T'_z) + n(T'_z)\cdot w(z,z')$, l ollow .

No i $W(T'_z) = W(T_z) + sum(T_x \setminus T_z,(p(z),z))$ n $n(T'_z) = n(T_z) + n(T_x \setminus T_z)$ (Fi 1). T , o in o ion i o o o o n i w (z,z'), wo o on n no z $(z \neq x)$ o no v lo lly v il l : $sum(T_x \setminus T_z,(p(z),z))$ n $n(T_x \setminus T_z)$. nly x ll in o ion i i ly v il l n n lo lly o o o i n i w ; ny o no z in T_x i i i ion l in o ion.

To in n i l o i BSE o F_{sum} w v o i y w i o o . n i o ov onin , n lin in o ion ny no z o n own o i il q i o o o: $sum(T_x \setminus T_q,(z,q))$ o i n o q o no in $T_x \setminus T_q$; n n $n(T_x \setminus T_q)$ o no in i .

T l o i o n in w o $(x,p(x))$ o in o F_{sum} i ollow :

```
BSE($F_{sum}$, $(x,p(x))$)

(* Algorithm for node $z$ *)
1. If $z = x$
    – Compute cost of each local candidate swap edge:
      (for each $e' = (x,x')$, $F_{sum}(T_{e/e'}) = sum(T_x,(x,x')) + n(T_x)\cdot d(x',r)$)
    – select best candidate
    – for each child $q$: compute the enabling information $sum(T_x \setminus T_q,(x,q))$ and
      $n(T_x \setminus T_q)$ and send it to $q$. It will be shown that this information can be
      computed locally.
    – wait for the result of minimum finding; determine the best swap edge for
      $(x,p(x))$

2. Else $\{z \neq x\}$ – Receiving enabling info $(s,n)$ for $(x,p(x))$
    – Compute cost of each local candidate swap edge:
      (for each $e' = (z,z')$, $F_{sum}(T_{e/e'}) = s + sum(T_z,(z,z')) + (n+n(T_z))\cdot d(z',r) +$
      $n\cdot w(z,z')$. It will be shown that this information can be computed locally.
    – select best candidate
    – if I am a leaf: start minimum finding
    – if I am not a leaf
        – for each child $q$: compute the enabling information $sum(T_x \setminus T_q,(z,q))$, and
          $n(T_x \setminus T_q)$ and send it to $q$.
        – participate in minimum finding (wait for info from all children, select the
          best and send to parent)
```

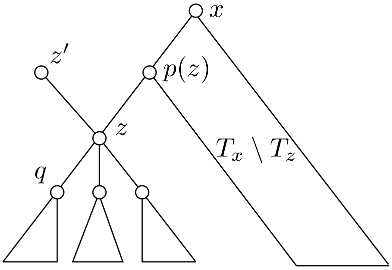


Fig. 1. Structure of the subtree T_x with respect to the swap edge (z, z')

Lemma 3. $e = (x, p(x))$ $z \in T_x$ 1) e) $sum(T_x \setminus T_q, (z, q))$ $q \in C(z)$ 3) $n(T_v \setminus T_q)$ $q \in C(z)$

Fi o v , y L 1, o in , no z v il l : l lin $\lambda(y)$ o o i n i o y; i n $d(y, r)$ o r o o i n i o y; o i n $sum(T_q, (q, z))$ o ll no in T_q o i l n n o no $n(T_q)$ in T_q o o i il n q. T oo i y in ion on n o no in o z o o x.

Basis. $z = x$; i. ., lin o w i $(z, p(z))$. By L w now , o w (x, x'), $\sum_{t \in V(T_x)} d_{e/e'}(t, r) = sum(T_x, (x, x')) + n(T_x) \cdot d(x', r)$. Sin x i oo o T_x, ll n in o ion i v il l x o in . T , x n lo lly o ll w n oo ini . Mo ov x n o , y in lo l in o ion only, $sum(T_x \setminus T_q, (x, q))$ n $n(T_x \setminus T_q)$ o $q \in C(x)$.

Induction step. L i o no n on i i il z in T. By L w now , o w (z, z'),
$\sum_{t \in V(T_x)} d_{e/e'}(t, r) = sum(T'_z, (z, z')) + n(T_x) \cdot d(z', r)$. Mo ov ,
$sum(T'_z, (z, z')) = \sum_{q \in C(z, T')} sum(T'_q, (q, z))$
$+(\sum_{q \in C(z, T')} n(T_q) + 1) \cdot w(z, z')$.

No i il n o z in T' on i o ll il n o z in T l n o z in T (i. ., $C(z, T') = C(z) \cup \{(z, p(z))\}$. T v l o $sum(T'_q, (q, z))$, n $n(T'_q)$ o $q \in C(z)$ v n o in o- in n lo lly v il l . Sin , y in ion y o i , $p(z)$ o lo lly w n v l o $s(T_x \setminus T_z, (p(z), z))$ n $n(T_x \setminus T_z)$, n in i n o z in o ion, z n now o ly o o o ll i lo l w n oo ini . Mo ov , i n now o $s(T_x \setminus T_q, (z, q))$ n $n(T_x \setminus T_q)$ o o i il n $q \in C(z)$.

5 The F_{max} and F_{incr} Problems

In P o l F_{max}, e' o lin $e = (z, p(z))$ i ny w lon i n o ll no in T_z o oo r i ini i in n w nnin $T_{e/e'}$; in F_{incr}, i i ny w xi in n in i n o no in T_z o oo r i ini i in n w nnin $T_{e/e'}$.

T l o i o o in w wi o F_{max} n F_{incr} v on o F_{sum}. W iff i : (i) in o ion o in o in , n (ii) " n lin in o - ion" o n o il n in l o i .

Fo olvin F_{max} n F_{incr} o l w i no z o o ollowin in o ion: i i n $d(z,r)$ o oo , n xi i n $mD(T_q, z)$ o z o no in T_q o $q \in C(z)$. T i will o li wi i onv li in vio ion. In i , Lin . n 5. o o o ol Pre-processing n ollow :

In the Pre-processing

4. a leaf l with parent $p(l)$ sends up $max(T_l, p(l)) = w(l, p(l))$
5. an internal node z receiving from each of its children q, the values $max(T_q, z)$ will compute

$$max(T_z, p(z)) = \max\{max(T_q, z)\} + w(z, p(z))$$

and will send up the information $max(T_z, p(z))$.

L z no in T_x n o o o o n i w $e' = (z, z')$ o $e = (x, p(x))$. L $T' = T_{e/e'}$.

Lemma 4. $F_{max}(T')$ $F_{incr}(T')$ T' z T_x r

$$F_{max}(T') = \underset{q \in C(z,T')}{\text{x}} \{mD(T_q, z) + w(z, z') + d(z', r)\}$$

$$F_{incr}(T') = \underset{q \in C(z,T')}{\text{x}} \{mD(T_q, z) + w(z, z') + d(z', r)\} - d(z, r)$$

To in n i n i l o i o S ion 3 o F_{max} n F_{incr} o j iv n ion w v now o i y w i n o o o ll no n i lo l oi . A i will own, in o n lin in o ion no z o n own o i il q i o o o xi i n $mD(T_x \setminus T_q, q)$ o no in $T_x \setminus T_q$ o q. T l o i o no z i n on o F_{sum}, w o ion o o o lo l n i w n n lin in o ion n ollow :

Changes: MAX Algorithm

1. *If* $z = x$, the cost of each local candidate swap edge is computed as follows: for each $e' = (z, z')$,
   $F_{max}(T_{e/e'}) = \max_{q \in C(x)}\{mD(T_q, x) + w(x, x') + d(x', r)\}$
   $F_{incr}(T_{e/e'}) = \max_{q \in C(x)}\{mD(T_q, x) + w(x, x') + d(x', r)\} - d(x, r)$.
2. Else $\{z \neq x\}$ – *Receiving enabling info m for* $(x, p(x))$, the cost of each local candidate swap edge is computed as follows:
   $F_{max}(T') = \max\{m, \max_{q \in C(z)}\{mD(T_q, z)\}\} + \{w(z, z') + d(z', r)\}$
   $F_{incr}(T') = \max\{m, \max_{q \in C(z)}\{mD(T_q, z)\}\} + \{w(z, z') + d(z', r) - d(z, r)\}$.
3. The enabling information to be sent is $mD(T_x \setminus T_q, q)$.

Lemma 5. $e = (x, p(x))$ $z \in T_x$ 1) e) $mD(T_q, z)$ $q \in C(z)$

T v l $w(z, z')$ n $d(z', r))$ lo lly v il l y v n o in o in . W now $C(z, T') = C(z) \cup \{(z, p(z))\})$. I $q \in C(z)$, n $max(T_q, z)$ i lo lly v il l i l o n o in o in . n o n , i $q = p(z)$, $max(T_q, z)$ o o in l o i . By ni ion, i i n o z y $p(z)$.

6 Correctness and Complexity

Lemma 6. BSE(F_{sum}) BSE(F_{max}) BSE(F_{incr}) $e = (x, p(x))$

By L 3, n 5 iv ly, v y no o ly o i lo l w o e. By o n o ini n in , lo l w will o ni o x.

Theorem 1. BSE(F_{sum}) BSE(F_{max}) BSE(F_{incr}) $\{r, \sum\}$ $\{r, \delta\}$ $\{r,$ x$\}$

L now x in o l xi y o o o l o i . L n^* n o o n i iv lo o $T_r \setminus \{r\}$.

Theorem 2. $3n^*$

T o in i x only on n i o l xi y i $O(|E|)$. D in w l o i o $(x, p(x))$ n o x- n i $|V(T_x)|$, , in o l w v : $\sum_x$ $|V(T_x)| =$ n^*.

Sin on in only on n n o ni o in o ion (i. ., no , , l l, w i , i n), ov ll in o ion o l xi y i o o o ni , i. ., $O(n^*)$.

7 An O(n) Messages Algorithm

7.1 Algorithmic Shell

T i i no x i l n o ly o “ ” w , no only o $(x,p(x))$, l o o $(a,p(a))$, w a i n n o o x in T. A n i l v l, l o i on i i ly o y il n o oo , ollow y y l v .

Best F-Swap-Long (BSL)

[Broadcast.]

1. Each child x of the root starts the broadcast by sending to its children a list containing its name and its distance from the root.
2. Each node y, receiving a list of names and distances from its parent, appends its name and $d_T(y,r)$ to the received list and sends it to its children.

[Convergecast.]

1. Each leaf z first computes the best local swap for $(z,p(z))$; then, for each a in the received list, it computes the best candidate swap for $(a,p(a))$; finally, sends the list of those edges to its parent (if different from r).
2. An internal node y waits until it receives the list of best swap edges from each of its children. Based on the received information and on its local swap edges, it computes its best swap edge for $(y,p(y))$; it then computes for each ancestor a the best candidate for $(a,p(a))$; finally, it sends the list of those edges to its parent (if different from r).

To ow ow i n i l o i i n o olv i o l , w n o i y ow onv i on . T iff n in ol ion : (i) o ion o w in onv , n (ii) i ion l in o ion, o on n i , o o ni o n o o wi w .

In ollowin , w will no y (x) o i o no x; i i n li o o ( ), w in i w o $(x,p(x))$; v l o o j iv n ion o in w $(x,p(x))$ n i wi ; n li o o i o i l o l in olv . Mo ov , l (x) w li o i o n o o x; i i li o o ( ) in i in o no $a \in A(x)$ (o in l) n i o $(a,p(a))$ (o in), n v l o o j iv n ion ( ); i in (x).

L i in il o ion x y no x. Fi o ll x o w o $(x,p(x))$ y on i in $InS(x)$ o ll lo l w o $(x,p(x))$ n o w n i o i o i il n (Al o i MyBSE). T n o n o a i o , on w in T_x, n i o $(a,p(a))$ (Al o i MyABSE). No w x o o i n o n wo n n l w o yi n o w n. x Al o i MyBSE.

```
MyBSE
(* Algorithm for node $x$, where $e = (x, p(x))$ is the link to be swapped *)
1. Determine which of $x$'s incident edges are swap edges for $(x, p(x))$; i.e., $x$ constructs the set $InS(x)$.
2. For each swap edge $e_i = (x, y_i) \in InS(x)$, compute the value of the objective function via $e_i$, and the value of the other attributes and insert them together with $e_i$ in $SL(x)$.
3. If $x$ is not a leaf, from each $ASL(x_j)$ received from $x_j \in C(x)$, extract $(e_j, value, attributes, x)$ (or $NIL$, if no such record exists), and insert $(e_j, value, attributes)$ in $SL(x)$ (or $NIL$).
4. Sort $SL(x)$ in non decreasing order of $value$. The minimal element of $SL(x)$ gives one of the best swap edges for $x$ and the value which minimizes the objective function.
```

```
MyABSE
(* Algorithm for node $x$ *) For each ancestor node $a \in A(x)$:
1. Select the swap edge $e_i \in SL(x)$ which is also a swap edge for $(a, p(a))$, if any, with the minimal value of $value$, and consider its record $(e_i, v_i, attributes, a)$.
2. For $x_j \in C(x)$, $1 \leq j \leq h$, let $(e_j, v_j, attributes, a)$ be the record from $ASL(x_j)$. Update the values of $v_j$ and of the $attributes$ in relation to node $x$. Consider the set of the updated records $\{(e_j, v_j, attributes, a) \cup (e_i, v_i, attributes, a)\}$, $1 \leq j \leq h$, where $(e_i, v_i, attributes, a)$ is the record computed in Step 1. Select from this set the record $(\overline{e}, \overline{v}, attributes, a)$ with minimal $value$, if any, and insert it, in $ASL(x)$ (to be sent to $x$'s parent); if no record can be selected, insert $NIL$ in $ASL(x)$.
```

7.2 Identifying a Swap Edge

In o o no o i i on o i in i n i w i i i n o , in onv , in o ion oll in o .

Property 3. $(u, v) \in E \backslash E(T)$ $u \in T_u$ $v \in T \backslash T_u$ w $(x, p(x))$, $x \in A(u)$, u,

P o y 3 iv o , o , u now ll i n o . v i n i no w o $e = (x, p(x))$, i i no i l o non o $a \in A(x)$.

8 The F_{sum} Problem with O(n) Messages

P o l F_{sum} i olv wi ino o i ion o onv P o Al o i BSL.

To o , no z n o i ion l in o ion: i n $d_{T'}(z,r)$ in $T_{e/e'}$ o on i w e' o $(z,p(z))$; w i $W(T_z)$ o T_z; n o no $n(T_z)$ in . T o o li (z) will v o : $(edge, F_{sum}(T_z), \{d_{T'}(z,r), W(T_z), n(T_z)\})$; i (l l in i in n o) o in o o (z).

T $n(T_z)$ n $W(T_z)$ ily o in iv ly o v l n o z y i il n z_j, n o w i o (z_j, z). N ly: $n(T_z) = \sum_{z_i \in C(z)} n(T_{z_i}) + 1$; n $W(T_z) = \sum_{z_i \in C(z)} W(T_{z_i}) + \sum_{z_i \in C(z)} n(T_{z_i})$ $w(z, z_i)$. I z i l $n(T_z) = 1$ n $W(T_z) = 0$.

L now ow ow o o n w v l o $F_{sum}(T_z)$, n o $d_{T'}(z,r)$ (S o MyBSE n o MyABSE).

Lemma 7. $(z,y) \in InS(z)$

( ) $F_{sum} = W(T_z) + n(T_z) \cdot (w(z,y) + d_{T'}(y,r)$

( ) $(e_i \neq NIL, F_{sum}(T_{z_i}), \{d_{T'}(z_i,r), W(T_{z_i}), n(T_{z_i})\}, z)$ z_i, $d_{T'}(z,r) = w(z,z_i) + d_{T'}(z_i,r)$,

$F_{sum}(T_z) = F_{sum}(T_{z_i}) + d_{T'}(z,r) + \sum_{j=1,j\neq i}^{h}(W(T_{z_j}) + n(T_{z_j})(w(z,z_j) + w(z,z_i) + d_{T'}(z_i,r)))$

A il n o z v l y in i o ion n n i i li o z. (i) ollow y L .

T n io o (ii) i n oo loo in Fi . I w e_i lon in o T_{z_i} i on i , ll no in T_{z_i} in in i i n

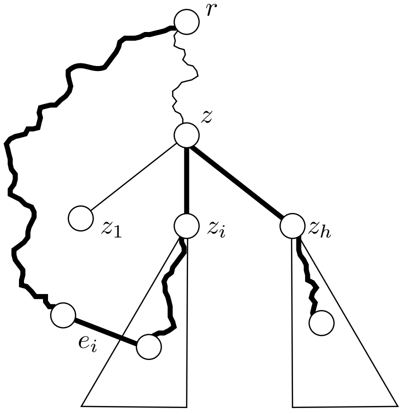


Fig. 2. Case (ii) in Lemma 7: the computation of $F_{sum}(T_z)$ via the swap edge e_i. The thick line represents the path to the root via e_i

o oo , n y on i o $F_{sum}(T_z)$ only o $F_{sum}(T_{z_i})$. No z on i o $d_{T'}(z,r)$. All o no in $T_{z_j}, 1 \leq j \leq h, j \neq i$, o oo , ollow o (z_j, z), (z, z_i) n n lly o w e_i.

T in onv now lon wi o in o o S ion , ill o on n i . W n lly v :

Theorem 3. $z \neq r$

( )

( ) $a \neq r$ a T_z

Fi o v , l o o , v y no iv l l o i n o (x r) n i n in w i w o i l n i n o (P o y n 3). T oo i y in ion on i $h(z)$ o T_z.

Basis. $h(z) = 0$; i. ., z i l . In i , on o on n on in only z, w il o on in ll o no . In o wo , only o i l w in i n on z. T , z n o ly o i w y o in v l o i n in oin (i) o L , ovin (i). I n l o i i ly in w wi o ll o i n o n o o v l o in oin (ii) o L , n l , o n o , n i .

Induction step. L o ol o ll no z wi $0 \leq h(z) \leq k-1$; w will now ow i ol o z wi $h(z) = k$. By in iv y o i , i iv o il y n i o n o o $y \in C(z)$, in l in z i l . H n , on li n on lo lly v il l $InS(z)$, z n o ly in i o i l w , w ll i i l w o o i n o .

Theorem 4. F_{sum} $O(n)$ $O(n_r^*)$

T o ollow i i ly o P o i n 3, n o , y L , ill v on n i .

9 The F_{max} and F_{incr} Problems with O(n) Messages

W will ow ow F_{max} i olv y BSL. T v l o ini i i xi l i n o no in T_z o oo vi w e_i. Si il ly o F_{sum}, w n o o in iv ly wo v l ; n ly, i n o z o oo vi e_i, $d_{T'}(z,r)$, n xi l i n o no in T_z o z, i $mD(T_q, z)$, wi $q \in C(z)$. T li (z) i now o o o o o o l n ; n ly: $(edge, F_{max}(T_z), \{d_{T'}(z,r), mD(T_q,z)\})$; (z) on in in o ion, l l .

L now ow ow o o n w v l o lon n w w e_i (S o MYBSE) n ow o o v l w n w n i o il i on i . T o ion o l o in S o MYABSE. W v :

Lemma 8. T_{z_k} T_z r $mD_2(z) = max_{q \neq k}(mD(T_q, z)$ T_{z_j} z $z_j \in \{C(z) \setminus z_k\}$ $(z,l) \in InS(z)$

( ) $F_{max}(T_z) =$ $\mathrm{x}_{q \in C(z,T)}(mD(T_q,z) + w(z,l) + d_{T'}(l,r))$
( ) $(e_s \neq NIL, F_{max}(T_{z_s}), \{d_{T'}(z_s,r), mD(z_s)\}, z)$ z_s $d_{T'}(z,r) = (w(z,z_s) + d_{T'}(z_s,r))$ $s = k$ $F_{max}(T_z) =$ $\mathrm{x}(F_{max}(T_{z_s}), mD_2s(z) + d_{T'}(z,r))$ $F_{max}(T_z) =$ $\mathrm{x}(F_{max}(T_{z_s}), mD(z) + d_{T'}(z,r))$

A il n o z v l y in i o ion n n i i li o z. F o v l z n o xi i n o no in T_z, n (i) ollow i i ly. Fo (ii), i w e_s o no lon o T_{x_k}, xi l i n i iv n y xi l v l on $F_{max}(T_{z_s})$ n $(mD(z) + d_{T'}(z,r))$. wi , ll no in T_{x_k} in in i i n o oo ; o ll o no (in $T_j, 1 \leq j \leq h, j \neq k$), ll no , o o oo o o (z_j, z), (z, z_k), n n lly o w z_s. H n , in i , o o i n o no w v o on i no xi l i n no lon in o T_{x_k}, w o i n i $mD_2(z)$.

T , i ollow :

Theorem 5. $x \neq r$

( ) $(x, p(x))$ F_{max}
( ) $a \neq r$ v T_u

F_{incr} n olv wi i l x n ion o ol ion o F_{max}.

Theorem 6. F_{max} F_{incr} $O(n)$ $O(n_r^*)$

I ollow i i ly o P o i n 3, n o L .

References

1. D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *CRC Handbook of Algorithms and Theory, CRC Press, 1997.*
2. A. Di Salvo and G. Proietti. Swapping a failing edge of a shortest paths tree by minimizing the average stretch factor. *Proc. of 10th Colloquium on Structural Information and Communication Complexity* (SIROCCO 2004) 2004.

3. P. Flocchini, T. Mesa, L. Pagli, G. Prencipe, and N. Santoro. Efficient protocols for computing optimal swap edges. *In Proc. of 3rd IFIP International Conference on Theoretical Computer Science* (TCS 2004), 2004, to appear.
4. A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79:43-59, 1988.
5. H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro. Polynomial-time computable backup tables for shortest-path routing. *Proc. of 10th Colloquium on Structural Information and Communication Complexity* (SIROCCO 2003), 163–177, 2003.
6. H. Mohanty and G.P.Bhattacharjee. A distributed algorithm for edge-disjoint path problem *Proc. of 6th Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS), 44-361, 1986.
7. E. Nardelli, G. Proietti, and P. Widmayer. Finding all the best swaps of a minimum diameter spanning tree under transient edge failures. *Journal of Graph Algorithms and Applications*, 2(1):1–23, 1997.
8. E. Nardelli, G. Proietti, and P. Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algoritmica*, 35:56–74, 2003.
9. P. Narvaez, K.Y. Siu, and H.Y. Teng. New dynamic algorithms for shortest path tree computation *IEEE Transactions on Networking*, 8:735–746, 2000.
10. L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, 3rd Edition*. Morgan Kaufmann, 2003.
11. G. Proietti. Dynamic maintenance versus swapping: An experimental study on shortest paths trees. *Proc. 3rd Workshop on Algorithm Engineering* (WAE 2000), 207–217 2000
12. R. E.Tarjan. Application of path compression on balanced trees. *Journal of ACM*, 26:690–715, 1979.

SRF TCP: A TCP-Friendly and Fair Congestion Control Method for High-Speed Networks

M i o F [1], F i i Hi o [1], To oy H no[2],
Hi o i S i no[1], n K n-i i [1]

[1] Faculty of Science and Technology, Keio University,
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa, Japan
{fukuhara, hirose, shigeno, okada}@mos.ics.keio.ac.jp
[2] NTT Access Network Service Systems Laboratories,
1-6 Nakase, Mihama-ku, Chiba-shi, Chiba, Japan
hatano.tomoya@ansl.ntt.co.jp

Abstract. TCP Reno congestion control carries two issues. First, its performance is poor in high-speed networks. To solve this TCP Reno drawback, HighSpeed TCP and Scalable TCP were proposed. However, the fairness between these proposed TCP and TCP Reno is not considered, when both connections coexist. Second, TCP Reno connections share bandwidth unfairly, when TCP flows with different RTTs use the same link. Many approaches have been proposed to solve this issue. However, no single method has been proposed to solve both issues. In this paper, we propose Square Root Fair TCP (SRF TCP). SRF TCP congestion control (1) sends packets efficiently in high-speed networks, (2) is TCP-friendly with TCP Reno and (3) shares fair bandwidth between flows with different RTTs. We evaluate the capabilities of SRF TCP through computer simulations and compare it with TCP Reno, High-Speed TCP and Scalable TCP.

1 Introduction

n ly, n in w i n l i- n i ion n y vol ion o o i l n wo . An o o vi in n in i llin y y y . M ny vi w i li v io n will o now on n i- n i ion i in wi ly. Mo ov , lli n wi l lin wi i l n y o in o n l. T P on ion on ol i n o i n v n i n wi- l y o in .

W n n T P, T P no, i in i- n wo , i nno lin n wi ff iv ly. In o o olv i o l n o , Hi S T P[1] n S l l T P[ ] w o o . T o o l v o l o i n wi T P no. W n T P no onn ion n o o T P onn ion o xi in lin , o o T P onn ion o in n wi iv ly.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 169–183, 2005.

In T P no, iff n in flow' TT n i n wi llo ion o flow. M ny ni w i i ov i n n o v n v lo . How v , ni i i n wi T P no n ion o i - n i ion.

A ol ion o o l , lo o o in on AQM[3,] o o . SFQ[5], w i i in y n wo , i l o o o . Ano o o i - n wo , n y X P[6], x li i - o o o on ion on ol. How v , o o l n o n o o n o n n wo . W v i o l y T P on ion on ol o l i ov in T P, i only n o . TF [ ], Bino i l on ion on ol[ ] n ny AIMD ni [9, 10] o T P- i n ly on ion on ol o . How v , ni nno n i i n ly in i - n wo .

In i , w o o S oo F i T P (S F T P), w i olv o l o T P no. In S F T P, w n n A K i iv , win ow in o o ion l o flow' TT n inv ly o o ion l o oo o i on ion win ow. W n i lo , win ow o o ion l o oo o i on ion win ow. S F T P i i l o i - n i ion, i ov i n wi T P no n i ov n i n wi llo ion y iff n in flow' TT.

T i o ni ollow . In ion , w i T P no on - ion on ol n i o l . W i l wo on T P on ion on ol ni o i - n i ion in ion 3, n on n i n y iff n in flow' TT in ion . S ion 5 o o S F T P. In ion 6, w n i l ion l o S F T P. S ion on l .

2 TCP Reno

2.1 Congestion Control

T P no in on ion win ow (W) y $1/W$ o iv A K n i in l o lo v n . T P no on ol on ion win ow ollow :

$$\text{In} \quad : W = W + \frac{1}{W}, \qquad (1)$$

$$\text{D} \quad : W = W - \frac{1}{}W. \qquad (\;)$$

B on iv A K n lo , T P on ion on ol ni on ol on ion win ow o n i ff iv ly. A l , lo v n i io i lly. T on ion win ow n o (T) l o lo (p) n T P no on n ion i o ollow [11]:

$$W_{reno} = \frac{1.}{p^{0.5}}, \qquad (3)$$

$$T_{reno} = \frac{Size_{pac}}{RTT}\frac{1.}{p^{0.5}}, \qquad ( )$$

w i i $Size_{pac}$.

2.2 Problem in High-Speed Networks

F o ion (3), T P no l io on in on on ion win ow n i v y T P in li i nvi on n . Fo x l , o T P no onn ion wi 1500- y n 100 TT, i vin y- o o 10 wo l i n v on ion win ow o 3000 n , n n v o o $\times 10^{-10}$ o [1]. T i i n n li i i n o n n wo . I o n 0000 TT o ov on ion win ow lo v n o i n o lin n wi . T on w y T P no nno i v i o i i on ion win ow i in y i ni n ly ll i o 1 TT o , w il i i y l i o i lo .

2.3 Fairness Between Flows with Different RTTs

Fo T P no, win ow in i inv ly o o ion l o TT. A T P no onn ion wi lon TT in on ion win ow lowly. A onn ion wi low in in win ow nno n in o n wo on i ion o ly, w il onn ion wi in in win ow o v il l n wi . A l , T P no on ion on ol - ni l ni o i n wi . In i ion, o ion ( ), o i inv ly o o ion l o i TT ov i i i .

3 TCP Congestion Control Mechanism for High-Speed Networks

3.1 HighSpeed TCP

Hi S T P i o o o n i i n ly in i - n wo . I , W_L, W_H, n P_H. To n T P o i ili y, i on n ion T P no w n n on ion win ow i xi W_L, n Hi S on n ion w n n on ion win ow i n W_L. Hi S T P v on ion win ow W_H, w n lo i P_H. To i li y, i o y on n ion iv i lin on lo -lo l . T i l in ollowin on n ion, o v l o v on ion win ow n W_L:

$$logW = \frac{logW_H - logW_L}{logP_H - logP_L}(logp - logP_L) + logW_L. \qquad (5)$$

Hi S T P n n l in o in n n ion . T y $a(W)$ win ow in TT n $b(W)W$ win ow o lo v n . iv n o $b(W) = b_H$ o $W = W_H$, v l o $b(W)$ o o v l o $W > W_L$ n i . Hi S T P l $b(W)$ v y lin ly lo o W, n $a(W)$ n o ollow :

$$b(W) = \frac{logW - logW_L}{logW_H - logW_L}(b_H - 0.5) + 0.5. \qquad (6)$$

$$a(W) = \ W^2 p(W) \frac{b(W)}{- b(W)}. \qquad (\ )$$

In i o o l, $W_L = 3\ $, $W_H = \ 3000$, $P_H = 10^{-7}$ n $b_H = 0.1$ - o n . n o ion , Hi S T P on n ion i o ollow :

$$W_{hs} = \frac{0.119}{p^{0.835}}, \qquad (\ )$$

$$T_{hs} = \frac{Size_{pac}}{RTT}\frac{0.119}{p^{0.835}}. \qquad (9)$$

T i i ow Hi S T P li i n n i ion in i - n - wo . How v , inv o o ion o TT in ion (9) ov Hi - S T P n i n y iff n in flow' TT. Mo ov , i i in- i Hi S T P i o iv n T P no n i i n i n wi llo ion wi T P no. F o ion ( ) n (9), o o ion o o w n T P no n Hi S T P i o ollow :

$$\frac{T_{hs}}{T_{reno}} = \frac{0.09\ 1}{p^{0.335}}. \qquad (10)$$

Fo x l , o o o 10^{-6}, o o ion o o i 10.0, w i i n i .

3.2 Scalable TCP

S l l T P l o i ov T P o n in i - wi n - wo . S l l T P on ol on ion win ow ollow :

$$\text{In} \quad : W = W + 0.01, \qquad (11)$$

$$\text{D} \quad : W = W - 0.1\ 5W. \qquad (1\ )$$

Fo S l l T P, on ion win ow i in y $0.01W$ o o ion l o i on ion win ow TT. A o in o [ ], ov y i lo i 13. TT, w i i o o ion l o TT n in n n o on ion win ow. n o n , o T P no, on ion win ow i in y 1 n TT n ov y i lo i o o ion l o o TT n on ion win ow. A S l l T P onn ion ov i on ion win ow in o i v n w n i l on ion win ow. T o , S l l T P i n o n wi in i - n wo . S l l T P on n ion i o ollow :

$$W_{sca} = \frac{0.0\quad 5}{p} \tag{13}$$

$$T_{sca} = \frac{Size_{pac}}{RTT}\frac{0.0\quad 5}{p} \qquad (1\ \ )$$

S l l T P l o n i n y iff n in flow' TT o - in o ion (1). Mo ov , o ion ( ) n (1), o o ion o o w n T P no n S l l T P i o ollow :

$$\frac{T_{sca}}{T_{reno}} = \frac{0.060}{p^{0.5}}. \tag{15}$$

Fo x l , o o o 10^{-6}, o o ion o o i 60.9. T n i n wi T P no i o io n o Hi S T P n i i l o in wo l .

3.3 The Relationship Between the Congestion Window and *p*

Fi 1 ow l ion w n *p* n *W* o T P no, Hi S T P n S l l T P. A *p*, win ow iff n w n T P no n Hi S T P i ll n o T P no n S l l T P. I ow Hi S T P i o T P- i n ly n S l l T P. How v , w n *p* ll , Hi S T P n i n wi llo ion. To i ov i n wi T P no o o Hi S T P, on ion win ow o n w T P o in Fi 1 o l l n Hi S T P on ion win ow.

4 Unfairness by the Difference in Flow's RTT

4.1 Needs for Improvement

S lli lin o in o n l n i i n i ion wi lin will l o i x ly. How v , lli lin v lon o n i o ion l y. Fo T P no, i oo i o onn ion wi lon TT o ov lv on ion win ow, i lly in i i n wo . Hi S T P l o n 1 3 on o onn ion wi 1 on TT o ov on ion win ow in 10 n wo . n i o ion, ov y i i oo lon o i n o lin n wi . In o o onn ion wi lon TT o ov on ion win ow n i n o i - n wo , win ow in n l o i in n n o TT i n .

4.2 Related Works

M ny o v n o o o olv n i n y iff n in flow' TT. F n l. [1] T P win ow in on n n ov

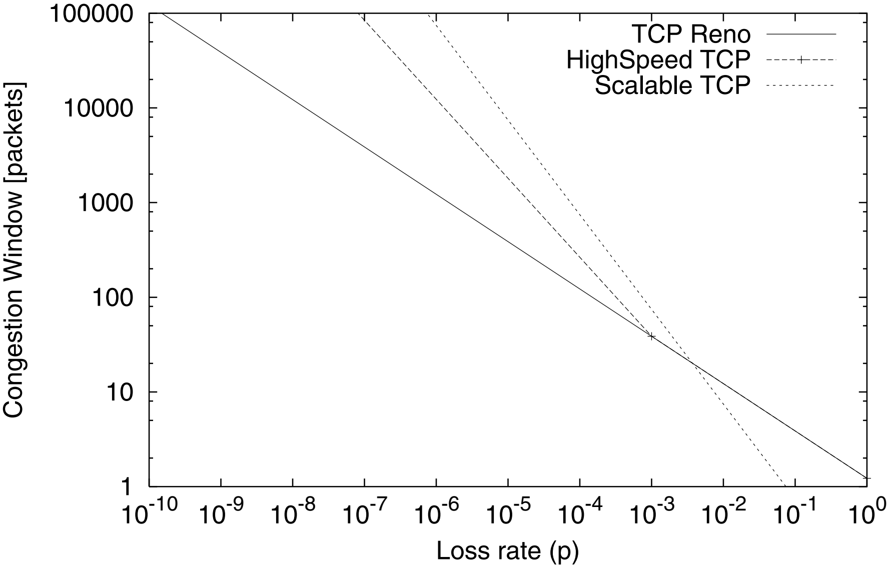


Fig. 1. The relation between p and W of TCP Reno, HighSpeed TCP and Scalable TCP

infl n o TT iff n . In i o o l, n i o l wi T P no in .

H nn l. [13] o o on ion on ol o w i in on ion win ow o iv A K ollow :

$$W = W + \frac{1}{W} S_{wnd}, \tag{16}$$

$$S_{wnd} = S_{wnd} + \beta RTT^2. \tag{1\ }$$

T win ow in i o o ion l o RTT^2 o i n wi - w n flow wi iff n TT . Al o o o on ion on ol i n wi T P no, no i ov n i o li i n n i ion in i - n wo .

5 Proposed Congestion Control Method

W o o S oo F i T P (S F T P) w i n n i wi i n o lin n wi in i - n wo , olv n i n y iff n in flow' TT n i ov i n wi T P no.

In i , o n w on ion on ol o , l on ion win ow in n ollow :

$$\text{In} \quad : W = W + aW^{\alpha}, \tag{1\ }$$

$$\text{D} \quad : W = W - bW^{\beta}. \tag{19}$$

5.1 Resolution of α and β

To n i i n ly in i - n wo , v l o α n β - i . T P no, w i i A i iv In M l i li iv D (AIMD) on ion on ol, o $\beta-\alpha=$. An TF n Bino i l on - ion on ol l o o $\beta-\alpha=$. In i - n wo , AIMD on ion on ol i on ion win ow i ni n ly w il win ow in i ll. A o in o [1], $\beta-\alpha=1$ i o n o i l on n- ion. W n $\beta-\alpha=1$, lo i in n n o on ion win ow n i n n on only TT, w i i i l o i - n i ion.

T oi o α n β o on i $(\alpha,\beta)=(-1,0),(0,1),(-1/ ,1/)$. W n $(\alpha,\beta)=(-1,0)$, win ow i *constant*. Fo l on ion win ow, win ow l iv ly oo ll, w il win ow l iv ly oo l o ll on ion win ow. T , $(-1,0)$ i no i l o lin n wi i n ly in o no l n i - n wo . W n $(\alpha,\beta)=(0,1)$, w i i o y S l l T P, win ow in TT l iv ly oo l l on ion win ow. W n o - i n o in i n o ll v l , win ow in TT l iv ly oo ll ll on ion win ow.

Fo $(\alpha,\beta)=(-1/ ,1/)$, win ow in ow ily n win ow o ily. T win ow in n n j o i l v l o v y on ion win ow. T i y n i n ly o n i ion in o no l n i - n wo . T o , w o $(\alpha,\beta)=(-1/ ,1/)$ o n w on ion on ol o . n o ion , on n ion i o ollow :

$$W=\frac{a}{bp}, \tag{ 0}$$

$$T=\frac{Size_{pac}}{RTT}\frac{a}{bp}. \tag{ 1}$$

5.2 Resolution of a and b

T on ion on ol o $(\alpha,\beta)=(-1/ ,1/)$ in on ion win ow y $aw^{\frac{1}{2}}$ TT n i y $bw^{\frac{1}{2}}$ o lo v n . F o ion (1), v o i inv ly o o ion l o TT. In o o o o in n n o TT, $b/a\propto RTT$ i i .

W n win ow in i o o ion l o TT, win ow in n o v l , in n n o flow' TT. T $a=a'RTT$ o i l .

T v l o a' n b i o ollowin o . (1) $b=15$ i n o win ow o o n $0.05W$ o 10 n . ( ) In o o on ol o n Hi S T P 10 lin , $p=10^{-7}$ i i l w n $W=$ 3000. F o ion (0), w $b/a'=1$. T n ov y i will 1 [*sec*] n $a'=1.$ 5 i i . W n $a'=1.$ 5, win ow in 1 TT i 36 o 10 n , w i i no oo l win ow in . T , win ow in n in S F T P i ollow :

$$\text{In} \quad : W = W + 1.\ 5RTTW^{-\frac{1}{2}}, \qquad (\ )$$

$$\text{D} \quad : W = W - 15W^{\frac{1}{2}}. \qquad (\ 3)$$

An S F T P on n ion i o ollow :

$$W_{srf} = \frac{0.0\ 33RTT}{p}, \qquad (\ )$$

$$T_{srf} = \frac{0.0\ 33Size_{pac}}{p}. \qquad (\ 5)$$

In ion (5), T_{srf} i in n n o TT n S F T P i n wi w n flow wi iff n TT .

F o ion ( ) n (5), o o ion o o w n T P no n S F T P i o ollow :

$$\frac{T_{srf}}{T_{reno}} = \frac{0.0069}{p^{0.5}}, \qquad (\ 6)$$

w $RTT = 100ms$ i . Fo o o 10^{-6}, o o ion o o i .0. T i n o S F T P wi T P no i i ov in o i on o o Hi S T P o S l l T P.

5.3 Comparison to TCP Reno

S F T P i o o n T P no. W n $W \le \frac{0.64}{RTT^2}$, win ow in o S F T P ($1.\ 5RTTW^{-\frac{1}{2}}$) i l n $1/W$, w i i win ow in o T P no. W n $W \le 900$, win ow o S F T P ($15W^{\frac{1}{2}}$) i o n $0.5W$, w i i win ow o T P no. n o ion , S F T P o wo n T P no. T o , win ow in n o S F T P o v l T P no.

W n $\frac{0.64}{RTT^2} \le W \le 900$, S F T P in on ion win ow y $1.\ 5RTTW^{-\frac{1}{2}}$ o v y iv A K n i y $0.5W$ o lo v n . S F T P on n ion i o ollow :

$$W_{srf} = \frac{1.6\ RTT^{0.667}}{p^{0.667}}, \qquad (\ )$$

$$T_{srf} = \frac{Size_{pac}}{RTT^{0.333}}\frac{1.6}{p^{0.667}}. \qquad (\ )$$

W n $\frac{0.64}{RTT^2} \le W \le 900$, o o ion o o w n T P no n S F T P i o ollow :

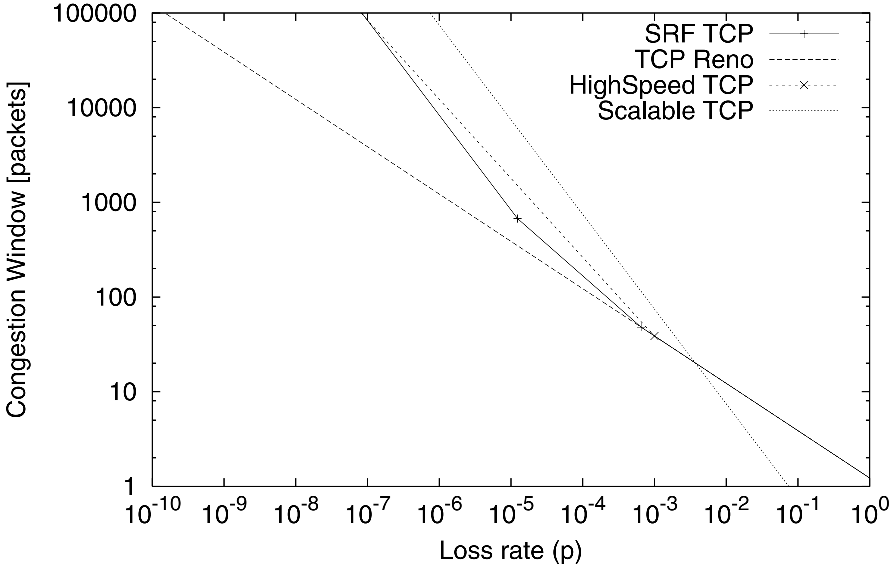


Fig. 2. The relation between p and W of SRF TCP

$$\frac{T_{srf}}{T_{reno}} = \frac{0.\ 3}{p^{0.166}}, \qquad (\ 9)$$

w $RTT = 100ms$ i .

W n on ion on ol o in ion (30) n (31) S F T P.

$$\text{In} \quad : W = W + max(1.\ 5RTTW^{-\frac{1}{2}}, \frac{1}{W}), \qquad (30)$$

$$\text{D} \quad : W = W - min(15W^{\frac{1}{2}}, \frac{1}{\ }W). \qquad (31)$$

5.4 Comparison to HighSpeed TCP and Scalable TCP

Fi ow l ion w n p n on ion win ow o S F T P. F o , v y p, win ow iff n w n T P no n S F T P i ll n win ow iff n w n T P no n o T P . T o , S F T P i on i o i ov i n wi T P no.

A i ion lly, on ion win ow o S F T P i lo o o Hi - S T P o i n o lin n wi ll p.

6 Evaluation

W v l o o S F T P o o i l ion in o i- on o T P no, Hi S T P n S l l T P wi N wo Si l o v ion (n -) [15]. T i l ion o olo y i own in Fi 3. In i o l, lin o l n n wi , o n i o ion l y n n o T P onn ion n . All i l ion w n lon no o n y on i n vio .

6.1 Link Utilization

Fi n 5 ow lin ili ion w n v yin n o flow n i in 100 M n 1 o l n lin , iv ly. T o n i o ion l y i 100 . F o , S F T P ili lin n wi o i n ly n T P no n l i n ly n Hi S T P n S l l T P in 100 M w n only on flow i n i . A n o flow in , iff n w n S F T P n Hi S T P ll, w il S l l T P ili n wi i n ly.

A own in 5, S F T P ili lin n wi i n ly Hi S T P n o n T P no, l o n o flow . n o n , S l l T P nno ili lin n wi ll n o flow , S l l T P in on ion win ow i n l lo n lon ov y i .

Fi 6 ow lin ili ion w n on flow i n i o- l n n wi in . S own in 6, S F T P i n o lin n wi wi n wi . In , S F T P lo o Hi S wi l on ion win ow, n i i l ion, o- n o S F T P i l o o Hi S T P. A n ow

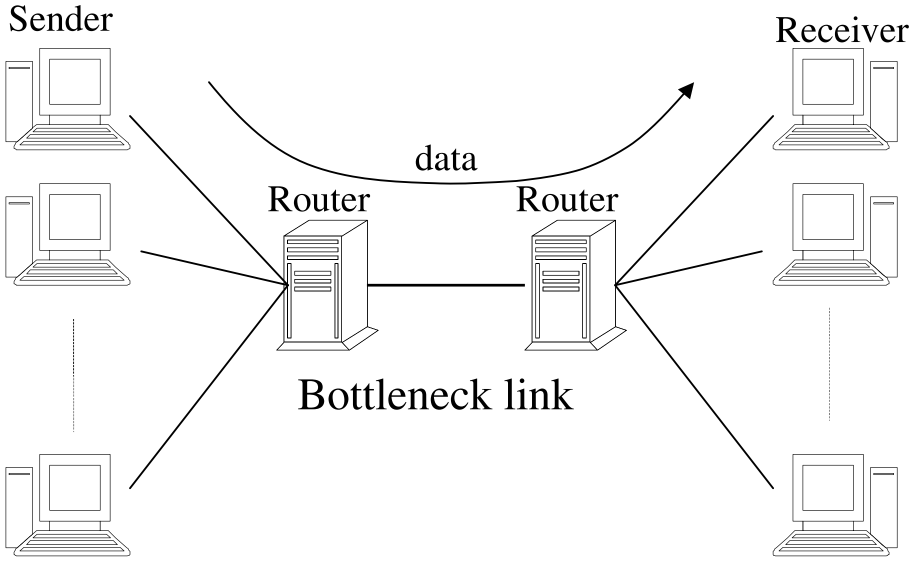


Fig. 3. Simulation model

n wi , S F T P ili n wi T P no in o o in T P o i ili y.

T o , S F T P ili lin n wi wi n wi n in T P o i ili y n ow n wi . T i o n i o v no ly ll n o flow .

6.2 Fairness with TCP Reno

Fi ow i n w n on T P no flow n on S F T P, Hi S T P o S l l T P flow n i o o l n n wi

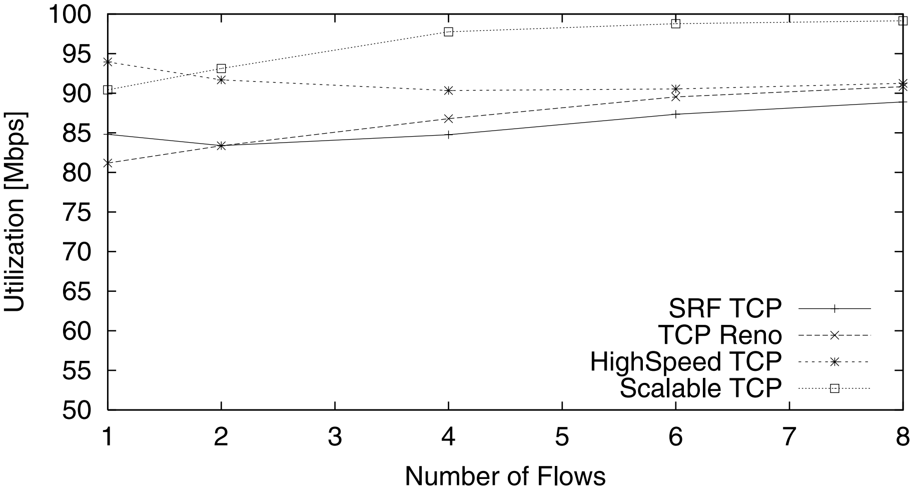


Fig. 4. Link utilization of 100 Mbps bottleneck link

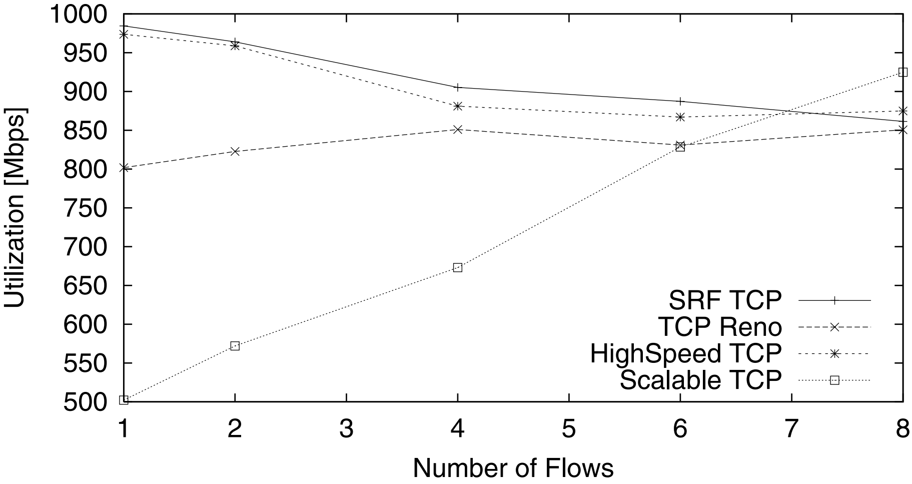


Fig. 5. Link utilization of 1 Gbps bottleneck link

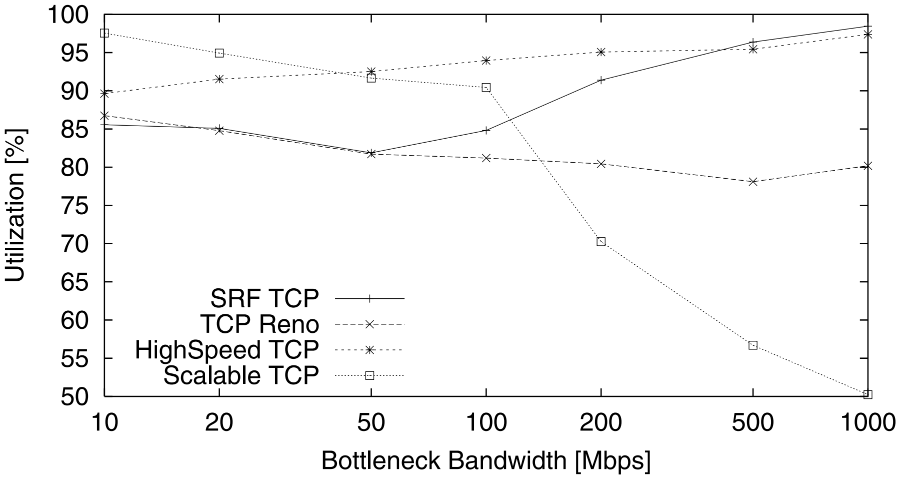


Fig. 6. Link utilization when one flow is transmitted

in . F i n i n o o ion o o n o
T/T_{reno}, w T i o o T P in lin wi T P no.
W n i n i l o 1, wo onn ion n wi n
i i i i ion.

A own in , v l o ll T P lo o 1 n y
i n wi T P no n ow n wi . A wi n wi , w il
Hi S T P n S l l T P iv on ion on ol n

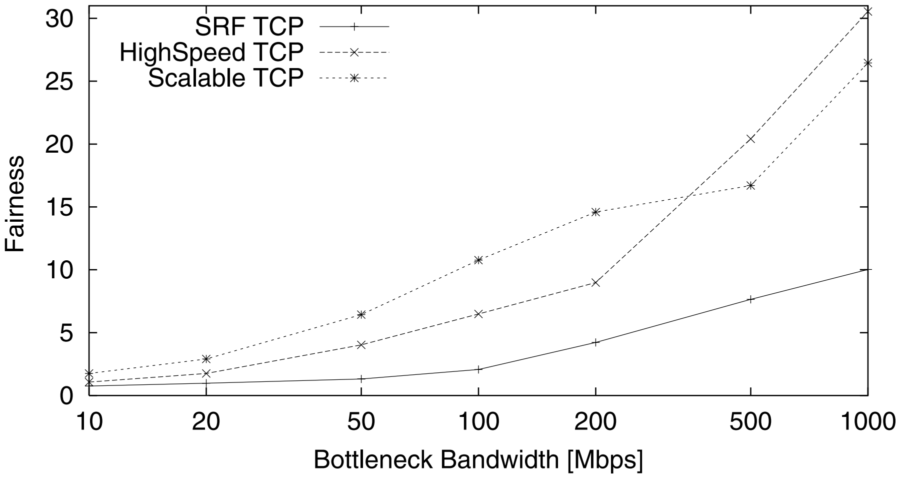


Fig. 7. Fairness with TCP Reno

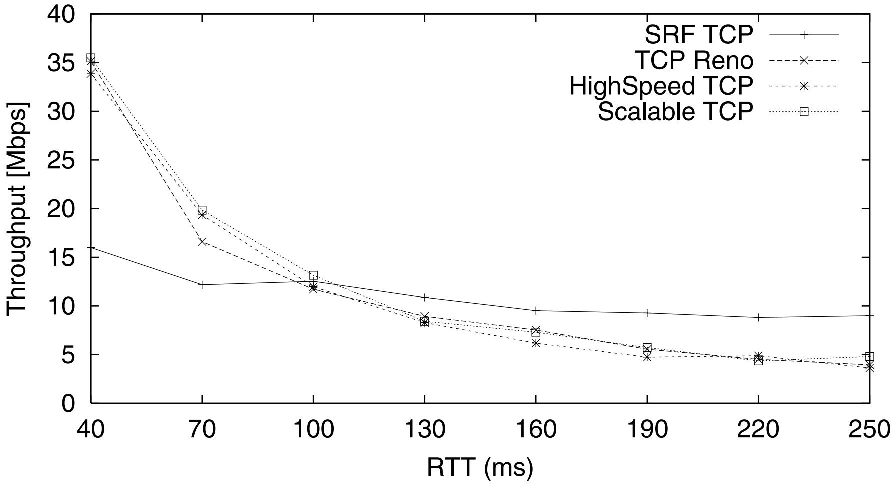


Fig. 8. Each throughput of flows with different RTTs in 100 Mbps

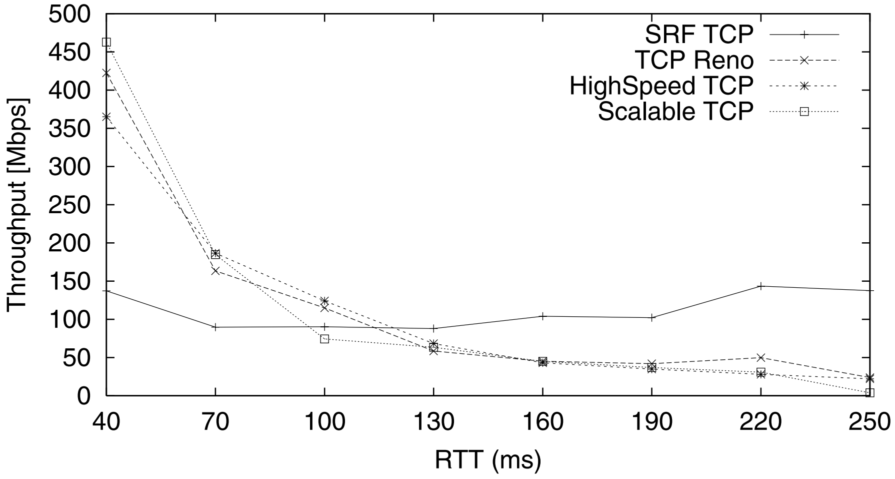


Fig. 9. Each throughput of flows with different RTTs in 1 Gbps

n i n wi wi T P no, S F T P i ov i n wi T P no. F o l , S F T P ov o n o i ili y wi T P no.

6.3 Fairness Between Flows with Different RTTs

Fi n 9 ow o o flow wi iff n TT in 100 M n 1 o l n n wi , iv ly. TT o flow #1 o # 0, 0, 100, 130, 160, 190, 0 n 50 , iv ly.

F o , o o T P no, Hi S T P n S l- l T P i ; flow wi o TT o n wi n flow wi lon TT. n o n , o S F T P, o o flow i l o v l in n n o i TT. T i i o o S F T P i in n n o i TT in ion (5), w il o o T P no, Hi S T P n S l l T P inv ly oo ion l o TT in ion ( ), (9) n (1). T o , S F T P ov o n i n w n flow wi iff n TT .

7 Conclusion

In i , w o o S oo F i T P (S F T P) o li i n n i ion in i - n wo , i n wi T P no n i n w n flow wi iff n TT . S F T P on ion on ol o inon ion win ow inv ly o o ion l o oo o on ion win ow n o o ion l o i TT o iv A K, n -on ion win ow o o ion l o oo o on ion win ow o lo v n .

F o v l ion o S F T P o o i l ion, ollowin oin ov . Fi , S F T P i n o lin n-wi wi n wi . I i o vio ly o v w n lin i y ll onn ion . S on , S F T P i ov i n wi T P no in o i on o Hi S T P n S l l T P w n onn ion i n-i wi T P no onn ion. ff iv n o i ov n l wi n wi . Fin lly, S F T P i n wi w n flow wi iff n TT in n n o i TT .

Al o n wi ili ion, i n wi T P no n i n -w n flow wi iff n TT i ov , ll i no olv -ly. Mo i ov n n o i .

Acknowledgement

T i wo w o y i l n o .

References

1. S. Floyd, “HighSpeed TCP for Large Congestion Windows”, Internet Draft draft-floyd-tcp-highspeed-02.txt, February 2003.
2. Tom Kelly, “Scalable TCP: Improving Performance in HighSpeed Wide Area Networks”, December 2002. http://www-lce.eng.cam.ac.uk/ ctk21/scalable/.
3. T. J. Ott, T. V. Lakshman, and L. H. Wong, “SRED: Stabilized RED,” in Proceedings of the IEEE INFOCOM ’99, March 1999.
4. M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, “Tuning RED for web traffic,” in Proceedings of the ACM SIGCOMM 2000, August 2000.
5. I. Stoica, S. Shenker, and H. Zhang. “Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocations in high speed networks,” In Proc. of ACM SIGCOMM ’98, August 1998.

6. D. Katabi, M. Handley and C. Rohrs. “Congestion Control for High Bandwidth-Delay Product Networks,” ACM SIGCOMM 2002, August 2002.
7. Mark Handley, Jitendra Padhye, Sally Floyd, and Joerg Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification,” RFC 3448, January 2003.
8. D. Bansal and H. Balakrishnan, “Binomial Congestion Control Algorithms,” In Proceedings of IEEE INFOCOM 2001, April 2001.
9. Y. Richard Yang and Simon S. Lam, “General AIMD congestion control,” in Proceedings of ICNP, November 2000.
10. S. Floyd, M. Handley, J. Padhye, and J. Widmer. “Equation-based congestion control for unicast applications,” in Proceedings of the ACM SIGCOMM 2000, August 2000.
11. J. Padhye, V. Firoiu, D. Towsley and J. Krusoe, “Modeling TCP Throughput: A Simple Model and its Empirical Validation,” ACM SIGCOMM ’98, P303–314, 1998.
12. Fang, W. and Peterson, L.: TCP mechanisms for Diff-Serv Architecture, *Princeton University, CS Dept., Technical Report TR-605-99* (1999).
13. T. Hamann, J. Walrand, “A New Fair Window for ECN Capable TCP (New-ECN),” in Proceedings of lEEE INFOCOM 2000, March 2000.
14. T. J. Ott, “ECN protocols and the TCP paradigm,” in Proceedings’ of lEEE INFOCOM 2000, March 2000.
15. The network simulator ver.2 - ns-2. http://www.isi.edu/nsnam/ns/

Embedded Systems - Challenges and Work Directions

Jo Si i

Verimag and ARTIST2 European Network of Excellence

Abstract. Embedded Systems are components integrating software and hardware jointly and specifically designed to provide given functionalities. These components may be used in many different types of applications, including transport (avionics, space, automotive, trains), electrical and electronic appliances (cameras, toys, television, washers, dryers, audio systems, cellular phones), power distribution, factory automation systems, etc.

Their extensive use and integration in everyday products marks a significant evolution in information science and technology. A main trend is the proliferation of embedded systems, that should work in seamless interaction while respecting real-world constraints.

Embedded systems have a number of specific characteristics, which play a role in structuring the technical domain including criticality, reactivity and autonomy.

The coming generations of embedded systems - primarily used in mass-market products - need development methods and tools allowing to jointly consider functionality, quality, physical implementation, and market constraints: The need to jointly consider functional and extra-functional constraints leads to a system-centric approach to development. Here, the main focus is the end result: a system as the combination of hardware and software, in interaction with its physical environment.

Current methods and tools do not allow system-centric approaches. These approaches raise difficult, fundamental research problems, which are the basis of an emerging theory that should bring together information and physical sciences. Information sciences consider models of computation based on abstract notions of machines (e.g., automata, complexity and computability theory, algorithms, etc.), that do not take into account physical properties of computation (e.g., execution times, delays, latency, etc.). There is no unified theory allowing to predict the behavior of an application software on a given execution platform which determines execution speed and other dynamic properties of the application.

System-centric approaches raise two grand challenges common to all the activities of system development. The first is theory and tools for rigorous component-based engineering. This determines our ability to build complex systems from simpler ones by mastering their complexity. The second is intelligence, a long term vision for systems that are able to analyze and adapt their behavior according to changes of their environment.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 184–185, 2005.

We discuss specific work directions in system development activities to meet these challenges, including modeling, programming and compilation, operating systems design, controller synthesis, testing and verification.

Reference

1. ARTIST: "Selected Topics in Embedded Systems Design: Roadmaps for Research", http://www.artist-embedded.org/Roadmaps/

Comparison of Failures and Attacks on Random and Scale-Free Networks

J n-Lo ill [1], M i L y[1], n l n M ni n[2]

[1] LIAFA – CNRS – Université Paris 7 – 2 place Jussieu,
75251 Paris Cedex 05, France
Fax : 33 (0)1 44 27 68 49
{guillaume, latapy}@liafa.jussieu.fr
[2] CREA – CNRS – École Polytechnique – 1, rue Descartes,
75005 Paris, France
Fax : 33 (0)1 55 55 90 40
magnien@shs.polytechnique.fr

Abstract. It appeared recently that some statistical properties of complex networks like the Internet, the World Wide Web or Peer-to-Peer systems have an important influence on their resilience to failures and attacks. In particular, scale-free networks (*i.e.* networks with power-law degree distribution) seem much more robust than random networks in case of failures, while they are more sensitive to attacks.

In this paper we deepen the study of the differences in the behavior of these two kinds of networks when facing failures or attacks. We moderate the general affirmation that scale-free networks are much more sensitive than random networks to attacks by showing that the number of links to remove in both cases is similar, and by showing that a slightly modified scenario for failures gives results similar to the ones for attacks. We also propose and analyze an efficient attack strategy against links.

Keywords: Internet, Complex Networks, Random Graphs, Scale-Free Graphs, Resilience, Fault tolerence, Reliability, Network Topology.

Introduction

In n o n wo [1,] wi n no , o $\frac{n\cdot(n-1)}{2}$ o i l lin xi wi iv n o ili y p. In o wo , n o n wo i on o n no y oo in $m = p \cdot \frac{n\cdot(n-1)}{2}$ lin n o . In n wo , i i ion p_k ollow Poi on l w: $p_k = e^{-z}\frac{z^k}{k!}$ w z i v . In i iv ly, i i ion n o no v lo o v , n n o no wi iv n y x on n i lly w y o n .

How v , i n own n ly o l-wo l o l x n wo [3, ,5,6, , ,9], in i l In n [10], Wo l Wi W [,11] n P - o-P y [1], v ow -l w i i ion: $p_k \sim k^{-\alpha}$. In

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 186–196, 2005.

w v i , α i lo o .5. In i iv ly, i i ion n , i o no v low , n o no wi (v y) i i no n li i l .

Sin i iff n w n n o n wo n l-wo l o l x n - wo n i ov , on ffo n on n n in o i on n . n o o o i i i ni n ly infl n o n o n wo [,13,1 ,15,16,1], w i n o v ol- low . iv n n wo , on n o l i o il y n o ov l o no (o lin), w n i o l y ov l o i o o n no (o lin). T w y no (o lin) o n - in n i ll n . T li y o vi ovi y n wo n on i ion n o ly v l y i o i l onn o on n (n o in w i n o - ni in In n , o in n). T ili n o n wo o il o n n n ly y yin ow i o l onn o on n v i n ion o n o ov no (o lin). In i l , n wo i i o v i i o on n o i lin wi o i o n wo . In o wo , on n o o ion (wi o n wo i) o w ol n wo i onn . i i o in n wo i n y v n o o , o in n [16,1 ,1 ,19].

T o wi ly i y n in o in n n ly in [ ] n [13]. I on i in ovin no y in o o i . W will o i y. T ff o i y lo in Fi 1, o wi ff o il .

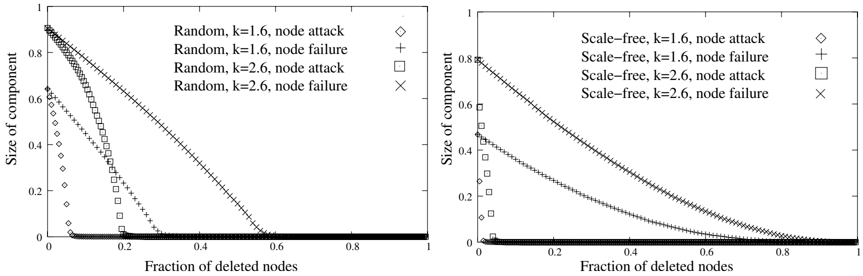


Fig. 1. Effects of random failures and attacks on random networks (left) and scale-free networks (right). The plots represent the size of the largest connected component as a function of the fraction of removed nodes. Different values of the mean degree k are considered

F o x i n ollowin o v ion n iv [,13]. Fi , i li iv iff n in vio o n o n l - n wo in o il : o n o n wo , i o l onn o on n o o o w n ni ion o no ov (i ion n), w o l - n wo , i v y lowly, n 0 only w n o no v n ov . T l - n wo o o ili n o il n n o n wo . How v , o o i o : l - n wo oll o i ly n n o n wo . T ow -l w i i ion o in In n , w i i o i v y ili n o il x ly n i iv o , v n n ll ’ [0].

Al o ov v y l ion o lin , w ow in S - ion 1 i i no i n o x l in li iv iff n w n il n o l - n wo . W n inv i i i- n (S ion) y o o in wo n w i , on in no n o in lin , n o in i ff wi o o l i l n il .

B o n in in o o , l y wo on o lo . T lo o x i n l l o in y i l ion v o i - l ion ov l n o l . T i i in n l n iv o n vio , i no l i l ion l o in on on in n y i ni n ly iff n in o (in i l in w on n ol v l o l - n wo).

on nin ol v l , w on i ol w w n v i o l onn o on n o n wo o ll n 5% o w ol . T lo n in ol in n ion o n o n o n wo , n x o- n n o l - n wo , w i in in on x . A l - n wo i onn i $\alpha \leq 3.$, o w will no in in w α i n i v l .

Fo lo o in ff o iff n il n o n o n l - n wo , w v o n o o n wo wi v . T v l w v o n 1.6 n .6, w i o on o l - n wo wi x on n 3 n .5 iv ly, n iv o v l in i .

In v l , w lo n i l v l ion o oxi ion o læ. T o læ v o n n o in n on in o - ion. B in o x i n ion i y n i , i i v l y i iff n o oxi ion v l , w i o l - o n in i iv .

All l - n wo v n n in l o i o o inin n wo wi i i i ion i in [1]. W v n- l - n wo wi N no n x on n α y win N w n 1 n N, ollowin ow -l w wi x on n α. T n i o n o ly onn . So oo in ollowin lin i o n o ly o n .

W l o n o in o w no ion : $\zeta(\alpha)$ i i nn ζ n ion, n y $\zeta(\alpha) = \sum_{k=1}^{\infty} k^{-\alpha}$. K- oni n , no y $H_K^{(\alpha)}$, i l o $H_K^{(\alpha)} = \sum_{k=0}^{K} k^{-\alpha}$. Fin lly, iv n i i ion p_k, w no y $\langle k\rangle$ n $\langle k^2\rangle$ n o n iv ly: $\langle k\rangle = \sum_{k=0}^{\infty} kp_k$ n $\langle k^2\rangle = \sum_{k=0}^{\infty} k^2 p_k$.

1 The Links Point of View

T l i l y ov i - no . Sin in l- n wo i i o n i y w n no , i no v v y l n o lin o . T o , on y won i i n y o on n wo i on n o n o ov i l n in o il . Li wi , on y won i l in ov l o o lin in l- n wo n in n o on i o iff n w n wo. T x l n ion lly v n o o y o o o iv n in i iv x l n ion o l n ov .

T i o i ion i o v l i y y o l i l n lin oin o vi w. In , l i l y n vi w y in lin , w lin j n o i no ov . T n, i o i n o on n n lo n ion o n o ov lin , Fi . In i , vio o n wo n n o lin ov l, il , i l o lo o i on.

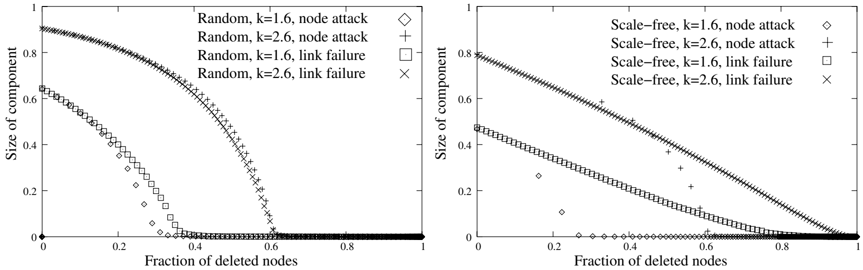


Fig. 2. The effects of the classical node attack when considering links, and of link failure, for random networks (left) and scale-free networks (right)

A l n , i lin y o i n n n- o ov l. T i n on o lly wi in o n v n v lo in [1 , 15]. F o i , on o in ol m_c o lin v o n o ly ov o n wo i :

$$m_c = 1 - \frac{\langle k\rangle}{\langle k^2\rangle - \langle k\rangle}$$

T i l n o in y ollowin onin : w n lin ov , i n i i ion o n wo . T n w i i ion n x li i ly o . Sin lin ov n o , n wo i n o n wo wi n w i i ion. T xi i ion [1] o i in i n wo i n o on n o no , n ov o l i o in o li ion o i i ion o n w i i ion o n wo .

I n o i n i y i ol p_c o . . . . il [1 ,]. T i n in i l lin il o no l- n wo oll . T o , l- n wo oll in l i l n i n y o i y i . . . o i ov ny lin . I n o lin ov n o ly, n n wo o no oll .

L now y o v l i ly i n y o i lin . T ion m_c o lin ov o n wo n o in nn w n on in [1 ,] o n o no . Fo ny n wo , ion $m(p_c)$ o lin ov in n i l o $s(p_c)^2 +$ $s(p_c)(1-s(p_c))$, w $s(p_c)$ n n o (lin ’ n- oin) o ov no . $s(p_c)$ n v l y ollowin ion [15, 1].

Fo l- n wo :

$$s(p_c) - \quad = \frac{-\alpha}{3-\alpha}\left(s(p_c)^{(3-\alpha)/(2-\alpha)} - 1\right), \tag{1}$$

o

$$s(p_c) = 1 - \frac{H^{(\alpha-1)}_{K_c-1}}{\zeta(\alpha-1)}, \text{ wi } \quad K_c \quad \text{ i yin } \quad H^{(\alpha-2)}_{K_c} - H^{(\alpha-1)}_{K_c} = \zeta(\alpha-1). \qquad (\ )$$

Fo n o n wo :

$$s(p_c) = \sum_{k=K_c+1}^{\infty} \frac{k \cdot p_k}{z}, \text{ wi } \quad K_c \quad \text{ i yin } \quad \sum_{k=0}^{K_c} k^2 \cdot p_k - \sum_{k=0}^{K_c} k \cdot p_k = z \tag{3}$$

T v l lo in Fi 3, w ll o x i n l v l o ol . W n in il o o ion o o i l v l o ol o n o n wo , o in y olvin ion 3. Solvin i ion iv v l o $K(p)$, xi l in n wo , in n ion o n z o n wo . By ni ion, $K(p)$ n only in v l . B in , in n o n wo , o no ll in ll o v l o n n, i i no lw y o i l o o in v l o $K(p)$ i y x ly ion. W v o n oin o in v l o z yi l l o , o v l o z o i in ny o ion o o i l ol . I i non l in in o o v x i n l v l o ol ollow v i y w o i l o .

W n now on l i ly on i n y o l i l - y. Fi , l o n o lin ov in n on l- n wo i , i i no i n o x l in oll o n wo :

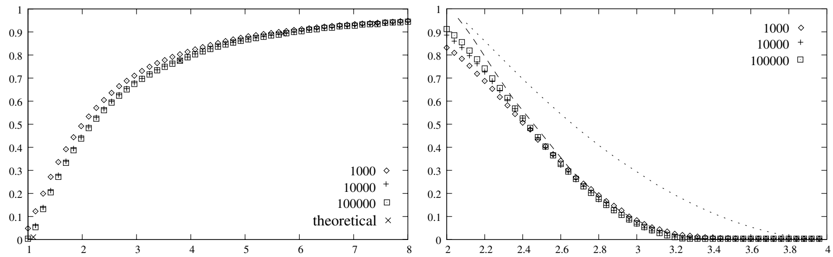


Fig. 3. Experimental values of the critical fraction $m(p_c)$ of links that must be removed in a classical node attack to disconnect random networks as a function of the mean degree (left), and scale-free networks as a function of the degree exponent (right). We have represented theoretical and experimental values. For scale-free networks, the values obtained from Equation 1 (dotted line) and from Equation 2 (dashed line) are plotted

i n o lin i n o ly ov , n n wo o no oll . How v , n o ov lin in l i l o n o n wo n o l - n wo v y i il , o v l o n w in in. T i o on l ion l - n wo i l ly n i iv o l i l : in o lin , y o n o n wo .

2 New Attack Strategies

In [1] i ion o n wo o l o ly v i n o on n i iv n:

$$\langle k^2\rangle - \ \langle k\rangle > 0 \iff p_1 < \sum_{k=3}^{\infty} k(k- \ )p_k$$

T y oin i o o o ion o no o 1 in n wo . T o , i ny y i in in in i o o ion o l i ly n wo . U in i , w o o wo n w - i (on in no n o in lin) w i iv o in i on l i n y o l i l .

2.1 Almost-Failures Attack

y i ly on i in n o ly ovin no o l . T i n o no o i n 1 n in n o no o 0 o 1. T ff o i i own in Fi .

No i i i ly iff n o no il , n y i i o i n . I lly i li iv ly iff n o il , in i i l y ol .

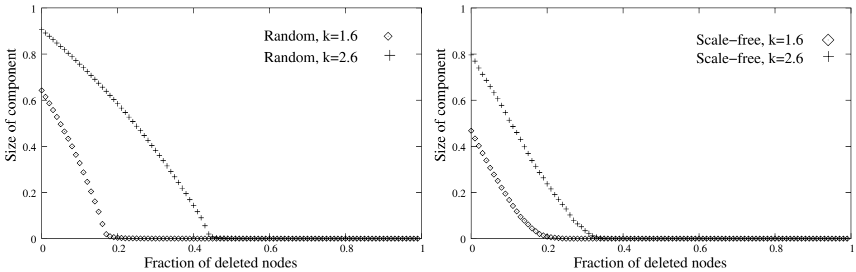


Fig. 4. The effect of the new node attack strategy on random networks (left) and scale-free networks (right)

W n ily ov i y ovi in n o n o i ol : w n ll no ini i lly i on v n ov , n n wo ly o no v i n o on n ny o , in ll no v o 1. T o i n o on n i oy w n ion $1 - p_1 - p_0$ o no n ov .

Fo l - n wo wi x on n α, i n i y i l o $1-1/\zeta(\alpha)$. Fo n o n wo wi n z, i i l o $1-e^{-z}(z+1)$. T lo o n i i own in Fi 5, o wi x i n l v l .

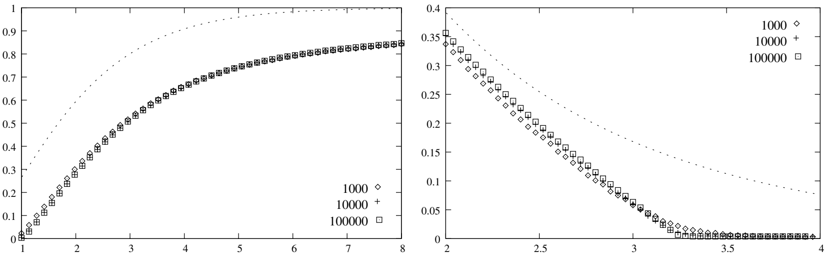


Fig. 5. The plots for the upper bound for the new node attack strategy (lines), and for experimental values of the threshold for networks of size 10^3, 10^4 and 10^5, for random networks (left) and scale-free networks (right). The lines represent the theoretical upper bound

No i v l o ol i l (on o ov l ion o no o oy n wo). i , o , i no o o in n i n y, o ow li iv iff n w n l i l y n no il on l-n wo li on , in n , no no o 1 ov : i no o i n 1 n o ly ov , n li iv vio i ov .

2.2 Efficient Link Attack

W v n in S ion 1 , l o l i l i l y ol
w n on i o lin oin o vi w, i i no i n in i . S ill
on in in o o ion o no o 1 oll
n wo , w now o o ollowin y on lin : w ov
n o lin w n no o l . T ff o i i
own in Fi 6.

A x , i y i l y ol m_c. A in, w n
ow i y ovi in n o n ollow .

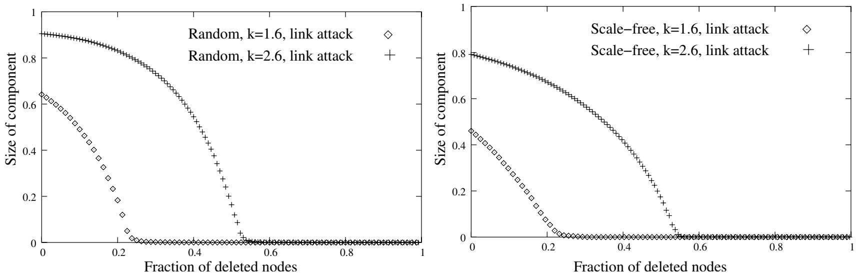


Fig. 6. The effect of the new link attack strategy on random networks (left) and scale-free networks (right)

W n ll lin w n no o l v n ov ,
n wo i o o in o i join (n l no i onn o
no o 1). Sin xi l o no in ni l - n wo
wi N no n v l $N^{\frac{1}{\alpha-1}}$ [ ], i o l onn
o on n (l) i lin wi o N w n v $\alpha >$.

An o n o m_c i o iv n y ion o lin w n
no o l . T i n i y i 1 in ion o lin in i n
o l on no o 1. T n o lin i iv n y n
o no o 1, in n o lin w n wo no o 1.

T i l n n o ollow . T Np_1 no o
1, o vin o ili y $Np_1/$ $|E|$ o in onn o no
no o 1[1] ($|E| = N\langle k\rangle/$ no n o lin in n wo).
T o n o . . . o 1 j n o no no o 1
i $N^2p_1^2/$ $|E| = Np_1^2/\langle k\rangle$ on v . Fin lly, n o lin w n wo
no i o $Np_1^2/$ $\langle k\rangle$.

F o i w v n o lin j n o l on no o
1 i : $Np_1 - Np_1^2/$ $\langle k\rangle$, n n o lin . . j n o ny no
o 1 i : $|E| - Np_1 + Np_1^2/$ $\langle k\rangle$. T ion o lin o i :

$$1 - \frac{p_1}{\langle k\rangle} + \frac{p_1^2}{\langle k\rangle^2}.$$

[1] This is accurate in the limit of large N.

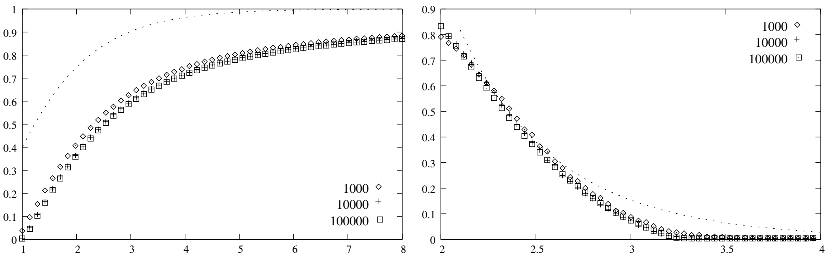


Fig. 7. Experimental values for the threshold for the new link attack strategy, for networks of size 10^3, 10^4 and 10^5, for random networks (left) and scale-free networks (right). The lines represent the upper bounds

Fo l- n wo , i n i y i l o:

$$1 - \frac{1}{\zeta(\alpha-1)} + \frac{1}{\zeta^2(\alpha-1)} = 1 - \frac{\zeta(\alpha-1)-1}{\zeta^2(\alpha-1)}.$$

Fo n o n wo , i i l o:

$$1 - \quad e^{-z} + e^{-2z}.$$

T i on n v l n i lly. T l o i v l ion i own in Fi , o wi x i n l v l .

I w o l o on o in in S ion 1, n w n o v o y i o i n n l i l on , vi w o lin oin o vi w. T i i no i in in in l i l y on y ov ny lin o no o 1, w i o no l in oyin n wo . y, on o o i , o on o lin w i lly i onn n wo .

3 Conclusion and Discussion

In i on i ion, w ovi il o i on o i o il n l i l on n o n l- n wo . i w o iv o i in i on l i n y o on l- o o n o n wo , n o o il .

To i v i , w inv i o n l i ion i n y o on l- n wo i o l n o lin y ov . W ow ovin n o lin n o l i , on i in i ion. How v , w n n o ov lin i on i , l- n wo no o il n n o on . Fin lly, w l i l i ion o n wo onn ivi y o i n wo n w i . T on i v y lo o i o il v li iv ly li l i l (i ol o l- n wo). T i n o ow n o ol o l i l i no o i i n y, o y o no ov no o 1. T on y w o o , on lin ov l, ow on n i n i o i n n l i l on , wi o ion o ov lin .

T l l o on l ion , i il n l i ll ly v iff n ly n l o n o o l- n o n wo on ly infl n i , on o l l in ivin on l ion o i . T n i ivi y o l- n wo o li on y v ny low- no . T i o n li on w n w oo no n o , w oo no wi i o ili y. Moov , l i l on l- n wo ov ny lin y on i ly no lly on i l o i i own.

T i wo y in ny i ion . Fi , y o v l ion o v io ol o l i ov . Li wi , i o ni i o l-wo l n wo i in n l no n oo n o l i . Mo ov , o o i o l-wo l o l x n wo , li lin o o l ion , o l n in o o n . F o o n l oin o vi w, i o il n on l n wo o in, li In n , Wo l Wi W n P - o-P y , l o iolo i l o o i l n wo , o l n . I i li ly n wo v o i n o i w i n v y ili n o il , n y n i iv o in i .

Acknowledgments. T i wo w ly n y oj . (http://www.laas.fr/~owe/METROSEC/) W w ly n Al n o V i n ni o l o n n i ion .

References

1. B. Bollobás. *Random Graphs.* Academic Press, 1985.
2. P. Erdös and A. Rényi. On random graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.
3. M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
4. A.-L. Barabási, Z. Deszo, E. Ravasz, S.H. Yook, and Z. Oltvai. Scale-free and hierarchical structures in complex networks. In *Sitges Proceedings on Complex Networks*, 2004.
5. S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks. *Adv. Phys. 51, 1079-1187*, 2002.
6. S.N. Dorogovtsev and J.F.F. Mendes. *Evolution of Networks: From Biological Nets tou the Internet and WWW.* Oxford University Press, 2000.
7. A.Z. Broder, S.R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J.L. Wiener. Graph structure in the web. *WWW9 / Computer Networks*, 33(1-6):309–320, 2000.
8. A.Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J.L. Wiener. Graph structure in the web. In *Proceedings of*

the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, pages 309–320. North-Holland Publishing Co., 2000.

9. M.E.J. Newman. Random graphs as models of networks. In Stefan Bornholdt and Heinz Georg Schuster, editors, *Hankbook of Graphs and Networks: From the Genome to the Internet.* Wiley-vch, 2003.
10. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.
11. L. Adamic and B. Huberman. Power-law distribution of the world wide web. *Science*, 287, 2000.
12. M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal special issue on peer-to-peer networking*, 6(1), 2002.
13. R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance in complex networks. *Nature*, 406:378–382, 2000.
14. D.S. Callaway, M.E.J. Newman, S.H. Strogatz, and D.J. Watts. Network robustness and fragility: Percolation on random graphs. *Phys. Rev. Lett.*, 85:5468–5471, 2000.
15. R. Cohen, K. Erez, D. ben Avraham, and S. Havlin. Breakdown of the internet under intentional attack. *Phys. Rev. Lett.*, 86:3682–3685, 2001.
16. S.-T. Park, A. Khrabrov, D.M. Pennock, S. Lawrence, C. Lee Giles, and L.H. Ungar. Static and dynamic analysis of the internet's susceptibility to faults and attacks. In *IEEE Infocom 2003*, San Francisco, CA, April 1–3 2003.
17. A. Broido and K. Claffy. Topological resilience in ip and as graphs. 2002. `http://www.caida.org/analysis/topology/resilience/`
18. V. Latora and M. Marchiori. Efficient behavior of small-world networks. *Phys. Rev. Lett.*, 87, 2001.
19. P. Crucitti, V. Latora, M. Marchiori, and A. Rapisarda. Efficiency or scale-free networks: error and attack tolerance. *Physica A*, 320:622–642, 2003.
20. R. Pastor-Satorras and A. Vespignani. *Evolution and Structure of the Internet: A Statistical Physics Approach.* Cambridge University Press, 2003. To appear.
21. M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6:161, 1995.
22. R. Cohen, K. Erez, D. ben Avraham, and S. Havlin. Resilience of the internet to random breakdown. *Phys. Rev. Lett.*, 85:4626, 2000.

Firewall Queries

Al x X. Li [1], Mo . o [1], H i o H. M [2], n Ann HH. N [2]

[1] Department of Computer Sciences, The University of Texas at Austin,
Austin, Texas 78712-0233, U.S.A.
{alex, gouda}@cs.utexas.edu

[2] Department of Computer Science, Texas State University,
San Marcos, Texas 78666-4616, U.S.A.
{hm1034, angu}@txstate.edu

Abstract. Firewalls are crucial elements in network security, and have been widely deployed in most businesses and institutions for securing private networks. The function of a firewall is to examine each incoming and outgoing packet and decide whether to accept or to discard the packet based on a sequence of rules. Because a firewall may have a large number of rules and the rules often conflict, understanding and analyzing the function of a firewall have been known to be notoriously difficult. An effective way to assist humans in understanding and analyzing the function of a firewall is by issuing firewall queries. An example of a firewall query is "Which computers in the private network can receive packets from a known malicious host in the outside Internet?". Two problems need to be solved in order to make firewall queries practically useful: how to describe a firewall query and how to process a firewall query. In this paper, we first introduce a simple and effective SQL-like query language, called the Structured Firewall Query Language (SFQL), for describing firewall queries. Second, we present a theorem, called the Firewall Query Theorem, as a foundation for developing firewall query processing algorithms. Third, we present an efficient firewall query processing algorithm, which uses firewall decision trees as its core data structure. Experimental results show that our firewall query processing algorithm is very efficient: it takes less than 10 milliseconds to process a query over a firewall that has up to 10,000 rules.

Keywords: Network Security, Firewall Queries, Firewalls.

1 Introduction

S vin lin o n in li io n n o i
, w ll i l l n in in iv n wo o o
in , in i ion , n v n o n wo . A w ll i l oin
o n y w n iv n wo n o i In n o ll in o in
n o oin v o o i . A n vi w l
wi ni n o l ; x l o l o / in ion
IP , o / in ion o n , n o o ol y . A w ll

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 197–212, 2005.

in o in n o oin o i ion o in o i on ion. A w ll on ion n w i l i i n w i il- l i i y n o l . l in w ll on ion i o o

$$\langle \quad \rangle \rightarrow \langle decision\rangle$$

T ⟨ ⟩ in l i ool n x ion ov o l n y i l n wo in on w i iv . Fo o vi y, w l on in i n i ion o n wo in on w i iv . T ⟨*decision*⟩ o l n , o , o o in ion o i ion wi o o ion lo in o ion. Fo i li i y, w ⟨*decision*⟩ in l i i o .

A l i n only i ( ) i i o l . T i o l l in w ll i lly olo y o n v y l on in l in w ll. T l in w ll o n onfli . Two l in w ll iff y v iff n i ion n i l on n o l . D o onfli on l , y o n on l in w ll, n l y v iff n i ion . To olv onfli on l , o in o in o o oin , w ll i o i ion o (i. ., i io i y) l .

T n ion (i. ., vio) o w ll i i in i on ion, w i on i o n o l . T on ion o w ll i o i o n o on n in i vin i y n n ion li y o w ll [ ]. How v , o w ll on In n oo ly on , wi n y o n wo n vi li Bl [6] n S - i [ ], w i o l ily lo y w ll- on w ll [6]. I n o v o w ll i y y on ion o [5]. An o in w ll on ion n o ill i i - i n i in l i i , o o l i i i n i in ill i i . T i will i llow n o i o o - i In n o iv n wo , o i l o l i i o ni ion w n iv n wo n o i In n . N i i i l . l ly, w ll on ion o l w ll n oo n n ly o in loy .

How v , o l n o l in w ll n l n o onfli on l , n n in n n ly in n ion o w ll v n nown o no o io ly i l [1]. T i li ion o ny l in w ll nno n oo wi o x inin ll l li ov l . T o o on i o i l i in n n - in n n ly in w ll . Fo x l , o o w ll o n on i o l w i n y iff n ini o iff n i n o i - n on . I i i l o n w w ll ini o o n n i li ion o l i no w i n y l .

An ff iv w y o i n in n n in n n ly in w ll i y i in w ll i . Fi w ll i ion on nin n ion o w ll. x l o w ll i “W i o in o i In n nno n il o il v in iv n wo ?” n “W i o in iv n wo n iv B TP[1] o o - i In n ?”. Fi in o n w o w ll i i o n o l o w ll ini o o n n n n ly n ion o w ll. Fo x l , in i ion o w ll i ll o in o i In n , x nown li io o , l o n il o il v in iv n wo , w ll ini o n w w ll i i i n y i in w ll y “W i o in o i In n nno n il o il v in iv n wo ?”. I n w o i y on in x ly nown li io o , n w ll ini o i w ll o i y i i n . wi w ll ini o now w ll il o i y i i n , n n o on - w ll. A no x l , o i ion o w ll i ny B TP o o i In n i o lo o n - in iv n wo . To w w ll i i i n , w ll ini o n i w ll y “W i o in iv n wo n iv B TP o o i In n ?”. I n w o i y i n y , n w ll ini o i w ll o i y i i n . wi w ll ini o now w ll il o i y i i n , n n o on w ll.

Fi w ll i l o l in v i y o o n io , - w ll in n n n w ll in . Fo w ll ini o , in w w ll i in on i ion i o ily in n n - ivi y. Fo x l , i ini o o in iv n wo i n , w ll ini o n i i o w i o o in iv n wo l o v ln l o y o . In o o i nin w ll, i n n i o w ll i o i n o y in w n w o i on i n wi w ll i ion.

To w ll i i lly l, wo o l n o olv : ow o i w ll y n ow o o w ll y. T on o l i ni lly i l . ll l in w ll n i iv o l o n l o n onfli . T n iv ol ion i o n v y i y y n i ion o . l ly,

[1] The Bootp protocol is used by workstations and other devices to obtain IP addresses and other information about the network configuration of a private network. Since there is no need to offer the service outside a private network, and it may offer useful information to hackers, usually Bootp packets are blocked from entering a private network.

i ol ion i in i l . Fo x l , o o y “W i o in o i In n nno n ny o iv n wo ?”, i n iv ol ion n o n ^{88} o i l n i ion o w ll o , w i i in i l .

In i , w n ol ion o o o l . Fi , w in o i l n ff iv SQL-li y l n , ll S Fi w ll Q y L n (SFQL), o i in w ll i . T i l n i o o “. ”. S on , w n o , ll Fi w ll Q y T o , o n ion o v lo in w ll y o in l o i . T i , w n n i n y o in lo i w ll i ion i o . Fo iv n w ll o n o l , w on n iv l n w ll i ion y on ion l o i . T n w ll i ion i o o i y o in l o i o n w in w ll y. x i n l l ow o w ll y o in l o i i v y i n : i l n 10 illi on o o y ov w ll o 10,000 l . l ly, o w ll y o in l o i i no in in in wi w ll ini o .

No w ll w on i in i i w ll , no l w ll in w i n ion o w ll n yn i lly -y. Al o no i o w ll in n i ily o ini o o w ll o . Fo w ll o iv n wo , n i no l in iv n wo no o i o iv n wo l o y w ll. Sin o o i i -w ll on ion , in o i , w “ w ll” o n “ w ll on ion” i no o wi i .

2 Related Work

T i li l wo n on on w ll i . In [1, 5], w ll n ly i y o i w ll i w n . In [1, 5], w ll y i i y i l (o o , o in ion , o vi), w vi i l (o o ol y , in ion o n). T n i o y “w i IP in o o n n w i vi in o vi o w i IP in o in ion ?”. W o yon [1, 5] in ollowin wo jo .

1. No l o i o o in w ll y ov n o l w n in [1] o [5]. on n ly, ow n l l w ll y n o in n nown, w il i n y o w ll y o in l o i i i l in o o in wi n . In on , w n n i n l o i o o in w ll y ov n o l . w ll y l o i l n 10 illi on o o y ov w ll o 10,000 l .

. T y l n i in [1] n [5] i oo i : i i only -
li l o IP n i only on n o l o o ,
in ion , o o ol y n in ion o n . T i
x iv ow o y l n in [1, 5] li i . Fo x l ,
v n only on i in IP , i nno x w ll y on n-
in o o n o li ion l . In on , o S
Fi w ll Q y L n i l o x in w ll i wi i-
y l .

In [1], o - o "w i " ion i il o w ll i
w i . How v , no l o i w n o o in o o
"w i " ion .

In [9], x y w o o o n ly w ll l . l ly, il in
n x y j o n ly in w ll i ov w o n i i l.

D in o n i l w ll on ion o y onfli ion w i -
in [3, ,1 ,]. Si il o onfli ion, ix y o o- ll " no -
li " w n in [1]. x inin onfli o no ly i l l in -
in o n i l w ll on ion o ; ow v , n o onfli o
no li in w ll i y i lly l , n n l in o on-
fli o no ly i n li l nin o l n on
n o o l in w ll, w i y in o .

So w ll i n o v n o o in [, 16, 0, 13]. T
wo i in w ll l , w il w i n ly in w ll l .

Fi w ll v ln ili i i n l i in [19, 11]. How v ,
o o [19, 11] v ln ili i o l in o w n
o in w o w ll, no on ion o w ll.

T o ool n ly v il l o n wo v ln ili y in ,
S n [10, 1] n N [3]. T v ln ili y in ool n
iv n wo on n li ly nown , n
i n i ion o w ll. Al o ool n o i ly
o llow ill i i o iv n wo , i nno n
o i l l i i o ni ion w n iv n wo n
o i In n .

3 Structured Firewall Query Language

3.1 Firewalls

In i ion, w n l yn x o w ll y l n n
ow ow o i l n o i w ll i .

W n ⸝ . . - ov l $F_1, \cdots, F_d$ d- l $(p_1, \cdots, p_d)$
w p_i i in o in $D(F_i)$ o l F_i, n $D(F_i)$ i n in v l
o nonn iv in . Fo x l , o in o o in n IP
i $[0, {}^{32})$. Fo vi y o n ion, w ll
ov d l $F_1, \cdots, F_d$, i no o wi i . W Σ o no
o ll . I ollow Σ i ni n $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_n)|$.

iv n w ll f, p in Σ i y f o i ion, no $f.p$, in { , , }. Two w ll f n f' iv l n , no $f \equiv f'$, iff o ny p in Σ, $f.p = f'.p$ ol . T i iv l n l ion i y i , l- fl iv , n n i iv .

A w ll on i o n o l . l i o ollowin o :

$$(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle \quad \rangle$$

w S_i i non y o $D(F_i)$, n $\langle \quad \rangle$ i i , o . I $S_i = D(F_i)$, w n l $(F_i \in S_i)$ y $(F_i \in \ )$, o ov onj n $(F_i \in D(F_i))$ l o . So xi in w ll o , Lin x' i in, i S_i n in x o 19 .16 .0.0/16, w 16 n x i 16 i o 19 .16 .0.0 in in y o . In i , w oo o n S_i non y o nonn iv in o wo on . Fi , ny o nonn iv in n o i lly onv o o x ([15]). S on , n ion o onv ni n in i l ni l ion .

A $(p_1, \cdots, p_d)$ l $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle \quad \rangle$ iff on i ion $(p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d)$ ol . Sin y o n on l in w ll, i o i ion o l . T i o l l in w ll i lly olo y o n v y l on in l in w ll.

H w iv n x l o i l w ll. In i x l , w only wo l : S (o) n D (in ion), n o l v o in [1, 10]. T i w ll on i o n o l in Fi 1. L f_1 n o i w ll.

r_1 : $S \in [4, 7] \ \wedge \ D \in [6, 8] \ \rightarrow$ *accept*
r_2 : $S \in [3, 8] \ \wedge \ D \in [2, 9] \ \rightarrow$ *discard*
r_3 : $S \in [1, 10] \ \wedge \ D \in [1, 10] \rightarrow$ *accept*

Fig. 1. Firewall f_1

3.2 Query Language

A , no Q, in o S Fi w ll Q y L n (SFQL) i o ollowin o :

select F_i
from f
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge$ (**decision** $= \langle \quad \rangle$)

w F_i i on o l $F_1, \cdots, F_d$, f i w ll, S_j i non y o o in $D(F_j)$ o l F_j, n $\langle \quad \rangle$ i i , o .

T l o y Q, no *Q.result*, i ollowin :

$$\{p_i|(p_1,\cdots,p_d) \quad , \quad \Sigma, \quad (p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d) \wedge (f.(p_1,\cdots,p_d) = \langle \quad \rangle)\}$$

ll Σ no o ll , n $f.(p_1,\cdots,p_d)$ no iion o w i w ll f $(p_1,\cdots,p_d)$.

W n ov y n in ll $(p_1,\cdots,p_d)$ in Σ ollowin on i ion

$$(p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d) \wedge (f((p_1,\cdots,p_d)) = \langle \quad \rangle)$$

ol , n oj in ll o l F_i.

Fo x l , ion o w ll in Fi 1, “W i o w o in [,] n n o in w o i 6?”, n o l ollowin y in SFQL:

select S
from f_1
where $(S \in \{[\ ,\ ]\}) \wedge (D \in \{6\}) \wedge ($**decision** $= \quad)$

T l o i y i { ,5,6, }.

A no x l , ion o w ll in Fi 1, “W i o nno n o o w o i 6?”, n o l ollowin y in SFQL:

select S
from f_1
where $(S \in \{ \quad \}) \wedge (D \in \{6\}) \wedge ($**decision** $= \quad)$

T l o i y i {3, }.

N x w iv o x l on ow o SFQL o i w ll i .

4 Firewall Query Examples

In i ion, w i o x l w ll i in SFQL. L f n o w ll i on w y o in Fi . T i w y o wo in : in 0, w i onn w y o o o i In n , n in 1, w i onn w y o o

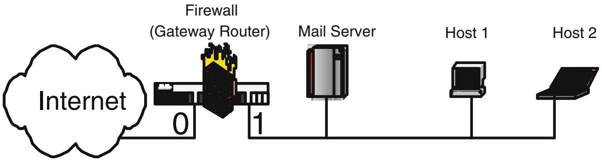


Fig. 2. Firewall f

in i lo l n wo . In x l , w ollowin v l : I (In), S (So IP), D (D in ion IP), N (D in ion Po), P (P o o ol Ty).

Q ion 1:
W i o in iv n wo o y w ll f n iv B TP[2] o o i In n ?
Q y Q_1:
select D
from f
where $(I \in \{0\}) \wedge (S \in \{\ \}) \wedge (D \in \{\ \}) \wedge (N \in \{6\ ,6\ \})$
$\wedge(P \in \{\ \}) \wedge ($**decision** $=\ )$
An w o ion 1 i $Q_1.result$.

Q ion :
W i o on il v o y w ll f o n?
Q y Q_2:
select N
from f
where $(I \in \{0,1\}) \wedge (S \in \{\ \}) \wedge (D \in \{\ \} \wedge (N \in \{\ \})$
$\wedge(P \in \{\ \}) \wedge ($**decision** $=\ )$
An w o ion i $Q_2.result$.

Q ion 3:
W i o in o i In n nno n SMTP[3] o il v o y w ll f?
Q y Q_3:
select S
from f
where $(I \in \{0\}) \wedge (S \in \{\ \}) \wedge (D \in \{\ \}) \wedge (N \in \{\ \})$
$\wedge(P \in \{\ \}) \wedge ($**decision** $=\ )$
An w o ion 3 i $Q_3.result$.

Q ion :
W i o in o i In n nno n ny o iv n wo o y w ll f?
Q y Q_4:
select S
from f
where $(I \in \{0\}) \wedge (S \in \{\ \}) \wedge (D \in \{\ \}) \wedge (N \in \{\ \}) \wedge (P \in \{\ \})$
$\wedge($**decision** $=\ )$
An w o ion i $T - Q_4.result$, w T i o ll IP o i o iv n wo

[2] Bootp packets are UDP packets and use port number 67 or 68.

Q ion 5:
W i o in o i In n n n SMTP o o o 1 n o in iv n wo o y w ll f?
Q y Q_{5a}:
select S
from f
where $(I \in \{0\}) \wedge (S \in \{ \}) \wedge (D \in \{ \}) \wedge (N \in \{ \})$
$\wedge(P \in \{ \}) \wedge ($**decision** $=)$
Q y Q_{5b}:
select S
from f
where $(I \in \{0\}) \wedge (S \in \{ \}) \wedge (D \in \{ \}) \wedge (N \in \{ \})$
$\wedge(P \in \{ \}) \wedge ($**decision** $=)$
An w o ion 5 i $Q_{5a}.result \cap Q_{5b}.result$.

5 Firewall Query Processing

In i ion, w i ow o o w ll y o on i n w ll . on i n w ll n in on i n w ll n ollow :

Definition 1 (Consistent Firewalls). A w ll i ll on i n w ll iff ny wo l in w ll o no onfli .

Definition 2 (Inconsistent Firewalls). A w ll i ll n in on i n w ll iff l wo l in w ll onfli .

ll wo l in w ll onfli iff y v iff n i ion n i l on n o l . Fo x l , wo l in w ll in Fi 1, n ly r_1 n r_2, onfli . No o ny wo l in on i n w ll, i y ov l , i. ., i l on n o l , y v i ion. So, iv n n on i n w ll, ll l in w ll v i ion. Fi 1 ow n x l o n in on i n w ll, n Fi 3 ow n x l o on i n w ll. In wo w ll x l , w only wo l : S (o) n D (in ion), n o l v o in $[1, 10]$.

in in on i n w ll i wo ol . Fi , in on i n w ll n onv o n iv l n on i n w ll, i in S ion 6. S on , own in ollowin o , i i i o o i o on i n w ll n o in on i n w ll .

Theorem 1 (Firewall Query Theorem). L Q y o ollowin o :

[3] SMTP stands for Simple Mail Transfer Protocol. SMTP packets are TCP packets and use port number 25.

$r'_1: \ S \in [4,7] \quad \wedge D \in [6,8] \quad \rightarrow a$
$r'_2: \ S \in [4,7] \quad \wedge D \in [2,5] \cup [9,9] \quad \rightarrow d$
$r'_3: \ S \in [4,7] \quad \wedge D \in [1,1] \cup [10,10] \rightarrow a$
$r'_4: \ S \in [3,3] \cup [8,8] \quad \wedge D \in [2,9] \quad \rightarrow d$
$r'_5: \ S \in [3,3] \cup [8,8] \quad \wedge D \in [1,1] \cup [10,10] \rightarrow a$
$r'_6: \ S \in [1,2] \cup [9,10] \wedge D \in [1,10] \quad \rightarrow a$

Fig. 3. Consistent firewall f_2

select F_i
from f
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge$ (**decision** $= \langle \quad \rangle)$

I f i on i n w ll on i o n l $r_1, \cdots, r_n,$ n w v

$$Q.result = \bigcup_{j=1}^{n} Q.r_j$$

w l r_j i o o

$$(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle \quad ' \rangle$$

n n i y o $Q.r_j$ i n ollow :

$$Q.r_j = \begin{cases} S_i \cap S'_i & \text{i } (S_1 \cap S'_1 \neq \emptyset) \wedge \cdots \wedge (S_d \cap S'_d \neq \emptyset) \wedge (\langle \quad \rangle = \langle \quad ' \rangle), \\ \emptyset & \text{o wi} \end{cases}$$

□

T Fi w ll Q y T o i li i l y o in l o i :
iv n on i n w ll f on i o n l $r_1, \ \cdots, \ r_n$ n y Q,

```
Rule – based Firewall Query Processing Algorithm
Input  : (1) A consistent firewall $f$ that consists of $n$ rules: $r_1, \cdots, r_n$,
         (2) A query $Q$:
             select $F_i$
             from $f$
             where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge$ (decision $= \langle dec \rangle$)
Output: Result of query $Q$
Steps:
1. $Q.result := \emptyset$;
2. for $j := 1$ to $n$ do /*Let $r_j = (F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$*/
     if $(S_1 \cap S'_1 \neq \emptyset) \wedge \cdots \wedge (S_d \cap S'_d \neq \emptyset) \wedge (\langle dec \rangle = \langle dec' \rangle)$
     then $Q.result := Q.result \cup (S_i \cap S'_i)$;
3. return $Q.result$;
```

Fig. 4. Rule-based Firewall Query Processing Algorithm

o $Q.r_j$ o j, n $\bigcup_{j=1}^{n} Q.r_j$ i l o y Q. W ll i l o i . Fi ow o o o i l o i .

6 FDT-Based Firewall Query Processing Algorithm

v l i l l in on i n w ll y x. Fo x l , in on i n w ll in Fi 3, l , n ly r'_1, r'_2, r'_3, x $S \in [\ ,\ ]$. T , i w ly ov y o in l o i in Fi o n w y, o in n , w o "w l " on in onj n $S \in \{3\}$, ov w ll in Fi 3, n l o i will i l l ion o $\{3\} \cap [\ ,\ ]$. l ly, l l ion no i l o i n y o .

In i ion, w n w ll y o in o no l l ion n n li o o on i n n in on i n w ll . T i o on i o wo . Fi , onv w ll (w on i n o in on i n) o n iv l n w ll i ion (o o FDT). S on , i FDT o o o in i . W ll l o i n FDT o o i . Fi w ll i ion n ollow . No w ll i ion i l y o w ll i ion i , w i in o in [13] l no ion o i yin w ll .

Definition 3 (Firewall Decision Tree). A Fi w ll D i ion T t ov l $F_1, \cdots, F_d$ i i ollowin o o i :

1. no v in t l l, no $F(v)$,
$$F(v) \in \begin{cases} \{F_1, \cdots, F_d\} & \text{i } v \text{ i non in l,} \\ \{\ \ \} & \text{i } v \text{ i in l.} \end{cases}$$
. e in t l l, no $I(e)$, i e i n o oin o no v, n $I(e)$ i non y o $D(F(v))$.
3. A i in t o oo o in l no i ll o t. i ion on in d non in l no , n i- no i l ll F_i o i $1 \le i \le d$.

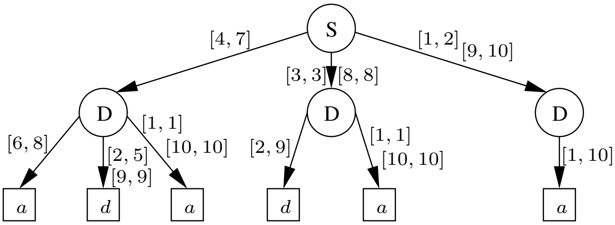


Fig. 5. Firewall Decision Tree t_3

. T o ll o oin o no v in t, no $E(v)$, i ollowin wo on i ion :
( ) : $I(e) \cap I(e') = \emptyset$ o ny wo i in e n e' in $E(v)$,
( ) : $\bigcup_{e \in E(v)} I(e) = D(F(v))$ □

Fi 5 ow n x l o n FDT n t_3. In i x l , w only wo l : S (o) n D (in ion), n o l v o in $[1, 10]$. In o i , in l in i x l , w "a" o n o n "d" o n o .

A i ion in n FDT t i n y $(v_1 e_1 \cdots v_k e_k v_{k+1})$ w v_1 i oo , v_{k+1} i in l no , n e_i i i o no v_i o no v_{i+1}. A i ion $(v_1 e_1 \cdots v_k e_k v_{k+1})$ in n FDT n ollowin l :

$$F_1 \in S_1 \wedge \cdots \wedge F_n \in S_n \;\rightarrow\; F(v_{k+1})$$

w

$$S_i = \begin{cases} I(e_j) & \text{i} \quad \text{i ion} \quad \text{no} \; v_j \quad \text{i l ll wi l} \; F_i, \\ D(F_i) & \text{i} \quad \text{i ion} \quad \text{no no} \quad \text{i l ll wi l} \; F_i. \end{cases}$$

Fo n FDT t, w $\Gamma(t)$ o no o ll l n y ll i ion o t. Fo ny p, i on n only on l in Γ_t p o on i n y n o l n o i ; o , t p o i ion o only l p in Γ_t. on i in FDT t_3 in Fi 5, Fi 3 ow ll ix l in Γ_{t_3}.

iv n n FDT t, ny n o l on i o ll l in Γ_t i iv l n o t. T o o l in w ll i i i l l in Γ_t non-ov l in . iv n n o l , n iv l n FDT n on in on ion l o i i in [0]. T o , n in on i n w ll n onv o n iv l n on i n w ll in ollowin wo : , on n iv l n FDT o o i in l in on i n w ll; on , n on l o i ion o FDT. T n ny n on i o ll l n y i ion o FDT i l in iv l n on i n w ll.

T o o o FDT- w ll y o in l o i i own in Fi 6. H w $e.t$ o no ( ) no e oin o, n w $t.$ o no oo o FDT t.

T ov FDT- w ll y o in l o i wo in , n FDT t n n SFQL y Q. T l o i y v in FDT o i oo . L F_j l l o oo . Fo o oin e o oo , w o $I(e) \cap S_j$. I $I(e) \cap S_j = \emptyset$, w i e n o no v e oin o. I $I(e) \cap S_j \neq \emptyset$, n w on in o v e oin o in i il ion. W n v in l no i n o n , w o l l o in l no n $\langle \;\; \rangle$. I y , in l n y i ion on inin in l no i $(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle \;\; ' \rangle$, n w $S_i \cap S'_i$ o $Q.result$.

```
FDT – based Firewall Query Processing Algorithm
Input : (1)An FDT $t$,
        (2)A query $Q$:  select  $F_i$
                         from    $t$
                         where   $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$
Output : Result of query $Q$
Steps:
1. $Q.result := \emptyset$;
2. CHECK( $t.root$, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle$ )
3. return $Q.result$;


CHECK( $v$, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$ )
1. if ( $v$ is a terminal node ) and ( $F(v) = \langle dec \rangle$ ) then
   (1) Let $(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$ be the rule
       defined by the decision path containing node $v$;
   (2) $Q.result := Q.result \cup (S_i \cap S'_i)$;
2. if ( $v$ is a nonterminal node ) then /*Let $F_j$ be the label of $v$*/
      for each edge $e$ in $E(v)$ do
        if $I(e) \cap S_j \neq \emptyset$ then
          CHECK( $e.t$, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$ )
```

Fig. 6. FDT-based Firewall Query Processing Algorithm

7 Experimental Results

So w v n wo w ll y o in l o i , l- l o i in S ion 5 n FDT- l o i in S ion 6. In i - ion, w v l i n y o o l o i . In n o li ly v il l w ll , w yn i w ll o in o i i o l-li l i i in [,1]. No w ll i l o l i . l ollowin v l : in , o IP - , in ion IP , in ion o n n o o ol y . T o i l n in SUN J v JDK 1. . T x i n w i o on S nBl 000 in nnin Sol i 9 wi 1 PU n 1 B o o y.

Fi ow v x ion i o o l o i v o l n o l in o i in l (y in on i n) w ll . T o i on l xi in i o l n o l in o i in l w ll , n v i l xi in i v x ion i (in illi on) o o in w ll y. No in Fi , x ion i o FDT- w ll y o in l o i o no in l FDT on ion i onv ion o w ll o n iv l n FDT i o only on o w ll, no o y. Si il ly, x ion i o l- w ll y o in l o i o no in l i o onv in n in on i n w ll o n iv l n on i n w ll i onv ion i o only on o w ll, no o y.

F o Fi , w n FDT- w ll y o in
l o i i o i n n l - w ll y o in
l o i . Fo x l , o o in y ov n in on i n w ll
10,000 l , FDT- y o in l o i o 10 il-
li on , w il l - y o in l o i o 100 il-
li on . T x i n l l in Fi on o n ly i
FDT- y o in l o i v x ion i y in -
l l ion .

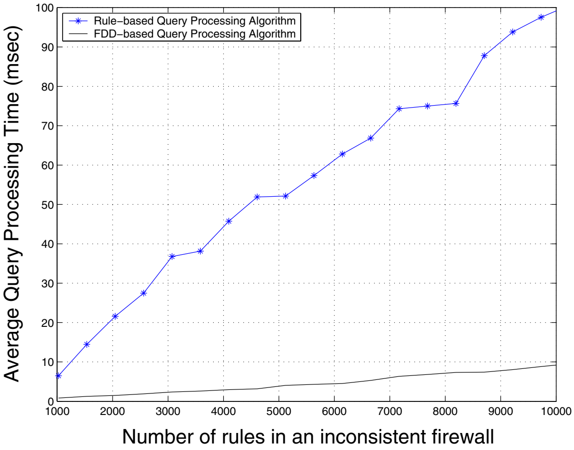


Fig. 7. Query Processing Time vs. Number of rules

8 Concluding Remarks

on i ion in i - ol . Fi , w in o i l n
ff iv SQL-li y l n , S Fi w ll Q y L n , o
i in w ll i . S on , w n o , Fi w ll Q y
T o , o n ion o v lo in w ll y o in l o i .
T i , w n n i n l o i w ll i ion i
o o o in w ll i . iv n w ll o n
o l , w on n iv l n w ll i ion . T n w ll
i ion i o o i y o in l o i
o n w w ll y. x i n l l ow i y
o in l o i i v y i n .

To o n ion i l , w v i o w w -
own v ion o w ll y l n w “ l ” l in y
only on l . In , “ l ” l in y n x n o
v o n on l . T l in i , . ., Fi w ll Q y
T o n wo w ll y o in l o i , n ll x n
o in ly o o o x n “ l ” l .

References

1. E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.
2. F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to cams? In *Proc. of IEEE INFOCOM*, 2003.
3. F. Baboescu and G. Varghese. Fast and scalable conflict detection for packet classifiers. In *Proc. of the 10th IEEE International Conference on Network Protocols*, 2002.
4. Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *Technical Report EES2003-1, Dept. of Electrical Engineering Systems, Tel Aviv University*, 2003.
5. CERT. Test the firewall system. http://www.cert.org/security-improvement/practices/p060.html.
6. CERT Coordination Center. http://www.cert.org/advisories/ca-2003-20.html.
7. D. Moore et al. http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.
8. D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Symp. on Discrete Algorithms*, pages 827–835, 2001.
9. P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *Proc. of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, pages 100–107, 2001.
10. D. Farmer and W. Venema. Improving the security of your site by breaking into it. *http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html*, 1993.
11. M. Frantzen, F. Kerschbaum, E. Schultz, and S. Fahmy. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Computers and Security*, 20(3):263–270, 2001.
12. M. Freiss. *Protecting Networks with SATAN.* O'Reilly & Associates, Inc., 1998.
13. M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proc. of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 320–327.
14. P. Gupta. *Algorithms for Routing Lookups and Packet Classification.* PhD thesis, Stanford University, 2000.
15. P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
16. J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. of IEEE Symp. on Security and Privacy*, pages 120–129, 1997.
17. A. Hari, S. Suri, and G. M. Parulkar. Detecting and resolving packet filter conflicts. In *Proc. of IEEE INFOCOM*, pages 1203–1212, 2000.
18. S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'00)*, pages 576–585, 2000.
19. S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214–232, 2003.
20. A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 595–604, June 2004.
21. A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. of IEEE Symp. on Security and Privacy*, pages 177–187, 2000.
22. J. D. Moffett and M. S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 4(1):1–22, 1994.

23. Nessus. http://www.nessus.org/. March 2004.
24. A. D. Rubin, D. Geer, and M. J. Ranum. *Web Security Sourcebook*. Wiley Computer Publishing, 1th edition, 1997.
25. A. Wool. Architecting the lumeta firewall analyzer. In *Proc. of the 10th USENIX Security Symposium*, pages 85–97, August 2001.
26. A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

Self-tuning Reactive Distributed Trees for Counting and Balancing

Phuong Hoai Ha, Marina Papatriantafilou, and Philippas Tsigas

Department of Comp. Science, Chalmers University of Technology,
SE-412 96 Göteborg, Sweden
{phuong, ptrianta, tsigas}@cs.chalmers.se

Abstract. The main contribution of this paper is that it shows that it *is* possible to have reactive distributed trees for counting and balancing with no need for the user to fix manually any parameters. We present a data structure that in an on-line manner balances the trade-off between the tree traversal latency and the latency due to contention at the tree nodes. Moreover, the fact that our method can expand or shrink a subtree several levels in any adjustment step, has a positive effect in the efficiency: this feature helps the self-tuning reactive tree minimize the adjustment time, which affects not only the execution time of the process adjusting the size of the tree but also the latency of all other processes traversing the tree at the same time with no extra memory requirements. Our experimental study compared the new trees with the reactive diffracting ones on the SGI Origin2000, a well-known commercial ccNUMA multiprocessor. This study showed that the self-tuning reactive trees i) select the same tree depth as the reactive diffracting trees do; ii) perform better and iii) react faster.

1 Introduction

Distributed data structures suitable for synchronization that perform efficiently across a wide range of contention conditions are hard to design. Typically, “small”, “centralized” such data structures fit better low contention levels, while “bigger”, “distributed” such data structures can help in distributing concurrent processor accesses to memory banks and in alleviating memory contention.

Diffracting trees [1] are distributed data structures. Their most significant advantage is the ability to distribute a set of concurrent process accesses to many small groups locally accessing shared data, in a coordinated manner. Each process(or) accessing the tree can be considered as leading a *token* that follows a path from the root to the leaves. Each node is a computing element receiving tokens from its single input (coming from its parent node) and sending out tokens to its outputs; it is called *balancer* and acts as a *toggle mechanism* which, given a stream of input tokens, alternately forwards them to its outputs, from left to right (sending them to the left and right child nodes, respectively). The result is an even distribution of tokens at the leaf nodes. Diffracting trees have been introduced for *counting-problems*, and hence the leaf nodes are counters, assigning numbers to each token that exits from them. Moreover, the number of tokens that are output at the leaves, satisfy the *step property*, which states that: when there are no tokens present inside the tree and if out_i denotes the number of tokens that have been output at leaf i, $0 \leq out_i - out_j \leq 1$ for any pair i and j of leaf-nodes such that $i < j$ (i.e.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 213–228, 2005.

if one makes a drawing of the tokens that have exited from each counter as a stack of boxes, the combined outcome will have the shape of a single step).

The fixed-size diffracting tree is optimal only for a small range of contention levels. To solve this problem, Della-Libera and Shavit proposed the *reactive diffracting trees*, where each node can shrink (to a counter) or grow (to a subtree with counters as leaves) according to the current load, in order to attain optimal performance [2]. The algorithm in [2] uses a set of parameters to make its decisions, namely folding/unfolding thresholds and the time-intervals for consecutive reaction checks. The parameter values depend on the multiprocessor system in use, the applications using the data structure and, in a multiprogramming environment, on the system utilization by the other programs that run concurrently. The programmer has to fix these parameters manually, using experimentation and information that is commonly not easily available (future load characteristics). A second characteristic of this scheme is that the reactive part is allowed to shrink or expand the tree only one level at a time, making the cost of a multi-adjustment phase on a reactive tree become high.

In this work we show that reactiveness and these two characteristics are not tied together: in particular, we present a tree-type distributed data structure that has the same semantics as the reactive trees that can expand or shrink many levels at a time, without need for manual tuning. To circumvent the need for manually setting parameters, we have analyzed the problem of balancing the trade-off between the two key measures, namely the contention level and the depth of the tree, in a way that enabled the use of efficient on-line methods for its solution. The new data structure is also considerably faster than the reactive diffracting trees, because of the low-overhead, multilevel reaction part: the new reactive trees can shrink and expand many levels at a time without using clock readings. The self-tuning reactive trees[1], like the reactive diffracting trees, are aimed in general for applications where such distributed data structures are needed. Since the latter were introduced in the context of counting problems, we use similar terms in our description, for reasons of consistency.

The rest of this paper is organized as follows. Section 2 presents the key idea and the algorithm of the self-tuning reactive tree. Section 3 describes the implementation of the tree. Section 4 presents an experimental evaluation of the self-tuning reactive trees, compared with the reactive diffracting trees, on the Origin2000 platform, and elaborate on a number of properties of our algorithm. Section 5 concludes this paper. Due to the space constraint, the correctness proof of our algorithm is presented in [3].

2 Self-tuning Reactive Trees

2.1 Problem Description

The problem we are interested in is to construct a tree that satisfies the following requirements:

[1] We do not use term *diffracting* in the title of this paper since our algorithmic implementation does not use the *prism* construct, which is in the core of the algorithmic design of the (reactive) diffracting trees.

1. It must evenly distribute a set of concurrent process accesses to many small groups locally accessing shared data (counters at leaves), in a coordinated manner like the (reactive) diffracting trees. The step-property must be guaranteed.
2. Moreover, it must automatically and efficiently adjust its size according to its load in order to gain performance. It must not require any manually tuning parameters.

In order to satisfy these requirements, we have to tackle the following algorithmic problems:

1. Design a dynamic mechanism that would allow the tree to predict when and how much it should resize in order to obtain good performance whereas the load on it changes unpredictably. Moreover, the overhead that this mechanism will introduce should not exceed the performance benefits that the dynamic behavior itself will bring.
2. This dynamic mechanism should not only adjust the size of the tree in order to improve performance, but, more significantly, adjust it in a way that the tree still guarantees the fundamental properties of the structure, such as the step property.

2.2 Key Idea

The ideal reactive tree is the one in which each leaf is accessed by only one process(or) –holding a token [2] – at a time and the cost to traverse it from the root to the leaves is kept minimal. However, these two latency-related factors are opposite to each other, i.e. if we want to decrease the contention at the leaves, we need to expand the tree and so the cost to traverse from the root to the leaves increases.

What we are looking for is a tree where the *overall overhead*, including the *latency due to contention* at the leaves and the *latency due to traversal* from the root to the leaves, is minimal and with *no manual tuning*. In addition to this, an algorithm that can achieve the above, must also be able to cope with the following difficulties: If the tree expands immediately when the contention level increases, then it will pay the expensive cost for travel and this cost is going to be unnecessary if after that the contention level suddenly decreases. On the other hand, if the tree does not expand in time when the contention-level increases, it has to pay the large cost of contention. If the algorithm knew in advance about the changes of contention-levels at the leaves in the whole time-period that the tree operates, it could adjust the tree-size at each time-point in a way such that the overall overhead is minimized. As the contention-levels change unpredictably, there is no way for the algorithm to know this kind of information, i.e. the information about the future.

To overcome this problem, we have designed a reactive algorithm based on the online techniques that are used to solve the online currency trading problem [4].

Definition 1. *Let* surplus *denote the number of processors that exceeds the number of leaves of the self-tuning reactive tree, i.e. the subtraction of the number of the leaves from the maximal number of processors in the system that potentially want to access*

[2] For reasons of brevity, throughout the paper, instead of using the phrase “process(or) holding a token” we use simply the term process or processor.

the tree. The surplus *represents the contention level on the tree because the surplus processors cause contention on the leaves.*

Definition 2. *Let* latency *denote the latency due to traversal from the root to the leaves.*

Our challenge is to balance the trade-off between *surplus* and *latency*. Our solution for the problem is based on an optimal competitive algorithm called *threat-based algorithm* [4]. The algorithm is an optimal solution for the one-way trading problem, where the player has to decide whether to accept the current exchange rate as well as how many of his/her dollars should be exchanged to yens at the current exchange rate without knowledge on how the exchange rate will vary in the future.

2.3 The New Algorithm

In the self-tuning reactive trees, to adapt to the changes of the contention efficiently, a leaf should be free to shrink or grow to any level suggested by the reactive scheme in one adjustment step. With this in mind, we designed a data structure for the trees such that the time used for the adjustment and the time in which other processors are blocked by the adjustment are kept minimal. Figure 1 illustrates the self-tuning reactive tree data structure. Each balancer has a *matching* leaf with corresponding identity. Symmetrically, each leaf that is not at the lowest level of the tree has a *matching* balancer with corresponding identity. The squares in the figure are balancers and the circles are leaves. The numbers in the squares and circles are their identities. Each balancer has two outputs, $left$ and $right$, each of them being a pointer that can point to either a leaf or a balancer. A shrink or expand operation is essentially a switch of such a pointer (from the balancer to the matching leaf or from the leaf to the matching balancer, respectively). The solid arrows in the figure represent the present pointer contents.

Assume the tree has the shape as in Figure 1, where the solid arrows are the pointers' current contents. A processor p_i first visits the tree at its root IN, then following the root pointer visits balancer 1. When visiting a balancer, p_i switches the balancer's toggle-bit

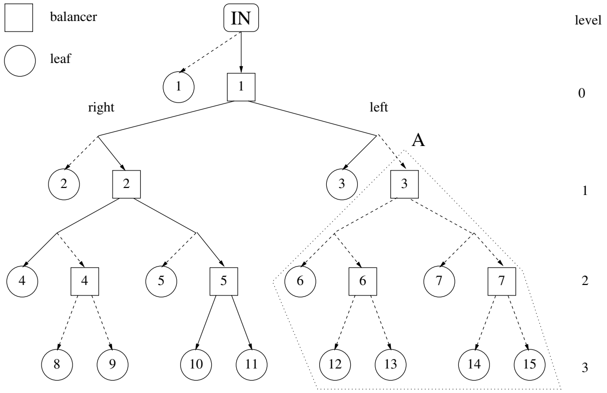


Fig. 1. A self-tuning reactive tree

to the other position (i.e. from left to right and vise-versa) and then continues visiting the next node according to the toggle-bit. When visiting a leaf L, p_i before taking an appropriate counter value and exiting, checks the *reaction condition* according to the current load at L. The reaction condition estimates which tree level is the best for the current load.

The reaction procedure. In order to balance the trade-off between *surplus* and *latency*, the procedure can be described as a game, which evolves in *load-rising* and *load-dropping transaction phases*.

Definition 3. *A* load-rising (resp. load-dropping) transaction phase *is a maximal sequence of subsequent visits at a leaf-node with monotonic non-decreasing (resp. non-increasing) estimated contention-level over the entire tree. A load-rising phase ends when a decrease in contention is observed; at that point a load-dropping phase begins.*

During a load-rising phase, a processor traversing that leaf may decide to expand the leaf to a subtree of depth that depends on the amount of the rising contention-level. That value is computed using the *threat-based on-line* method of [4], following the principle: “expand *just enough* to guarantee a bounded competitive ratio, even in the case that contention may drop to minimum at the next measurement”. Symmetric is the case during a load-dropping phase, where the reaction is to shrink a subtree to the appropriate level, depending on the measurement. The computation of the level to shrink to or to expand to uses the number of processors in the system as an upper bound of contention. The reaction procedure is described in detail in Section 3.2.

Depending on the result of checking the reaction condition, the processor acts as follows:

Recommended reaction: Grow to level l_{lower}, i.e. the current load is too high for the leaf L and L should expand to level l_{lower}. The processor, before exiting the tree through L, must help in carrying out the expansion task. To do so, the corresponding subtree must be constructed (if it was not already existent), the subtree’s counters’ (leaves’) values must be set, and the pointer pointing to L must switch to point to its corresponding balancer, which is the root of the subtree resulting from the expansion.

Recommended reaction: Shrink to level l_{higher}, the current load at the leaf L is too low and thus L would like to cause a shrink operation to a higher level l_{higher}, in order to reduce the latency of traversing from the root to the present level. This means that the pointer to the corresponding balancer (i.e. ancestor of L) at level l_{higher} must switch to point to the matching counter (leaf) and the value of that counter must be set appropriately. Let B denote that balancer. The sub-tree with B as a root contains more leaves than just L, which might not have decided to shrink to l_{higher}, and thus the processor must take this into account. To enable processors do this check, the algorithm uses an asynchronous vote-collecting scheme: when a leaf L decides to shrink to level l_{higher}, it adds its *weighted vote* for that shrinkage to a corresponding vote-array at balancer B.

Definition 4. *The weight of the vote of leaf L is the number of lowest-level leaves in the subtree rooted at the balancer matching L.*

As an example in Figure 1 the weight of the vote of leaf 4 is 2. Note that when voting for balancer B, the leaf L is not concerned about whether B has shrunk into its matching leaf or not. The processor that helps L write its vote to B's vote-array, will then check whether there are enough votes collected at B's vote-array. If there are enough votes collected at B' vote-array, i.e. if the sum of their weights is more than half of the total possible weight of the sub-tree rooted at B (i.e. if more than half of that subtree wants to shrink to the leaf matching B), the shrinkage will happen. After completing the shrinkage task, the processor increases and returns the counter value of L, thus exiting the tree. In the checking process, the processor will abort if the balancer B has shrunk already by a concurrent operation.

In the shrinkage procedure, the leaf matching B and the leaves of the sub-tree rooted at B must be locked in order to (i) collect their counters' values, (ii) compute the next counter value for the leaf matching B and (iii) switch the pointer from B to its matching leaf. Note that all the leaves of subtree B need to be locked *only if* the load on the subtree is *so small that it should be shrunk to a leaf*. Therefore, locking the subtree in this case effectively behaves as if locking a leaf (i.e. as it is done in the classical reactive diffracting trees) from the performance point of view.

Example of executing grow: Consider a processor p_i visiting leaf 3 in Figure 1, and let the result of the check be that the leaf should grow to sub-tree A with leaves $1\ , 13, 1$ and 15: The processor first constructs the sub-tree, whereas at the same time other processors may continue to access leaf 3 to get the counter values and then exit the tree without any disturbance. After that, it locks leaf 3 in order to (i) switch the pointer to balancer 3 and (ii) assign the proper values to counters $1\ , 13, 1$ and 15, then it releases leaf 3. At this point, the new processors following the left pointer of balancer 1 will traverse through the new sub-tree, whereas the old processors that were directed to leaf 3 before, will continue to access leaf 3's counter and exit the tree. After completing the expansion task, p_i continues its normal task to access leaf 3's counter and exits the tree.

Example of executing shrink: Consider a processor p_i visiting leaf 10 in Figure 1 and let the result of the reaction condition be that the subtree should shrink to leaf . Because the sub-tree rooted at balancer contains more leaves besides 10, which might not have decided to shrink to , processor p_i will check the votes collected at for shrinking to that level. Assume that leaf has voted for balancer , too. The weight of leaf 's vote is two because the vote represents leaves and 9 at the lowest level. Leaf 10's vote has weight 1. Therefore, the sum of the weights of the votes collected at balancer is 3. In this case, processor p_i will help balancer to perform the shrinkage task because the weight of votes, 3, is more than half of the total possible weight of the sub-tree (i.e. more than half of 4, which is the number of the leaves at the lowest level of the subtree – $\ , 9, 10$ and 11). Then p_i locks leaf and all the leaves of the sub-tree rooted at balancer , collects the counter values at them, computes the next counter value for leaf and switches the pointer from balancer to leaf . After that, all the leaves of the sub-tree are released immediately so that other processors can continue to access their counters. As soon as the counter at leaf is assigned the new value, the new processors going along the right pointer of balancer 1 can access the counter and exit the tree whereas the old processors are traversing in the old sub-tree. After completing the shrinkage task, the processor exits the tree, returning the value from counter 10.

Space needs of the algorithm. In a system with n processors, the algorithm needs $n-1$ balancer nodes and $n-1$ leaf nodes. Note that it may seem that the data structure for the self-tuning reactive trees uses more memory space than the data structure for the reactive diffracting trees, since it introduces an auxiliary node (matching leaf) for each balancer of the tree. However, this is actually splitting the functionality of a node in the reactive diffracting trees into two components, one that is enabled when the node plays the role of a balancer and another that is enabled when the node plays the role of a leaf (cf. also Section 3.3 and Section 3.4). In other words, the corresponding memory requirements are similar. From the structure point of view, splitting the node functionality is a fundamental difference between the self-tuning trees and the reactive diffracting trees. The voting arrays' space needs at each balancer are $O(k)$, which are similar to the space needs for the prism at each balancer of the reactive diffracting trees, where k is the number of leaves of the subtree rooted at the balancer.

3 Implementation

3.1 Preliminaries

Data structure and shared variables: Figure 3 describes the tree data structure and the shared variables used in the implementation.

The synchronization primitives used for the implementation are *test-and-set (TAS)*, *fetch-and-xor (FAX)* and *compare-and-swap (CAS)*. Their semantics are described in [3]. Moreover, in order to simplify the presentation and implementation of our algorithm, we define, implement and use advanced synchronization operations: *read-and-follow-link* and *conditionally-acquire-lock.* The read-and-follow-link operations and the conditionally-acquire-lock operation are outlined in pseudo-code in Fig. 2. The way

```
$NodeType$ ASSIGN($NodeType * trace_i, NodeType * child$)
    $*trace_i := child$;/*mark $trace_i$ under update,clearing mask-bit*/
    $temp := *child$; /*get the expected value*/
    $temp.mask := 1$; /*set the mask-bit*/
    if ($local := CAS(trace_i, child, temp)) = child$ then return $temp$;
    else return $local$;

$NodeType$ READ($NodeType * trace_i$)
    do
        $local := *trace_i$;
        if $local.mask = 0$ then /*$trace_i$ is marked*/
            $temp := *local$; /*help corresponding Assign() ...*/
            $temp.mask := 1$;
            $CAS(trace_i, local, temp)$;
    while($local.mask = 0$); /*... until the Assign() completes*/
    return $local$;

boolean ACQUIRELOCK_COND( int $lock$, int $Nid$)
    while (($CurOccId := CAS(lock, 0, Nid)) \neq 0$) do
        if $IsParent(CurOccId, Nid)$ then return $Fail$;
        Delay using exponential backoff;
    return $Success$;
```

Fig. 2. The read-and-follow-link operations (Assign/Read) and conditionally-acquire-lock operation (AcquireLock_cond)

```
type $NodeType$ = record $Nid : [1..MaxNodeId]; kind : \{BALANCER, LEAF\}; mask$: bit; end;
    $BalancerType$ = record $state : \{ACTIVE, OLD\}; level$ : int; $toggleBit$ : boolean;
        $parent : [1..MaxNodeId]; leftChild, rightChild : NodeType$;
        $votes$ : array$[1..SizeOfMySubtree]$ of int; end;
    $LeafType$ = record $state : \{ACTIVE, OLD\}; level, count, init$ : int;
        $parent : [1..MaxNodeId]; lock : \{0..MaxNodeId\}; contention, totLoadEst$ : int;
        $transPhase : \{RISING, DROPPING\}$;
        $latency, baseLatency, surplus, baseSurplus, oldSugLevel, sugLevel$ : int; end;
shared variables
    $Balancers$ : array$[0..MaxNodeId]$ of $BalancerType$;
    $Leaves$ : array$[1..MaxNodeId]$ of $LeafType$;
    $TokenToReact$ : array$[1..MaxNodeId]$ of boolean;
    $Tracing$ : array$[1..MaxProcs]$ of $[1..MaxNodeId]$;
private variables
    $MyPath$ : array$[1..MaxLevel]$ of $NodeType$; /*one for each processor*/

int CHECKCONDITION($LeafType\ L$)
C0  $TotLoadEst := MIN(MaxProcs, L.contention * 2^{L.level})$;
C1  $FirstInPhase := False$;
C2  if $(L.transPhase = RISING)$ and $(TotLoadEst < L.totLoadEst)$ then
        $L.transPhase := DROPPING; L.baseLatency := L.latency; FirstInPhase := True$;
C3  else if $(L.transPhase = DROPPING)$ and $(TotLoadEst > L.totLoadEst)$ then
        $L.transPhase := RISING; L.baseSurplus := L.surplus; FirstInPhase := True$;
C4  if $L.transPhase = RISING$ then $Surplus2Latency(L, TotLoadEst, FirstInPhase)$;
C5  else $Latency2Surplus(L, \frac{1}{TotLoadEst}, FirstInPhase)$;
    $L.totLoadEst := TotLoadEst; L.oldSugLevel := L.sugLevel$;
C6  $L.sugLevel := log_2(MaxProcs - L.surplus)$;
    if $L.sugLevel < L.level$ then return $SHRINK$;
    else if $L.sugLevel > L.level$ then return $GROW$;
    else return $NONE$;

SURPLUS2LATENCY($L, TotLoadEst, FirstInPhase$)
SL0  $X := L.surplus; baseX := L.baseSurplus; Y := L.latency$;
SL1  $rXY := TotLoadEst; LrXY := L.totLoadEst$;
SL2  if $FirstInPhase$ then
         if $rXY > mXY * C$ then $deltaX := baseX * \frac{1}{C} * \frac{rXY - mXY * C}{rXY - mXY}$; /*C: comp. ratio*/
SL3  else $deltaX := baseX * \frac{1}{C} * \frac{rXY - LrXY}{rXY - mXY}$;
SL4  $L.surplus := L.surplus - deltaX; L.latency := L.latency + deltaX * rXY$;

LATENCY2SURPLUS($L, \frac{1}{TotLoadEst}, FirstInPhase$)
/* symmetric to the above with: $X := L.latency; baseX := L.baseLatency; Y := L.surplus$;
    $rXY := \frac{1}{TotLoadEst}; LrXY := \frac{1}{L.totLoadEst}$;*/
```

Fig. 3. The tree data structure and CheckCondition, Surplus2Latency and Latency2Surplus procedures

these locking mechanisms interact and ensure safety and liveness in our data structure accesses is explained in the descriptions of the implementations of the *Grow* and *Shrink* procedures and is proven in [3].

3.2 Reaction Conditions

As mentioned in section 2.3, each leaf L of the self-tuning reactive tree estimates which level is the best for the current load. The leaf estimates the total load of tree by using the following formula:

$$TotLoadEst = L.contention * \quad ^{L.level}$$

line C0 in $CheckCondition()$ in Figure 3, where $MaxProcs$ is the maximum number of processors potentially wanting to access the tree and $L.contention$, the contention of a leaf, is the number of processors that currently visit the leaf. $L.contention$ is increased by one every-time a processor visits the leaf L and is decreased by one when a processor leaves the leaf. Because the number of processors accessing the tree cannot be greater than $MaxProcs$ we have an upper bound for the load: $TotLoadEst \leq MaxProcs$.

At the beginning, the initial tree is just a leaf, so the the initial $surplus$, $base Surplus$, is $MaxProcs - 1$ and the initial $latency$, $baseLatency$, is 0. Then, based on the contention variation on each leaf, the values of $surplus$ and $latency$ are updated according to the online trading algorithm. Procedure $Surplus\ \ Latency()$ (respectively $Latency\ \ Surplus()$) is invoked (lines C4, C5) to adjust the number of surplus processors that the tree should have at that time. The surplus value will be used to compute the number of leaves the tree should have and consequently the level the leaf L should shrink/grow to. .

Procedure $Surplus\ \ Latency(L, TotLoadEst,\ FirstInPhase)$ in Figure 3 exchanges $L.surplus$ to $L.latency$ according to the *threat-based algorithm* [4] using $TotLoadEst$ as exchange rate. For self-containment, the computation implied by this algorithm is explained below. In a load-rising transaction phase, the following rules must be followed:

1. The tree is expanded only when the estimated current total load is the highest so far in the present transaction phase.
2. When expanding, expand *just enough* to keep the competitive ratio $c = \varphi - \frac{\varphi-1}{\varphi^{1/(\varphi-1)}}$, where $\varphi = \frac{MaxProcs}{2}$, even if the total load drops to the minimum possible in the next measurement.

Following these, the number of leaves the tree should have more is:

$$deltaSurplus = baseSurplus * \frac{1}{C} * \frac{TotLoadEst - TotLoadEst^{-}}{TotLoadEst -}$$

where $TotLoadEst^{-}$ is the highest observed total load before the present measurement and $baseSurplus$ is the number of surplus processors at the beginning of the present transaction phase (line SL3, where mXY is the lower bound of the estimated total load). Everytime a new transaction phase starts, the value $baseSurplus$ is set to the last value of $surplus$ in the previous transaction phase (line C3). The parameter $FirstInPhase$ is used to identify whether this is the first exchange of the transaction phase. At the beginning,

$$surplus = baseSurplus = MaxProcs - 1$$

i.e. the tree degenerates to a node. Both variables $TotLoadEst^{-}$ and $baseSurplus$ are stored in fields $TotLoadEst$ and $baseSurplus$ of the leaf data structure, respectively.

Symmetrically, when the tree should shrink to reduce the traversal latency, the exchange rate is the inverse of the total load, $rXY = \frac{1}{TotLoadEst}$, which is increasing. In this case, the value of $surplus$ increases and that of $latency$ decreases.

3.3 Expanding a Leaf to a Sub-tree

A grow operation of a leaf L to a subtree T, whose root is L's matching balancer B and whose depth is $L.SugLevel - L.level$, essentially needs to (i) set the counters at the new leaves in T to proper values to ensure the step property; (ii) switch the corresponding child pointer of L's parent from L to B; and (iii) activate the nodes in T. (Figure 5 illustrates the steps taken in procedure grow, which is given in pseudocode in Figure 4.) Towards (i), it needs to:

```
GROW(int $Nid$) /*Leaves[Nid] becomes OLD;Balancers[Nid] and its subtree become ACTIVE*/
G0   $L := Leaves[Nid]$; $B := Balancers[Nid]$;
G1   forall $i$, $Read(Tracing[i])$ /* Can't miss any processors sincethe current ones go to Leaves[Nid]*/
         if $\exists$ pending processors in the subtree rooted at $B$ then return; /*abort*/
G2   for each balancer $B'$ in the subtree rooted at B,up to level $L.sugLevel - 1$
         forall $entries\ i : B'.votes[i] := 0$; $B'.toggleBit = 0$;
G3   for each leaf $L'$ at level $L.sugLevel$ of the subtree rooted at B,in decreasing order of nodeId do
         if not $AcquireLock\_cond(L'.lock, Nid)$ then Release all acquired locks; return; /*abort*/
G4   if (not $AcquireLock\_cond(L.lock, Nid)$) or $(L.state = OLD)$ then
     /*$1^{st}$: an ancestor activated an overlapping $Shrink$; $2^{nd}$:someone already made the expansion*/
         Release all acquired locks; return; /*abort*/
G5   Switch parent's pointer from L to $B$;
G6   forall $i$, $Read(Tracing[i])$ /*Can't miss any since the new ones go to $B$*/
         $ppL :=$ #(pending processors at $L$);
G7   $CurCount := L.count$; $L.state := OLD$;
G8   $Release(L.lock)$;
G9   for each balancer $B'$ as described in step $G2$ do $B'.state := ACTIVE$;
G10  for each leaf $L'$ as described in step $G3$ do
         update $L'.count$ using ppL and $CurCount$; $L'.state := ACTIVE$; $Release(L'.lock)$;
     return;/*Success*/

ELECT2SHRINK( int $Nid$, $NodeType\ MyPath[]$)
E0   $L := Leaves[Nid]$;/*the leaf asks to shrink*/
     if $L.oldSugLevel < L.sugLevel$ then /*new suggested level islower than older suggestion*/
E1      for($i$:= $L.oldSugLevel$; $i < L.sugLevel$; $i++$) do $Balancers[MyPath[i].Nid].votes[Nid] := 0$;
     else for ($i := L.sugLevel$; $i < L.oldSugLevel$; $i++$) do
E2          $B := Balancers[MyPath[i].Nid]$;
E3          $B.votes[Nid] := 2^{MaxLevel - L.level}$; $bWeight := 2^{MaxLevel - B.level}$; /*weight of $B$'s subtree*/
E4          if $\frac{\sum_i B.votes[i]}{bWeight} > 0.5$ then $Shrink(i)$; break;

SHRINK ( int $Nid$)/*Leaves[Nid] becomes ACTIVE; Balancers[Nid] and its subtree become OLD*/
S0   $B := Balancers[Nid]$; $L := Leaves[Nid]$;
S1   if $(TAS(TokenToReact[Nid]) = 1)$ then return; /*abort, someone is doing the shrinkage*/
S2   forall $i : Read(Tracing[i])$ /*can't miss any since the currentones go to $B$*/
         if $\exists$ pending processor at $L$ then return;/*abort*/
S3   if ( not $AcquiredLock\_cond(L.lock, Nid)$) or $(B.state = OLD)$ then
         /*$1^{st}$: some ancestor is performing Shrink; $2^{nd}$: someone already made the shrinkage*/
         Release possibly acquired lock; return; /*abort*/
S4   $L.state := OLD$; /*avoid reactive adjustment at $L$*/
S5   forall leaf $L'$ in $B$'s subtree, in increasing order of nodeId do
         $AcquireLock\_cond(L'.lock, Nid)$; /*No fails expected since Grow operations by ancestors
                                                       will abort at G1*/
S6   Switch the parent's pointer from $B$ to $L$
S7   forall $i : Read(Tracing[i])$; $eppB :=$ #(effective pending processors in $B$'s subtree;
         /*can't miss any since the new ones go to $L$*/
S8   for each balancer $B'$ in the subtree rooted at $B$ do $B'.state := OLD$;
     $SL := \emptyset$; $SLCount := \emptyset$;
S9   for each leaf $L'$ in the subtree rooted at $B$ do
         if $(L.state = ACTIVE)$ then $SL := \cup L'$; $SLCount := \cup L'.count$; $L'.state := OLD$;
         $Release(L'.lock)$;
S10  $L.count := f(eppB, SL, SLCount)$;
S11  $L.state := ACTIVE$;
S12  $Release(L.lock)$;
S13  $Reset(TokenToReact[Nid])$;
```

Fig. 4. The Grow, Elect2Shrink and Shrink procedures

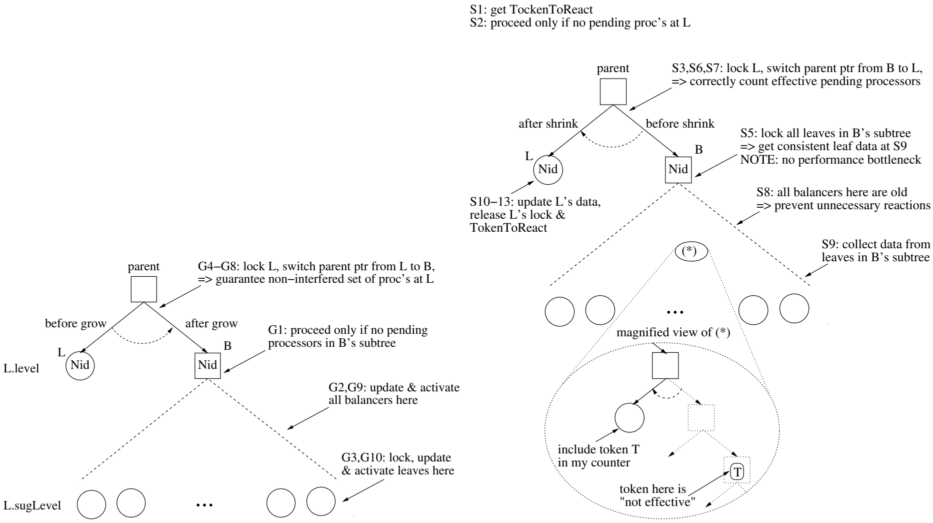

S1: get TockenToReact
S2: proceed only if no pending proc's at L
parent
S3,S6,S7: lock L, switch parent ptr from B to L,
=> correctly count effective pending processors
after shrink
before shrink
S5: lock all leaves in B's subtree
=> get consistent leaf data at S9
NOTE: no performance bottleneck
L
B
Nid
Nid
S10–13: update L's data,
release L's lock &
TokenToReact
S8: all balancers here are old
=> prevent unnecessary reactions
S9: collect data from
leaves in B's subtree
(*)
...
magnified view of (*)
include token T
in my counter
T
token here is
"not effective"
parent
G4–G8: lock L, switch parent ptr from L to B,
=> guarantee non–interfered set of proc's at L
before grow
after grow
G1: proceed only if no pending
processors in B's subtree
L
B
L.level
Nid
Nid
G2,G9: update & activate
all balancers here
G3,G10: lock, update
& activate leaves here
L.sugLevel
...

Fig. 5. Illustration for Grow and Shrink procedures

- make sure there are no pending tokens in T. If there are any, *Grow* aborts (step G1 in *Grow*), since it should not cause "old" tokens get "new" values (that would cause "holes" in the sequence of numbers received by all tokens in the end). A new grow operation will be activated anyway by subsequent tokens visiting L, since L has high contention.
- acquire the locks for the new leaves, to be able to assign proper counter values to them (step G3 in *Grow*) to ensure the step property.
- make a consistent measurement of the number of pending processors in L and $L.count$ to use in the computation of the aforementioned values for the counters. Consistency is ensured by acquiring L's lock (step G4) and by switching L's parent's pointer from L to B (i.e. performing action (ii) described above; step G5 in *Grow*), since the latter leaves a "non-interfered" set of processors in L.

Each of these locks' acquisition is *conditional*, i.e. if some ancestor of L holds it, the attempt to lock will return fail. In such a case the grow procedure aborts, since the failure to get the lock means that there is an overlapping shrink operation by an ancestor of L. (Note that overlapping grow operations by an ancestor of L would have aborted, due to the existence of the token (processor) at L (step G1 in *Grow*).) Furthermore, the new leaves' locks are requested in *decreasing* order of node-id, followed by the request of $L.lock$, to avoid deadlocks.

Towards action (iii) from above, the grow procedure needs to reset the tree's T balancers' toggle bits and vote arrays (before switching L's parent's pointer from L to B; step G2) and set the state values of all balancers and bottom-level leaves in T to ACTIVE (after having made sure that the growing will not abort; step G9-G10).

3.4 Shrinking a Sub-tree to a Leaf

Towards a decision of whether and where to shrink to, the token at a leaf L_0 with recommended reaction to shrink to level $L_0.SugLevel$ must add L_0's vote in the vote arrays of the balancers of its path from the root, starting from level $L_0.SugLevel$, up to level $L_0.level - 1$ (it must also take care to remove potentially existing older votes at layers above that; step E1 in $Elect\ \ Shrink$ in Figure 4). When a balancer with enough votes is reached, the shrink operation will start (steps E3-E4 in $Elect\ \ Shrink$). Figure 5 and Figure 4 illustrate and give the pseudocode of the steps taken towards shrinking.

Symmetrically to a grow operation, a shrink from a subtree T rooted at balancer B (with enough votes) to B's matching leaf L, essentially needs to (i) set the counter at L to the proper value to ensure the step property; (ii) switch the corresponding child pointer of B's parent from B to L; and (iii) de-activate the nodes in T. Towards (i), it needs to:

- make sure there are no pending tokens in L. If there are any, shrink aborts (step S2 in $Shrink$), since it should not cause "old" tokens get "new" values. Subsequent tokens' checking of the reaction condition may reinitiate the shrinking later on anyway.
- acquire L's lock (step S3), to be able to assign an appropriate counter value to it, to ensure the step property.
- make a consistent measurement of (1) the number of pending processors in T and (2) the values of counters of each leaf L' in T. Consistency is ensured by acquiring $L'.lock$ for all L' in T (step S5) and by switching B's parent's pointer from B to L (i.e. performing action (ii) described above; step S6 in $Shrink$), since the latter leaves a "non-interfered" set of processors in T.

Similarly to procedure grow, these locks' acquisition is conditional. Symmetrically with grow, the requests are made first to $L.lock$ and then to the locks of the leaves in T, in *increasing* order of node-id, to avoid deadlocks. Failure to get $L.lock$ implies an overlapping shrink operation by an ancestor of L. Note that overlapping grow operations by an ancestor of L would have aborted, due to the existence of the token at B (step G1 in $Grow$). Note also that an overlapping shrink by some of L's ancestors cannot cause any of the attempts to get some $L'.lock$ to fail, since that shrink operation would have to first acquire the lock for L (and if it had succeeded in getting that, it would have caused the shrink from B to L to abort earlier, at step S3 of $Shrink()$).

Towards action (iii) from above, the shrink procedure sets the balancers' and leaves' states in T to OLD (steps S8-S9 in $Shrink$), after having made sure that the shrink will not abort.

4 Evaluation

In this section, we evaluate the performance of the self-tuning reactive trees proposed here. We used the reactive diffracting trees of [2] as a basis of comparison since they are the most efficient reactive counting constructions in the literature.

The source code of [2] is not publicly available and we implemented it following exactly the algorithm as it is presented in the paper. We used the full-contention benchmark, the index distribution benchmark [2] and the surge load benchmark [2] on the SGI Origin2000, a popular commercial ccNUMA multiprocessor.

In [2], besides running the benchmarks on a non-commercially availiable machine with 32 processors (Alewife), the authors also ran them on the simulator simulating a multiprocessor system similar to Alewife with up to 256 processors.

The most difficult issue in implementing the reactive diffracting tree is to find the best folding and unfolding thresholds as well as the number of consecutive timings called *UNFOLDING_LIMIT, FOLDING_LIMIT* and *MINIMUM_HITS* in [2]. However, subsection *Load Surge Benchmark* in [2] described that the reactive diffracting tree sized to a depth 3 tree when they ran index-distribution benchmark [1] with 32 processors in the highest possible load ($work = 0$) and the number of consecutive timings was set at 10. According to the description, we run our implementation of the reactive diffracting tree on the ccNUMA Origin 2000 with 32 MIPS R10000 processors and the result is that folding and unfolding thresholds are and 1 microseconds, respectively. This selection of parameters did not only keep our experiments consistent with the ones presented in [1] but also gave the best performance for the diffracting trees in our system. Regarding the prism size (prism is an algorithmic construct used in diffracting process in the reactive diffracting trees), each node has $c\ ^{(d-l)}$ prism locations, where $c = 0.5$, d is the average value of the reactive diffracting tree depths estimated by processors passing the tree and l is the level of the node [2, 5]. The upper bound for adaptive spin *MAXSPIN* is 128 as mentioned in [1].

In order to make the properties and the performance of the self-tuning reactive tree algorithm presented here accessible to other researchers and to help reproducibility of our results, C code for the tested algorithms is available at `http://www.cs.chalmers.se/~phuong/sat_jul04.tar.gz`.

4.1 Full-Contention and Index Distribution Benchmarks

The system used for our experiments was a ccNUMA SGI Origin2000 with sixty four 195MHz MIPS R10000 CPUs with 4MB L2 cache each. The system ran IRIX 6.5. We ran the reactive diffracting tree *RD-tree* and the self-tuning reactive tree *ST-tree* in the full-contention benchmark, in which each thread continuously executed only the function to traverse the respective tree, and in the index distribution benchmark with $work = 500\mu s$ [2][1]. Each experiment ran for one minute and we counted the average number of operations per second.

Results: The results are shown in Figure 6 and Figure 7. The right charts in both the figures show the average depth of the ST-tree compared to the RD-tree. The left charts show the proportion of the ST-tree throughput to that of the RD-tree.

The most interesting result is that when the contention on the leaves increases, the ST-tree automatically adjusts its size close to that of the RD-tree that requires three experimental parameters for each specific system.

Regarding throughput and scalability, we observed that the ST-tree performs better than the RD-tree. This is because the ST-tree has a faster and more efficient reactive scheme. The surge load benchmark in the next subsection shows that the reactive trees *continuously* adjust their current size slightly around the average size corresponding to a certain load (cf. Figure 8). Therefore, an efficient adjustment procedure will significantly improve the performance of the trees.

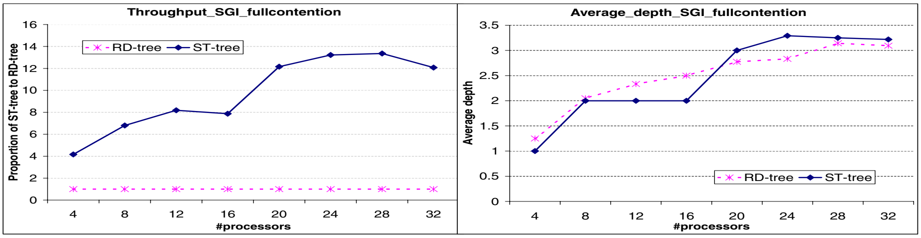


Fig. 6. Throughput and average depth of trees in the full-contention benchmark on SGI Origin2000

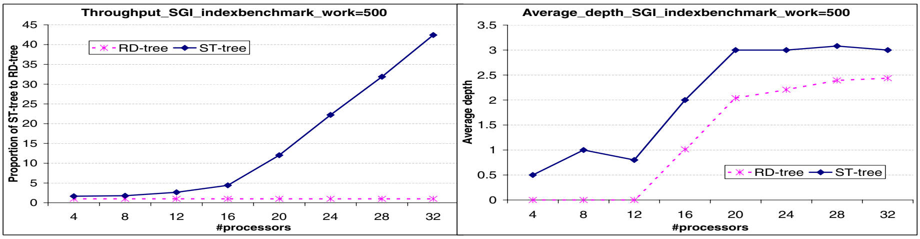


Fig. 7. Throughput and average depth of trees in the index distribution benchmark with $work = 500\mu s$ on SGI Origin2000

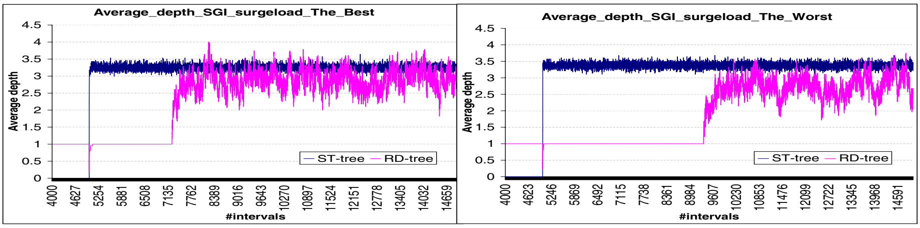


Fig. 8. Average depths of trees in the surge benchmark on SGI Origin2000, best and worst measurements. In a black-and-white printout, the darker line is the ST-tree

Studing the figures closer, in the full-contention benchmark (Figure 6), we can observe the scalability properties of the ST-tree, which shows increased throughput with increasing number of processors (as expected using the aforementioned arguments) in the left chart. The right chart shows that the average depth of the ST-trees is nearly the same as that of the RD-tree, i.e. the reaction decisions are pretty close.

In the index distribution benchmark with $work = 500\mu s$, which provides a lower-load environment, the ST-tree can be observed to show very desirable scalability behavior as well, as shown in Figure 7. The charts of the average depths of both trees have approximately the same shapes again, but the ST-tree expands from half to one depth unit more than RD-tree. This is because the throughput of the former was larger, hence the contention on the ST-tree leaves was higher than that on RD-tree leaves, and this made the ST-tree expand more.

4.2 Surge Load Benchmark

The benchmark shows how fast the trees react to contention variations. The benchmark is run on a smaller but faster machine[3], ccNUMA SGI2000 with thirty 250MHz MIPS R10000 CPUs with 4MB L2 cache each. On the machine the optimal folding and unfolding thresholds, which keep our experiments consistent with the ones presented in [1], are 3 and 10 microseconds, respectively. All other parameters are kept the same as the benchmarks discussed in the previous subsection.

In this benchmark we measured the average depth of each tree in each interval of 400 microseconds. The measurement was done by a monitor thread. At interval 5000, the number of threads was changed from four to twenty eight. The average depth of the trees at the interval 5001 was measured after synchronizing the monitor threads with all the new threads, i.e. the period between the end of interval 5000 and the beginning of interval 5001 was not 400 microseconds. Figure 8 shows the average depth of both trees from interval 4000 to interval 15000. The left chart shows the best reaction time figures for the RD-tree and the ST-tree; the right one shows the worst reaction time figures for the RD-tree and the ST-tree. In the benchmark, the ST-tree reached the suitable depth 3 for the case of 28 threads at interval 5004 in the best case and 5008 in the worst case, i.e. only after 5 to 8 intervals since the time all 28 threads started to run. The RD-tree reached level 3 at interval 7447 in the best case and at interval 9657 in the worst case. That means the reactive scheme introduced in this paper and used by the ST-tree makes the same decisions as the RD-tree, and, moreover, it reacts to contention variations much faster than the latter.

5 Conclusion

The self-tuning reactive trees presented in this work distribute the set of processors that are accessing them, to many smaller groups accessing disjoint critical sections in a coordinated manner. They collect information about the contention at the leaves (critical sections) and then they adjust themselves to attain adaptive performance. The self-tuning reactive trees extend a successful result in the area of reactive concurrent data structures, the reactive diffracting trees, in the following way:

- The reactive adjustment policy does not use parameters which have to be set manually and which depend on experimentation.
- The reactive adjustment policy is based on an efficient adaptive algorithmic scheme.
- They can expand or shrink many levels at a time with small overhead.
- Processors pass through the tree in only one direction, from the root to the leaves and are never forced to go back.

Moreover, the self-tuning reactive trees:

- have space needs comparable with that of the classical reactive diffracting trees

[3] This is because the first machine was replaced with that one at our computer center while this experimental evaluation was still in progress.

- exploit low contention cases on subtrees to make their locking process as efficient as in the classical reactive diffracting trees although the locking process locks more nodes at the same time.

Therefore, the self-tuning reactive trees can react quickly to changes of the contention levels, and at the same time offer a good latency to the processes traversing them and good scalability behavior. We have also presented an experimental evaluation of the new trees, on the SGI Origin2000, a well-known commercial ccNUMA multiprocessor. We think that it is of big interest to do a performance evaluation on modern multiprocessor systems that are widely used in practice.

Last, we would like to emphasize an important point. Although the new trees have better performance than the classical ones in the experimental evaluation conducted and presented here, this is not the main contribution of this paper. What we consider as main contribution is the ability of the new trees to self-tune their size efficiently without any need of manual tuning.

References

1. Shavit, N., Zemach, A.: Diffracting trees. ACM Trans. Comput. Syst. **14** (1996) 385–428
2. Della-Libera, G., Shavit, N.: Reactive diffracting trees. J. Parallel Distrib. Comput. **60** (2000) 853–890
3. Ha, P.H., Papatriantafilou, M., Tsigas, P.: Self-adjusting trees. Technical Report 2003-09, Computing Science, Chalmers University of Technology (2003) `http://www.cs.chalmers.se/~phuong/ SAT_TR.ps.gz`.
4. El-Yaniv, R., Fiat, A., Karp, R.M., Turpin, G.: Optimal search and one-way trading online algorithms. Algorithmica **30** (2001) 101–139
5. Shavit, N., Upfal, E., Zemach, A.: A steady state analysis of diffracting trees. Theory of Computing Systems **31** (1998) 403–423

Optimal Resilience Asynchronous Approximate Agreement

I i A , Yon n A i , n D nny Dol v

School of Computer Science and Engineering,
The Hebrew University of Jerusalem, Israel
{ittaia, mitmit, dolev}@cs.huji.ac.il

Abstract. Consider an asynchronous system where each process begins with an arbitrary real value. Given some fixed $\epsilon > 0$, an approximate agreement algorithm must have all non-faulty processes decide on values that are at most ϵ from each other and are in the range of the initial values of the non-faulty processes.

Previous constructions solved asynchronous approximate agreement only when there were at least $5t + 1$ processes, t of which may be Byzantine. In this paper we close an open problem raised by Dolev et al. in 1983. We present a deterministic optimal resilience approximate agreement algorithm that can tolerate any t Byzantine faults while requiring only $3t + 1$ processes.

The algorithm's rate of convergence and total message complexity are efficiently bounded as a function of the range of the initial values of the non-faulty processes. All previous asynchronous algorithms that are resilient to Byzantine failures may require arbitrarily many messages to be sent.

Keywords: approximate agreement, Byzantine agreement, asynchronous systems.

1 Introduction

In l i l By n in n l o l o o in wi o
ini i l v l n n on on o ini i l v l i i o
vin o l y o . In oxi v ion i i i
v l o ll non- l y o v n lly onv o n i o n
y o n $\epsilon > 0$.

I i w ll now in yn ono o ni ion o l in -
n i i o i l n o i ili y o vin v n on l y o [ ]. In
on , Dol v l. [3,], ow oxi n i o i l
in yn ono y v $5t+1$ o , t o w i y By n in .
In i w olv o n ion i y [3,]. W ow A -
oxi A n n wi $3t+1$ o , t o w i y
By n in . Fi l. [ ] ow i no oxi n o-
o ol wi $3t$ o l o n ol t By n in il . H n o
l o i o i l ili n .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 229–239, 2005.

T l n n y o n in o l n o o n
n o l n o n . W o n n o o n n il
in ion n ion o n o ini i l v l o non- l y o .
o n n i n y i in on o ll vio yn ono
ol ion [ ] in w i l y o n o o ol o n o n
i ily i (y n l) n o o n .

T l n in i o in in wo il in lo .
n i n yn ono v ion o li l -B o o o ol o S i n
n To [11]. T o il in lo i nov l ni . T wi-
n ni li i ili y o l y o o li o n o
v l y w l o oll . T i il in lo o v y ow l
n i n l non- l y o o i ly onv i v l .

T i l o y o wo on on ion o w o i
o A oxi A n o l . In yn ono v ion o ox-
i n , M n y n S n i [10] i ov i o l xi y y
in A n o o ol o Dol v [ ] il in lo . F
[5] iv l o i wi y o i lly o i l onv n o yn-
ono v ion o o l . F [6] l o iv i n l o i o
yn ono oxi n o l i ili n in w -
v i o il y o i ion n - il . Ki n A n
[9] iv y i yn ono l o i n wi n o By n in n
ni n il .

Ano l n iv o w nin o i o A n i o i
o ili i in ion o y only n ni x n y
o in ion (y xi in ni x ion). B [1] n
n o i $3t + 1$ ili n By n in A n o o ol wi o ili i
in ion. In on , o o o ol i ini i n lw y n
in ion.

1.1 Model and Problem Definition

on i o n o . P o o ni y lly onn yn-
ono n wo wi li l FIF nn l . M n will v n lly
iv ni n o n o n o i . T nn l w n ny wo
o in in FIF o y, i p n o q m n l n
m' n q will iv m n only l iv m'.

W t o o y By n in . All o o
ollow l o i n no non- l y.

A non- l y o in wi n i y l in v l
n x o (i ily ll) - $\epsilon > 0$. An
i y ollowin wo on i ion :

Agreement. All non- l y o v n lly l wi o v l
wi in ϵ o o ;

Validity. T v l o y non- l y o wi in n
o ini i l v l o non- l y o .

1.2 Notations

L V no o o , n G o non- l y o . H n $n=|V|$ n $|G|\geq n-t$.

L S no ni l i o l . In i iv ly S n o o o l n in w i i ion on i . Fo x l $\{1,1,3\}$ l $\{1,3,1\}$ iff o $\{1,3\}$. Fo lly l $\mathbb{R}$ no o l n $\mathbb{N}$ o n l n n S i n ion $S:\mathbb{R}\mapsto\mathbb{N}$ $\{r\in\mathbb{R}\mid S(r)\neq 0\}$ i ni . D n $|S|=\sum_{r\in\mathbb{R}}S(r)$, $\text{in}\,S=\text{in}_{S(r)\neq 0}\{r\in\mathbb{R}\}$, $\text{x}\,S=\text{x}_{S(r)\neq 0}\{r\in\mathbb{R}\}$, n o S $\delta(S)=\text{x}\,S-\text{in}\,S$.

iv n l i S no $s_1,s_2,\ldots,s_{|S|}$ v l o S o in non in o . Fo ny $t<|S|/$, n $trim(S,t)$ l i on inin v l $s_{t+1},s_{t+2},\ldots,s_{|S|-t-1}$ (ovin t l n t ll v l o S). D n

$$reduce(S,t)=\frac{\text{x}(trim(S,t))+\text{in}(trim(S,t))}{}\,.$$

iv n o o V, l P o (o ,v l) i . Fo lly, $P\subset(V\times\mathbb{R})$. D n $P_{|2}$ l i o v l o on oo in in P. To o n no ion , w x n l i o o o P, o x l $\text{x}\,P=\text{x}\,P_{|2}$, $reduce(P,t)=reduce(P_{|2},t)$.

W ollowin onv n ion o nin v l o v i l in x ion o o o ol. All o o o ol v x li i o n n in wi 1 n in n in y on i ion. iv n v i l x w no x_p^h v l o v i l x on o p w n p o l i h- o n .

2 Reliable Broadcast and a $4t+1$ Resiliency

T i i o [ ] i o l $n-t$ v l , i t l n t ll o v l , n n o o v in n ion o inin v l n x oxi ion. W in y no in w y l o i o Dol v l. [ ] il o $t+1$ o o w i o t y By n in . S o $t+1$ non l y o in wi 0 n no t non l y o in wi 1. T o l i inin t By n- in o y n onfli in v l o iff n o . S i lly, ll o in wi v l $i\in\{0,1\}$ y l $t+1$ v l l i o i in will o v l i n n v o onv . W ov o i i l y y in li l -B o .

In o in i ly $n-t$ v l , i l $t+1$ l o i $n-t$ v l v n n y li l -B o .

T o i o li l -B o ov v i ion o yn- ono li l -B o o [11]:

Correctness. I non- l y o p wi m on o n h - o li l -B o (m,h) n ll non- l y o will v n lly li l -A (p,m,h).

```
Reliable-Broadcast code for process $p$ with message $m$ on round $h$:
    send $(p, m, h)$ to all processes;

Echo() method for process $q$:
    upon receiving $(p, m, h)$ from $p$
        if $q$ never sent a message of the form $(p, \cdot, h)$ then
            send $(p, m, h)$ to all processes;
    upon receiving $(p, m, h)$ from at least $t + 1$ unique processes;
        if $q$ never sent a message of the form $(p, \cdot, h)$ then
            send $(p, m, h)$ to all processes;

Condition for Reliable-Accept$(p, m, h)$ at process $q$:
    Received $(p, m, h)$ from at least $n - t$ unique processes;
```

Fig. 1. Code for Reliable-Broadcast(m) and Reliable-Accept(p, m)

Non-forgeability. I non- l y o p o no o o n h li l -B o (m, h) n no non- l y o will v o li l -A (p, m, h).

Uniqueness. I non- l y o o li l -A (p, m, h) n n- o non- l y o o li l -A (p, m', h) n $m = m'$;

Lemma 1. [illegible]

Non- o ili y ol in non- l y o will n v iv non xi n i ly n n y iv o t in i . T o non- l y o will n v o non xi in n l ly will n v .

o n ol in v n lly v y non- l y o will iv i i o p o $t + 1$ in i n o non- o ili y only wo o ion .

Fo ni n , o on i ion li l -A (p, m, h) ol o non- l y o q n l $t + 1$ non- l y o v n (p, m, h) n ny o non- l y o n o $n - (t + 1)$ o o (p, m', h) wi $m' \neq m$, n m' will n v . □

iv n li l -B o i iv w n i l $t + 1$ ili n A oxi A n o o ol. In o n , o w i n il i o li l -A on $n - t$ iff n v l .

Theorem 1. U [illegible] lo $_2(\delta(U)/\epsilon)$ [illegible] ϵ [illegible]

```
Code for process $p$:

Local variables:
    $values \subset (V \times \mathbb{R})$ initially $values = \perp$;
    $init \in \mathbb{R}$; // the initial value;
    $val \in \mathbb{R}$ initially $val = init$;
    $round \in \mathbb{N}$ initially $round = 1$;

repeat:
    Reliable-Broadcast('value', $p, val, round$);
    $values := \perp$;
    repeat
        upon Reliable-Accept('value', $q, u, h$) and $h = round$ // the first time
            $values := values \cup (q, u)$;
    until $|values| \geq n - t$;
    $val := reduce(values, t)$;
    $round := round + 1$;
```

Fig. 2. The simple $4t + 1$ algorithm

T oo o i o n iv i l x i o l iv n o $3t + 1$ l o i n o li l -B o ni , v y wo non- l y o l $n-$ $t \geq$ t+1 o on v l in o n .

3 The $3t + 1$ Algorithm

W no o $3t$+1 ili n , i ly in li l -B o n i in i no no . In wo , wo o y l i o v l in only on v l , n i in l in l i will no in . Fo x l , o $n =$, $t = 1$ n l v l 0, 0, 1, 1; l y o n n ll o wi v l $i \in \{0, 1\}$ will iv 3 v l n i in i n will l i n no o will .

H n , o $3t + 1$ ili n l o i w n i ion l ni o in wi n . A wi n o o p i o w o $n-t$ v l w l o y p. P o p w i o $n - t$ wi n . Sin o $n - t$ wi n , v y wo o v l $t + 1$ o on wi n , n l on non- l y wi n . H vin o on non- l y wi n i li v y i o non- l y o v l $n - t$ o only v l .

i o i wi i o n . iv n wi i o n n n n o n , ivin o v i n will i n w w n o will l v n o n .

W l o n ni llow o o now w n o i on i v l n l . L U no l i o ini i l v l o non- l y o , i lly w i o o n n o o n (n n n o n) n ion o $\delta(U)$, n o ini i l v l o non-l y o (non- l y n).

W no in yn ono l o i o [ ] By n in o n in i ily i n low v l will o o ol o n o n i ily l (ni) n o o n .

In o o i v o n n i n y w loy i l ini-i l o n o o ol i non- l y n . T i i o o ll o (v n By n in on) o li l -B o v o o v l y . T i n o o o n v l ll in i n o ini i l v l U. W ow i ion o $\delta(U)$ y ny non-l y o i o n l in v l wi in ϵ o o .

Diff n o y v iff n i ion on n o o n i . H n , o l n o o o no l oo ly n o n v o in . S i lly, o wi n il i li l -A l $t+1$ ‘ l ’ n i o n l

```
Local variables:
    $values \subset (V \times \mathbb{R})$ initially $values = \bot$;
    $init \in \mathbb{R}$; // the initial value;
    $val \in \mathbb{R}$ initially $val = init$;
    $(\forall x \in V):\ report[x], proof[x] \subset (V \times \mathbb{R})$ initially $proof[x] := \bot$;
    $witnesses, proven \subset V$;
    $round, enough \in \mathbb{N}$ initially $round = 1$;
    $L \subset \mathbb{N}$ initially $L = \bot$;

Code for process $p$:
init();
repeat
    Reliable-Broadcast(‘value’, $p, val, round$);
    $values := \bot$;
    $(\forall x \in V):\ report[x] := \bot$;
    repeat
        // delay high round messages, discard low round messages
        upon Reliable-Accept(‘value’, $q, u, h$) and $h = round$
            FIFO-Broadcast(‘report’, $q, u, h$) to all;
            $values := values \cup (q, u)$;
        upon FIFO-Accept(‘report’, $q, u, h$) from process $r$ and $h = round$
            $report[r] := report[r] \cup (q, u)$;
        $witnesses := \{x \in V \mid report[x] \subseteq values$ and $|report[x]| \geq n - t\}$ ;
        check/decide();
    until $|witnesses| \geq n - t$;
    $val := reduce(values, t)$;
    $round := round + 1$;
```

Fig. 3. The $3t+1$ algorithm

```
Code for **init()**
    Reliable-Broadcast('init', $p$, $val$);
    **repeat**
        **upon** Reliable-Accept('init', $q, u$) (The first value from $q$)
                then $values := values \cup (q, u)$;
    **until** $|values| \geq n - t$;
    Reliable-Broadcast('proof', $p$, $values$);
    **repeat**
        **upon** Reliable-Accept('init', $q, u$) (The first value from $q$)
                then $values := values \cup (q, u)$;
        **upon** Reliable-Accept('proof', $q, vals$) (The first proof from $q$)
                then $proof[q] := vals$;
        $proven := \{v \in V \mid proof[v] \neq \bot$ and $proof[v] \subseteq values\}$;
    **until** $|proven| \geq n - t$;
    $values := \{(q, reduce(proof[q], t)) \mid q \in proven\}$;
    $val := reduce(values, t)$;
    $enough := \lceil \log_2(\delta(values)/\epsilon) \rceil + 1$;

Code for **check/decide()**
    **if** ($round = enough$) then Reliable-Broadcast('halt', $p$, $round$) to all;
    **upon** Reliable-Accept('halt', $q, h$) (the first halt from $q$) then $L := L \cup \{h\}$;
    **if** $|L| \geq t + 1$ and $round > \min(trim(L, t))$ then **decide** $val$ and **halt**;
```

Fig. 4. The **init()** and **check/decide()** methods for process p

n i ion o l on non- l y o w o ' l' i .

T o o $3t + 1$ l o i in Fi 3 n Fi .

4 Analysis

4.1 Informal Properties of Witness:

In o o v n in o n o p i l $n - t$ wi n . P o x i wi n o o p i $n - t$ v l x l i o w y p.

Sin ' o ' n vi FIF -B o n i x i non- l y wi n o o p, q n o p n q v $n - t$ v l x .

4.2 Liveness

Lemma 2. h h, $h + 1$

S in on i ion, l $S \subseteq G$ o non- l y o n v v n o o n $h + 1$.

v n lly v y $p \in G$ will li l -B o i v l . H n v n lly v y $p \in G$ will li l -A l $n-t$ v l . T o v y $p \in G$ will n l $n-t$ ‘ o ’ . H n v n lly ll $p \in S$ will li l -A v l in ‘ o ’ . H n ll $p \in G$ will v n lly v l $n-t$ wi n , n v n . □

Lemma 3. [illegible]

[illegible] S in on i ion, o o o non- l y o $S \subseteq G$ n v i .

W in y owin l on o l . v n lly y L o n n will i n *enough* v l o $t+1$ non- l y o n o l $t+1$ ‘ l ’ will n . ll o p l w n i o n n i l n in$(trim(L_p,t))$ n $|L_p| \geq t+1$ (l lin o [illegible] o). H n v n lly o non- l y o will l .

L h ini o n o o $p \in G$ l , n y L ll non- l y o will v n lly o n h. Sin ‘ l ’ n vi li l -B o , ll o non- l y o will v n lly iv o ‘ l ’ (wi o n v l-) p o l . H n ll non- l y o will v n lly v in$(trim(L_p,t)) \leq h$ n o v n lly l . □

4.3 Safety

Lemma 4 (Validity). [illegible] $p \in G$ [illegible] h,

$$\text{in}\, U \leq val_p^h \leq \quad \text{x}\, U \,.$$

[illegible] T oo i y in ion on o n n . l ly ini i l v l in U y ni ion. A in ll v l o vio o n (o ini i l v l o $h = 1$) o ll $p \in G$ in n , n n x v l val^h i o o *reduce*$(values^{h-1}, t)$ o o o v l w n y li l -B o (o in **init** o , i oo w n vi li l -B o). Sin o t By n in o , n *reduce* i t l n t ll v l , xi l n ini l inin v l will lw y in i n o xi l n ini l v l o o v l o non- l y o vio o n . H n v in in *reduce*$(values, t)$ will on v l in n o U y in ion y o i . □

T wi n o y i ollow :

Lemma 5. [illegible] p, q [illegible] h, [illegible]

$$\left|values_p^h \cap values_q^h\right| \geq n - t \,.$$

I non- l y o p,q ni o n h, y v l $t+1$ o on wi n . T i ollow o l $n-t$ wi n , n v y $n-t$ o $t+1$ in ion wi v y o o . H n p,q v l on o on non- l y wi n r.

By ni ion o wi n n FIF o i o ‘ o ’ - , $n-t$ v l y r will o in $values_p$ n in $values_q$. □

D n $U_i = \bigcup_{p \in G} val_p^i$ l i on inin val v l o ll non- l y o y ll o l o n i. W now ow n x on n i l in n .

Lemma 6.

$$\delta(U_i) \le \frac{\delta(U_{i-1})}{} .$$

By L 5 w now v y wo o v in o on l $n-t$ v l . L p,q wo i y non- l y o , wi $values_p^i, values_q^i$ i l i o v l . Wi o lo o n li y w $val_p^i \ge val_q^i$. D no $m =$ in(U_{i-1}) n $M =$ x(U_{i-1}). I i i n o ov ollowin :

$val_p^i - val_q^i \le \frac{M-m}{2}$

D no $R = values_p^i \cap values_q^i$, $|R| \ge n-t$ n no $V_p = trim(values_p^i, t)$, $V_q = trim(values_q^i, t)$.

L x i n o R, n $x \in V_q$ R l $n-t$ l n n w only i t o i . H n x$(V_q) \ge x$. In i ion, in$(V_q) \ge m$ $trim$ ov t ll l n in $values_q^i$. T o $val_q \ge \frac{m+x}{2}$. In i il ion, $x \in V_p$, w i i li in$(V_p) \le x$ n x$(V_p) \le M$ n $val_p \le \frac{M+x}{2}$. o inin wi ov w $val_p - val_q \le \frac{M-m}{2}$. □

4.4 Termination Detection

W now ow l o i n o i n ly ny o n o n non- l y v l o ϵ o o .

Lemma 7. k $k =$ in$\{enough_r \mid r \in G\}$ $p \in G$ k

$$(\forall p,q \in G)\ |val_p^k - val_q^k| \le \epsilon .$$

L p o $enough_p = k$. x in $n-t$ v l in $values_p$ n o **init** o . on i non- l y o q, i l o $values_q$ y n o i **init** o . D o v l n oo n vi li l -B o , o q n iv n o t v l no in $values_p$, ll o v l q

wi $values_p$. H n i w i t l n t ll v l in $values_q$ n o inin v l i in i n o $values_p$. Fo lly, w v

$$\text{in}\, values_p \le reduce(values_q, t) \le \text{x}\, values_p \,.$$

T n y i iv ly lyin L 6, ll non- l y o n o lo $_2(\delta(values_p)/\epsilon)$ o n , i v l will lo no . □

L U o ini i l v l o non- l y o n A o ll ini i l v l v n lly y li l -A y n o ini i l o n (o $U \subseteq A$). L $C = trim(A,t)$. l ly $\delta(C) \le \delta(U)$ ovin t l n t ll l n o A l in n i o n o U.

Lemma 8. [illegible]

$$\text{lo}\,_2\left(\frac{\delta(U)}{\epsilon}\right) .$$

Fo ny o q, n ov $r \in proven_q$ w v

$$\text{in}\, C \le reduce(proof[r]_q, t) \le \text{x}\, C$$

No i r y l y. T i i in $proof[r]$ i o $n-t$ v l- w n y li l -B o , n $proof[r] \subseteq A$ n i in $proof[r]$ l in n i ll n $\delta(C)$. H n in $U \le$ in $values_q$ n x $values_q \le$ x U, $values_q$ i in **init** o o $reduce(proof[r], t)$, o ll $r \in proven_q$. T o $enough_p \le$ lo $_2\left(\frac{\delta(U)}{\epsilon}\right)$ o ll $p \in G$.

L $E = \bigcup_{p \in G} enough_p$ n ll $p \in G$ l o in$(trim(E,t))$ o n . T i i y will v n lly iv $t+1$ ‘ l’ n i . How v o n in$(trim(E,t))$ no o n $n-t$ li n n nno v n . □

Theorem 2. [illegible] lo $_2(\delta(U))/\epsilon)$ [illegible] ϵ [illegible]

All non- l y o l y L 3. T in ion i in o lo $_2(\delta(U)/\epsilon)$ o n , o L . Sin in ion i $t+1$ *halt* , i o o n i l n $enough_p$ o o $p \in G$, n o L 6 n L i ion v l ϵ o o . Fin lly, L ov i ion v l in i ini i l v l o non- l y o . □

5 Conclusions

In i w olv o n ion l o o i in l olvin A oxi A n o l . T o o ol n li i ili y o l y o o infl n onv n o non- l y o . T nov l wi n ni in o v y ow l. W won ow l i i in olvin o o l . Fo x l , w i n i v o olvin lo yn oni ion o l ?

An in in o i i o n on o onv n . Now i i only n on n o v l o non- l y o , on n loo o n o i l onv n .

References

1. G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM Press, 1984.
2. D. Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1), 1982.
3. D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. In *Proceedings of the 3rd Symposium on Reliability in Distributed Systems*, 1983.
4. D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986.
5. A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 73–87. ACM Press, 1986.
6. A. D. Fekete. Asynchronous approximate agreement. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 64–76. ACM Press, 1987.
7. M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 59–70. ACM Press, 1985.
8. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
9. R. M. Kieckhafer and M. H. Azadmanesh. Reaching approximate agreement with mixed-mode faults. *IEEE Trans. Parallel Distrib. Syst.*, 5(1):53–63, 1994.
10. S. R. Mahaney and F. B. Schneider. Inexact agreement: accuracy, precision, and graceful degradation. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 237–249. ACM Press, 1985.
11. T. K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

Lock-Free and Practical Doubly Linked List-Based Deques Using Single-Word Compare-and-Swap

H° n S n ll n P ili T i

Department of Computing Science,
Chalmers University of Technology and Göteborg University,
412 96 Göteborg, Sweden
{phs, tsigas}@cs.chalmers.se
http://www.cs.chalmers.se/~{phs, tsigas}

Abstract. We present an efficient and practical lock-free implementation of a concurrent deque that supports parallelism for disjoint accesses and uses atomic primitives which are available in modern computer systems. Previously known lock-free algorithms of deques are either based on non-available atomic synchronization primitives, only implement a subset of the functionality, or are not designed for disjoint accesses. Our algorithm is based on a general lock-free doubly linked list, and only requires single-word compare-and-swap atomic primitives. It also allows pointers with full precision, and thus supports dynamic deque sizes. We have performed an empirical study using full implementations of the most efficient known algorithms of lock-free deques. For systems with low concurrency, the algorithm by Michael shows the best performance. However, as our algorithm is designed for disjoint accesses, it performs significantly better on systems with high concurrency and non-uniform memory architecture. In addition, the proposed solution also implements a general doubly linked list, the first lock-free implementation that only needs the single-word compare-and-swap atomic primitive.

1 Introduction

A (i. . o l - n) i n n l . Fo x l ,
o n o i l n in y o lin o
in o in y . A o o o ion , *PushRight*,
PopRight, *PushLeft*, n *PopLeft* o ion. T ni ion
o i li o v l , w *PushRight/PushLeft* o ion
n w v l o i /l o li . T *PopRight/PopLeft* o ion
o on in ly ov n n v l on i /l o
li .

To n on i n y o o j in on n nvi on n ,
o o on o i l x l ion, i. . o o o lo in . M l
x l ion y ' ov ll o n [1] i lo in ,

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 240–255, 2005.

i. . o on n o ion n no ny o w il o o i lo y lo . M l x l ion n l o lo , io i y inv ion n v n v ion.

In o o o l , v o o non- lo in l o i o o j . Non- lo in l o i o no involv - l x l ion, n o o no ff o o l lo in o l n . Lo - i l n ion non- lo in n n - l o on n ion y on n o ion n in l vin o i -o ion , lw y l on o ion will o . How v , i i o v ion o o o o ion o l o o o ion o n v ni . W i - [ ] l o i lo - n o ov y voi v ion w ll, ll o ion n n o ni in li i n o i own . n ly, o l o in l o ion- [3] i l n ion o non- lo in o i l n ion . T in o i l n ion w n lo - on n o no n o o ny on n o ion.

T i l n ion o lo - on n i ivi l , n n ly on in i o ly lin li o y li y, o y i in l lo o y l i l lo w lo o o . To o o nowl , xi no i l n ion o w i - , v l lo - i l n ion v n o o . How v , ll vio lo - l in v l i - o n , y i only i l n o o ion no lly o i wi n v on n y i ion [1] li A o l. [ ], o on o i w i i iv li Do l -Wo o -An -Sw (AS)[2] w i i no v il l in o n o y - . nw l [5] n AS - i l n ion w ll n l o ly lin li i l n ion [6], n i l o li ion i o AS - i l n ion [ ],[ ] wi l v ion y M in l. [9]. V loi [10] o n i l n ion o lo - o - ly lin li in o -An -Sw (AS)[3], o wi o ny o o l ion n i o no i l o i l n in . Mi l [11] v lo i l n ion on AS. How v , i i no i n o llow ll li o i join ll o ion v o yn oni , v n o y o on iff n n o . S - on ly, in o o o yn i xi i i i n x n

[1] The algorithm by Arora et al. does not support push operations on both ends, and does not allow concurrent invocations of the push operation and a pop operation on the opposite end.

[2] A CAS2 operations can atomically read-and-possibly-update the contents of two non-adjacent memory words. This operation is also sometimes called DCAS in the literature.

[3] The standard CAS operation can atomically read-and-possibly-update the contents of a single memory word.

AS o ion n o i lly o on wo j n wo , w i i no
v il l [4] on ll o n l o .

In i w n lo - l o i o i l n in on n
o ll li o i join (in n o ion
on iff n n o o no n ily in wi o). An
li i ion o i l o i ni l o [1] in M
00 . T l o i i i l n in o on yn oni ion i i iv
v il l in o n y . I llow oin wi ll i ion, n
o yn i xi i (in n o lo - y-
n i o y n l wi i n oll ion o), ill in
no l AS-o ion . T l o i i i in il l in i ,
o wi on nin n lyin lo - o y n -
n . In l o i i ion i n i o o ion
n n oo o i l n ion i lo - n lin i l [13] i
l o iv n.

W v o x i n o o n o o l o-
i wi wo o o i n l o i o lo - nown; [11]
n [9], l i l n in l o [1] n [15]. x i n
w o on iff n l i o o y i wi , o
9 o o iv ly. All y w nnin iff n o -
in y n w on iff n i . l ow
AS- l o i o o AS - i l n ion [5] o ny
n o n ny y . In non- ni o o y i wi
i on n ion o l o i , o i i join o y, o
i ni n ly n l o i in [11].

T o i o ni ollow . In S ion w i
y o y . T l l o i i i in S ion 3. T
x i n l v l ion i n in S ion . W on l wi
S ion 5.

2 System Description

no o o y l i- o o y on in o o
o wi i lo l o y. All no onn o o y
vi n in onn ion n wo . A o o-o in i nnin on
y o in i iv o ion . i n i lly x
on on o o o , w il o o n v (n) ny
i . T o-o in , o i ly nnin on iff n o o ,

[4] It is available on the Intel IA-32, but not on the Sparc or MIPS microprocessor architectures. It is neither available on any currently known and common 64-bit architecture.

[5] The CAS2 operation was implemented in software, using either mutual exclusion or the results from [15], which presented an software CASn (CAS for n non-adjacent words) implementation.

o j il in o y o o-o in n o ni . T yn oni i o ion on o j o -o ion on o o - o n o y. T o y y no o ni o ly i l o ll no in y ; o o n v iff n i on iff n o o y.

3 The New Lock-Free Algorithm

T l o i i on o ly lin li , Fi 1. To , v y no on in v l . T l o no i i in Fi 5 i i in i i l n ion. No o ly lin li lw y on in i n il y no .

In o o o ly lin li on ion on n n non-lo in , w in wo o n o i yn oni ion i i iv , F -An -A (FAA) n o -An -Sw (AS). Fi i i ion o i i iv w i v il l in o o n l o .

To in o l no o li w v o n iv o v n n x oin . T v o n on i n ly, no n ily ll on . ol ion i o o ly lin li in in ly lin li wi xili y in o ion in v oin , wi n x oin in o v oin . T , n x oin lw y o on i n in ly lin li , v oin only iv in o w o n vio no . T i i o i l o o v ion “l ” non- v oin will lw y oin o no i i ly o o o n no , n o “ in ” o i ion i i lw y

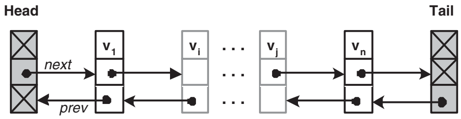


Fig. 1. The doubly linked list data structure

```
procedure FAA(address:pointer to word, number:integer)
    atomic do
        *address := *address + number;

function CAS(address:pointer to word, oldvalue:word, newvalue:word):boolean
    atomic do
        if *address = oldvalue then *address := newvalue; return true;
        else return false;
```

Fig. 2. The Fetch-And-Add (FAA) and Compare-And-Swap (CAS) atomic primitives

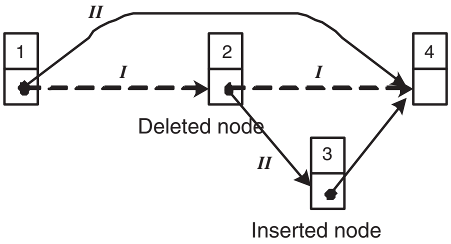


Fig. 3. Concurrent insert and delete operations can delete both nodes

o i l o v[6] o n x oin o i ly vio no .

n o l , i n l o non- lo in i l n ion on in ly lin li , i w n in in n w no in o li . B o lin li on o vio no i no o o l . I w n in n x oin o i vio no o i lly wi AS o ion, o oin o n w no , n n i i ly w vio no i l - n n w no will l w ll, ill in Fi 3. T v l ol ion o i o l . n ol ion i o AS o ion i n n wo oin o i lly, i o ion i no v il l in ny o n l i o o y . A on ol ion i o in xili y no [10] w n v y wo no l no , n l o in o y H i [16] i o l ion . T i l ion i o i lly o wi n x oin . Any on n in o ion will n no i o o i ly l ion , w n i AS o ion will il on in n x oin o o- - vio no . Fo o o ly lin li w n o in o l o w n in in in v oin .

In o o llow o y -wi yn i o y n l (w i o l lo - n v oll ion ili i), ll i ni n i o n i y oin v l o i l o n in o n x n v oin . In o o o i lly o n x n v oin o wi l ion on y Mi l [11], AS-o ion wo l n ili y o o i lly in l $30 + 30 + 1 = 61$ i on 3 - i y (n 6 +6 +1 = 1 5 i on 6 - i y oin n 6 i). In i o , o n 3 n 6 - i y only o AS o ion o in l wo - i .

How v , in o o ly lin li i l n ion, w n v n o n o v n n x oin in on o i , n - on i ion o i wi o i oin only involv oin i n . T o i i o i l o v n n x oin in

[6] As will be shown later, we have defined the deque data structure in a way that makes it possible to traverse even through deleted nodes, as long as they are referenced in some way.

wo , li in l ion in o wo . In o o v o n o l o i , l ion o n x oin o l lw y , n l ion o v oin o l o y ny o ion o v l ion on n x oin , o ny o in o . T , ll oin v l n , ill y only in n AS o ion .

3.1 The Basic Steps of the Algorithm

T in l o i , Fi , o in in n w no n i y o i ion in o o ly lin li will ollow :) A o i lly n x oin o o- - vio no ,) A o i lly v oin o o- -n x no . T in o l o i o l in no n i y o i ion ollowin :) S l ion on n x oin o o- - l no ,) S l ion on v oin o o- - l no ,) A o i lly n x oin o vio no o o- - l no ,) A o i lly v oin o n x no o o- - l no . A will own l in il i ion

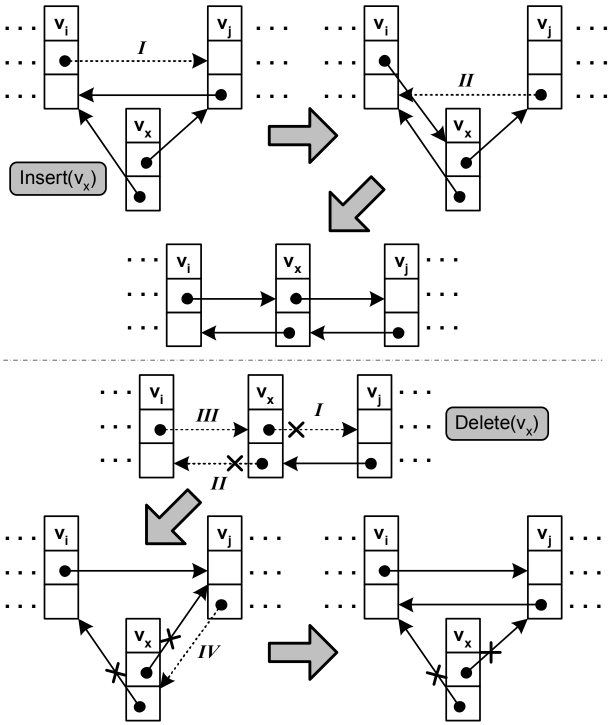


Fig. 4. Illustration of the basic steps of the algorithms for insertion and deletion of nodes at arbitrary positions in the doubly linked list, as described in Section 3.1

o l o i , l in ni n o li in o o i v lo - o y, ollowin in l o i o in in n l in .

3.2 Memory Management

A w on n ly (wi o i l ion) v in no will on in o ly llo n l i , w v o on i v l o o y n n . No no o l l i n n l - llo w il o o o i (o will) v in no . Fo i n y on w l o n o l o v n n x oin o l no , w wo l o wi o o - v in o o il y no w n v in l no w il v - in n o i ly in v o n n l i . T i n i i lly i o n o o ion y o l o l o ion in o . n on o y n n o l o SM o P o y Mi l [1] n H li y l. [1] iv ly, y n only n li i n o no o , n n l o - l o in ivi l n n v o n in ivi l no . How v , on o y n n o x l n o n in wo l i n o o n . T xi n l lo - n o n in y D l l. [1], o on non- v il l AS o i i iv .

Fo o i l n ion, w l lo - o y n n inv n y V loi [10] n o y Mi l n S o [19], w i o FAA n AS o i yn oni ion i iv . U in i w n no n only l i w n i no v o n x oin in li oin o i . n o l o wi i , n l o l wi n o n in , i i n no n l y li (i. . o o no o l y l n o , n - o no o i iv n o n , l o y no n y in). ol ion i o o o n i l y li n i ly o no i o i ly y l . T i i on y n in n x n v oin o l no o oin o iv no , in w y i on i n wi n i o o o ion .

T o y n n o l l o o n o - n oin ly. I w i ly - n n x o v oin in n o o in l n , i i o on in no n l i o w o l i . I n l o l ion i onn o v o n x oin w , in no i l . T y V loi l. o lo - oin - n in n n ily o o n l l ion .

T ollowin n ion n o n lin o o y n - n :

```
function MALLOC_NODE() :pointer to Node
function DEREF(address:pointer to Link) :pointer to Node
function DEREF_D(address:pointer to Link) :pointer to Node
```

function COPY(node:**pointer to** Node) :**pointer to** Node
procedure REL(node:**pointer to** Node)

T n ion *DEREF* n *DEREF_D* o i lly - n iv n lin n in n o n o o on in no . In lion o lin i , *DEREF* n ion n n NULL. T nion *MALLOC_NODE* llo n w no o o y ool. T n ion *REL* n n o n on o on in iv n no . I n o n o, n ion n ll *TerminateNode* nion will iv ly ll *REL* on no i no own oin o, n n i l i no . T *COPY* n ion in n o n o o on in iv n no .

A il o ow o i n ly ly o y n n o o i l o i no lw y ivi l, w will ovi il i ion o o wi il l o i i ion in i ion.

3.3 Pushing and Popping Nodes

T *PushLeft* o ion, Fi 5, in n w no l o oi ion in . T l o i ly i in lin L -L1 o in n w no (. .) w n no (.) n l o no (. . -), y o i lly n in n x oin o no . Bo yin o n x oin , i in lin L5 . . - no i ill v y n x no o , o wi . . - i in L6-L . A n w no n lly in , i i in lin P1-P13 o v oin o n x no . I i n il i i) i wi , ii) i i n x o n w no i l , o iii) n x no i no lon i ly n x o n w no . In ny o wo l , n o on n Po o P o ion , n on i ili y o v oin i n l o o . I , i o o i ili y n w no w l (n v oin o . . - no w o i ly l y y on n Po o ion) i ly o AS in lin P5, n n v oin i y llin *HelpInsert* n ion in lin P10. T linei ili y oin o *PushLeft* o ion i l AS o ion in lin L11.

T *PushRight* o ion, Fi 5, in n w no i o o i ion in . T l o i ly i in lin - 13 o in n w no (. .) w n i o no (.) n il no (. . -), y o i lly n in n x oin o , . no . B o yin o n x oin , i in lin 5 . . - no i ill v y n x no o , . , o wi , . i y llin *HelpInsert* n ion in 6, w i v oin o . . - no . A n w no n lly in , i i in lin P1-P13 o v oin o n x no , ollowin in *PushLeft* o ion. T lin i ili y oin o *PushRight* o ion i l AS o ion in lin 10.

```
union Link
        _: word
        ⟨p, d⟩: ⟨pointer to Node, boolean⟩

structure Node
        value: pointer to word
        prev: union Link
        next: union Link

// Global variables
head, tail: pointer to Node
// Local variables
node,prev,prev2,next,next2: pointer to Node
last,link1: union Link

function CreateNode(value: pointer to word) :pointer to Node
C1      node:=MALLOC_NODE();
C2      node.value:=value;
C3      return node;

procedure TerminateNode(node: pointer to Node)
RR1     REL(node.prev.p);
RR2     REL(node.next.p);

procedure PushLeft(value: pointer to word)
L1      node:=CreateNode(value);
L2      prev:=COPY(head);
L3      next:=DEREF(&prev.next);
L4      while T do
L5          if prev.next ≠ ⟨next,F⟩ then
L6              REL(next);
L7              next:=DEREF(&prev.next);
L8              continue;
L9          node.prev:=⟨prev,F⟩;
L10         node.next:=⟨next,F⟩;
L11         if CAS(&prev.next,⟨next,F⟩,⟨node,F⟩) then
L12             COPY(node);
L13             break;
L14         Back-Off
L15     PushCommon(node,next);

procedure PushRight(value: pointer to word)
R1      node:=CreateNode(value);
R2      next:=COPY(tail);
R3      prev:=DEREF(&next.prev);
R4      while T do
R5          if prev.next ≠ ⟨next,F⟩ then
R6              prev:=HelpInsert(prev,next);
R7              continue;
R8          node.prev:=⟨prev,F⟩;
R9          node.next:=⟨next,F⟩;
R10         if CAS(&prev.next,⟨next,F⟩,⟨node,F⟩) then
R11             COPY(node);
R12             break;
R13         Back-Off
R14     PushCommon(node,next);

procedure MarkPrev(node: pointer to Node)
MP1     while T do
MP2         link1:=node.prev;
MP3         if link1.d = T or CAS(&node.prev,link1,⟨link1.p,T⟩) then break;

procedure PushCommon(node, next: pointer to Node)
P1      while T do
P2          link1:=next.prev;
P3          if link1.d = T or node.next ≠ ⟨next,F⟩ then
P4              break;
P5          if CAS(&next.prev,link1,⟨node,F⟩) then
P6              COPY(node);
P7              REL(link1.p);
P8              if node.prev.d = T then
P9                  prev2:=COPY(node);
P10                 prev2:=HelpInsert(prev2,next);
P11                 REL(prev2);
P12             break;
P13         Back-Off
P14     REL(next);
P15     REL(node);

function PopLeft(): pointer to word
PL1     prev:=COPY(head);
PL2     while T do
PL3         node:=DEREF(&prev.next);
PL4         if node = tail then
PL5             REL(node);
PL6             REL(prev);
PL7             return ⊥;
PL8         link1:=node.next;
PL9         if link1.d = T then
PL10            HelpDelete(node);
PL11            REL(node);
PL12            continue;
PL13        if CAS(&node.next,link1,⟨link1.p,T⟩) then
PL14            HelpDelete(node);
PL15            next:=DEREF_D(&node.next);
PL16            prev:=HelpInsert(prev,next);
PL17            REL(prev);
PL18            REL(next);
PL19            value:=node.value;
PL20            break;
PL21        REL(node);
PL22        Back-Off
PL23    RemoveCrossReference(node);
PL24    REL(node);
PL25    return value;

function PopRight(): pointer to word
PR1     next:=COPY(tail);
PR2     node:=DEREF(&next.prev);
PR3     while T do
PR4         if node.next ≠ ⟨next,F⟩ then
PR5             node:=HelpInsert(node,next);
PR6             continue;
PR7         if node = head then
PR8             REL(node);
PR9             REL(next);
PR10            return ⊥;
PR11        if CAS(&node.next,⟨next,F⟩,⟨next,T⟩) then
PR12            HelpDelete(node);
PR13            prev:=DEREF_D(&node.prev);
PR14            prev:=HelpInsert(prev,next);
PR15            REL(prev);
PR16            REL(next);
```

Fig. 5. The algorithm, part 1(2)

T *PopLeft* o ion, Fi 5, i o l n n v l o l o no in . T l o i ly i in lin PL -PL o l o no ( ) l . B o yin o n x oin , i in lin PL i no y, n -on ly in lin PL9 no i no l y o l ion. I w o y, n ion n . I w o l ion, i i o n x oin o no y llin *HelpDelete* n ion, n n i o l o no . I v oin o w in o , i i o i y llin *HelpInsert* n ion. A no n lly y l AS o ion in lin PL13, i i in lin PL1 o n x oin o no y llin *HelpDelete* n ion, n in lin PL16 o v oin o no y llin *HelpInsert* n ion. A i , i i in lin PL 3 o o i l y li n in l y llin *RemoveCrossReference* n ion. T lin i ili y oin o *PopLeft* o ion il , i o ion o n x oin in lin PL3. T lin i ili y oin o *PopLeft* o ion , i o ion o n x oin in lin PL3.

T *PopRight* o ion, Fi 5, i o l n n v l o i o no in . T l o i ly i in lin P -P 19 o i o no ( ) l . B o yin o n x oin , i i) in lin P no i no l y o l ion, ii) in lin v oin o il ( ) no i o , n iii) in lin P i no y. I w o y, n ion n . I w o l ion o v oin o no w in o , i i o v oin o no y llin *HelpInsert* n ion, n n i o i o no . A no n lly i ollow *PopLeft* o ion. T lin i ili y oin o *PopRight* o ion il , i o ion o n x oin in lin P . T lin i ili y oin o *PopRight* o ion , i AS -o ion in lin P 11.

3.4 Helping and Back-Off

T *HelpDelete* - o , Fi 6, i o l ion o v oin n n o i lly n x oin o vio no o o- - l no , l llin n 3 o ov ll no l ion . T l o i n in lin HD1 l ion on v oin o iv n no i . I n ly i in lin HD6-HD3 o l (in n o in o n x oin in o no) iv n no ( ) y n in n x oin o vio non- no . Fi , w n ly n x oin o no i lw y in o no ( ) o i n v oin i lw y in o no ( ) o l (no n ily). B o yin o n x oin wi AS o ion in lin HD3 ,

i in lin HD6 i no l y l , in lin HD
no i no , in lin HD1 no i no , n in HD
i vio no o . I i , i i o
n x no . I i w i n o l i o w n

```
PR17        value:=node.value;
PR18        break;
PR19      Back-Off
PR20  RemoveCrossReference(node);
PR21  REL(node);
PR22  return value;

procedure HelpDelete(node: pointer to Node)
HD1   MarkPrev(node);
HD2   last:=⊥;
HD3   prev:=DEREF_D(&node.prev);
HD4   next:=DEREF_D(&node.next);
HD5   while T do
HD6     if prev = next then break;
HD7     if next.next.d = T then
HD8       MarkPrev(next);
HD9       next2:=DEREF_D(&next.next);
HD10      REL(next);
HD11      next:=next2;
HD12      continue;
HD13    prev2:=DEREF(&prev.next);
HD14    if prev2 = ⊥ then
HD15      if last ≠ ⊥ then
HD16        MarkPrev(prev);
HD17        next2:=DEREF_D(&prev.next);
HD18        if CAS(&last.next,⟨prev,F⟩
 ,⟨next2,F⟩) then REL(prev);
HD19        else REL(next2);
HD20        REL(prev);
HD21        prev:=last;
HD22        last:=⊥;
HD23      else
HD24        prev2:=DEREF_D(&prev.prev);
HD25        REL(prev);
HD26        prev:=prev2;
HD27      continue;
HD28    if prev2 ≠ node then
HD29      if last ≠ ⊥ then REL(last);
HD30      last:=prev;
HD31      prev:=prev2;
HD32      continue;
HD33    REL(prev2);
HD34    if CAS(&prev.next,⟨node,F⟩
 ,⟨next,F⟩) then
HD35      COPY(next);
HD36      REL(node);
HD37      break;
HD38    Back-Off
HD39  if last ≠ ⊥ then REL(last);
HD40  REL(prev);
HD41  REL(next);
```

```
function HelpInsert(prev, node: pointer to
 Node): pointer to Node
HI1   last:=⊥;
HI2   while T do
HI3     prev2:=DEREF(&prev.next);
HI4     if prev2 = ⊥ then
HI5       if last ≠ ⊥ then
HI6         MarkPrev(prev);
HI7         next2:=DEREF_D(&prev.next);
HI8         if CAS(&last.next,⟨prev,F⟩
 ,⟨next2,F⟩) then REL(prev);
HI9         else REL(next2);
HI10        REL(prev);
HI11        prev:=last;
HI12        last:=⊥;
HI13      else
HI14        prev2:=DEREF_D(&prev.prev);
HI15        REL(prev);
HI16        prev:=prev2;
HI17      continue;
HI18    link1:=node.prev;
HI19    if link1.d = T then
HI20      REL(prev2);
HI21      break;
HI22    if prev2 ≠ node then
HI23      if last ≠ ⊥ then REL(last);
HI24      last:=prev;
HI25      prev:=prev2;
HI26      continue;
HI27    REL(prev2);
HI28    if link1.p = prev then break;
HI29    if prev.next = node and CAS(
 &node.prev,link1,⟨prev,F⟩) then
HI30      COPY(prev);
HI31      REL(link1.p);
HI32      if prev.prev.d ≠ T then break;
HI33    Back-Off
HI34  if last ≠ ⊥ then REL(last);
HI35  return prev;

procedure RemoveCrossReference(
 node: pointer to Node)
RC1   while T do
RC2     prev:=node.prev.p;
RC3     if prev.prev.d = T then
RC4       prev2:=DEREF_D(&prev.prev);
RC5       node.prev:=⟨prev2,T⟩;
RC6       REL(prev);
RC7       continue;
RC8     next:=node.next.p;
RC9     if next.prev.d = T then
RC10      next2:=DEREF_D(&next.next);
RC11      node.next:=⟨next2,T⟩;
RC12      REL(next);
RC13      continue;
RC14    break;
```

Fig. 6. The algorithm, part 2(2)

o on o i vio no n o wi n l ion. T i x l ion i only i n x oin o non- no o n o v (i. . . i v li). wi i i no vio no o i i o n x no .

T *HelpInsert* - n ion, Fi 6, i o v oin o no n n n n o o i ly i vio no , l llin o ov ll in ion o o ov ll l ion . T l o i ly i in lin HI -HI33 o o v oin o iv n no ( ), iv n ion o vio (no n ily i ly vio) no ( ). B o yin o v oin wi AS o ion in lin HI 9, i in lin HI no i no , in lin HI19 i no , n in lin HI i vio no o . I i w i n o l i o w n o on o i vio no n o wi n l ion. T i x l ion i only i n x oin o non- no o n o v (i. . . i v li). I i , o i o . wi i i no vio no o i i o n x no . I in lin HI 9 , i o o i ili y no w l (n v oin o w o i ly l y y on n Po o ion) i ly o AS o ion. T i i in lin HI3 n n i o i ly i wi n w no .

B *HelpDelete* n *HelpInsert* o n in l o i o “ l in ” l o ion i o wi o o o o on n o ion , l o i i i l o - iv w ll lly on n y . In lly on n y o , l in y w ll vy on n ion on o i i i iv , n own o n i ni n ly. T o l o i , n o on iv il AS o ion (i. . il o l on n o ion) n o ion in o -off o . W n in -off o , o no in o w il , n in i w y voi i in on n o ion i o - wi o low . T ion o -off i ini i li o o v l (. . o o ion l o n o) o n o ion, n o on iv n in o -off o in on o ion invo - ion, ion o -off i n in o , . . in x on n i lly.

3.5 Avoiding Cyclic Garbage

T *RemoveCrossReference* - o , Fi 6, i o o - n w n iv n no ( ) n ny o no i n , y ly in v n n x oin lon y n lly no . Fi , w n ly v o n x l o i no on n ly y ny o o ion, i o i only ll y in o ion l no n o n x n v oin n ny on n in AS will il.

B o o i ni , i in lin 3 vio no ( ) i no lly , n in lin 9 n x no ( ) i no lly . A lon i i i v o l , n lon i i i v o i , w il on in o ly in v o n x l o in lin 5 o 11.

3.6 General Operations of Doubly Linked Lists and Correctness Proofs

D o i ion , il i ion o n l o ion o o ly lin li (i. . v l n i y in n l) w ll il oo o o n o lo - n lin i ili y i i i in n x n v ion o i [0].

4 Experimental Evaluation

In o x i n , on n o 1000 n o ly o n n i l o ion on , wi i i ion o 1/ *PushRight*, 1/ *PushLeft*, 1/ *PopRight* n 1/ *PopLeft* o ion . x i n w 50 i , n n v x ion i o x i n w i- . x ly n o o ion w o o ll iff n i l n ion o . B i o i l n ion, w l o o x i n wi lo - i l n ion y Mi l [11] n i - l n ion y M in l. [9], wo o o i n lo - v n o o . T l o i y M in l. w i l n o wi o on in o y n n y D l l. [1]. How- v , o [9] n [1] o i o ion AS w i i no v il l in ny o n y , AS o ion w i l n in o w - in wo iff n o . T o w o i l n AS in l x l ion (o o in [9]). T o o w o i l n AS in on o o i n o w i l n ion o ASN nown o l n o [9] n [1], i. . i l n ion y H i l. [15].

A l n- o ion w o j o - x i n in iff n i l n ion. All i l n ion w i n in n o il wi i o i i ion l v l. T o i i iv w i n in - ly l n .

T x i n w o in iff n n o , v yin o 1 o wi in in . T iff n l o w , wi v yin n o o o n l v l o o y i i ion. To i ly - iv nvi on n , w o o x i n on o l- o o P n i II P nnin Lin x, n S n Ul 0 y n- nin Sol i . wi o o . In o o v l o l o i wi ll on n y w l o S I i in 000 y nnin I ix 6.5 wi 9 50 MH MIPS 10000 o o . T l o x i n own in

Fi . T v x ion i i wn n ion o n o .

l ow o AS- l o i o o AS - i l n ion o ny n o . Fo y wi low o i on n y n ni o o y i , [11] o n . How v , o y wi ll on n y n non- ni o o y i o l o i o i ni n ly n [11] o n o , i on n o n o o l o i o o ll li o i join .

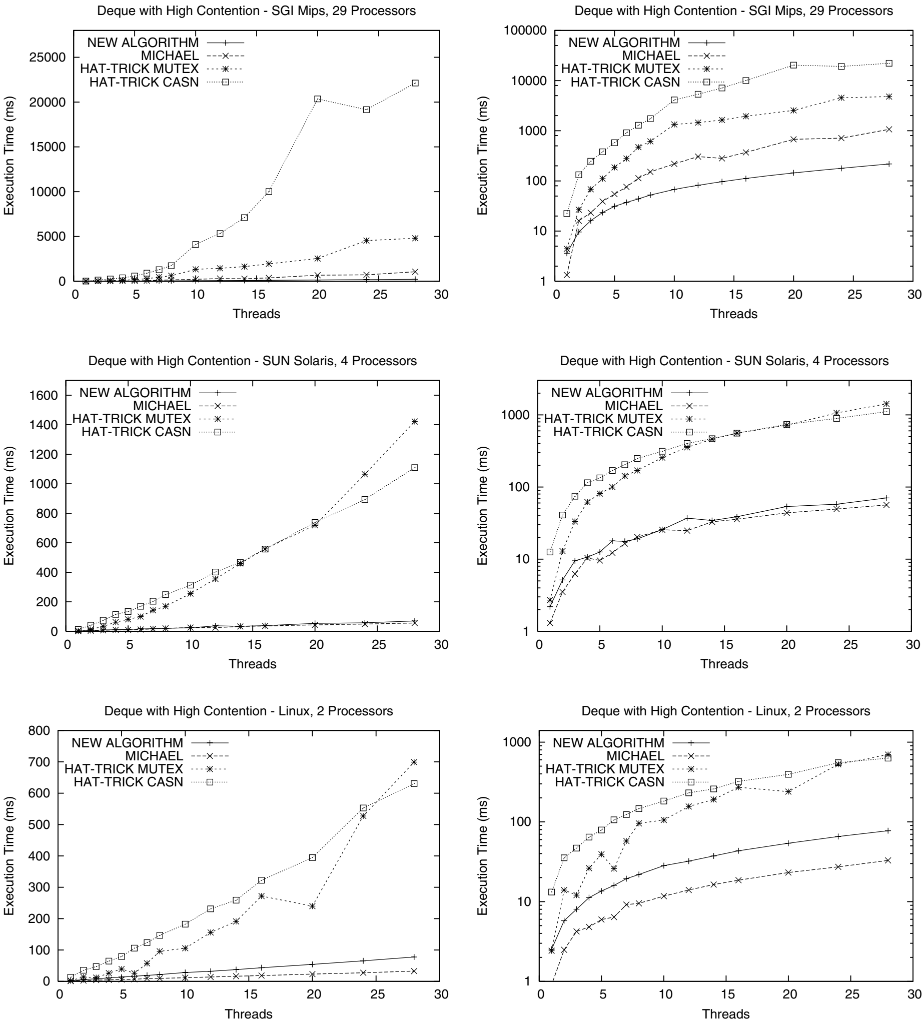


Fig. 7. Experiment with deques and high contention. Logarithmic scales in the right column

5 Conclusions

W v n lo - l o i i i l n ion o on n
ll ollowin : i) i o ll li o i join
, ii) lly i lo - o y n n , iii)
o i i i iv w i v il l in o n o y , n iv)
llow oin wi ll i ion o , n o yn i
i . In i ion, o o ol ion l o i l n ll n n l
o ion o n l o ly lin li in lo - nn .
T o ly lin li o ion l o o ini i n w ll n
v l o v n l no , n o i l o on n
li ion o lin li in i .

W v o x i n o o n o o l o-
i wi wo o o i n l o i o lo - nown, in
ll i l n ion o o l o i . T x i n ow o i l-
n ion o i ni n ly on y wi i on n y n
non- ni o o y i .

W li v o i l n ion i o i ly i l in o l i-
o o li ion . W n ly in o o in i in o N BL [1]
li y.

References

1. Silberschatz, A., Galvin, P.: Operating System Concepts. Addison Wesley (1994)
2. Herlihy, M.: Wait-free synchronization. ACM Transactions on Programming Languages and Systems **11** (1991) 124–149
3. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: Proceedings of the 23rd International Conference on Distributed Computing Systems. (2003)
4. Arora, N.S., Blumofe, R.D., Plaxton, C.G.: Thread scheduling for multiprogrammed multiprocessors. In: ACM Symposium on Parallel Algorithms and Architectures. (1998) 119–129
5. Greenwald, M.: Non-Blocking Synchronization and System Design. PhD thesis, Stanford University, Palo Alto, CA (1999)
6. Greenwald, M.: Two-handed emulation: how to build non-blocking implementations of complex data-structures using DCAS. In: Proceedings of the twenty-first annual symposium on Principles of distributed computing, ACM Press (2002) 260–269
7. Agesen, O., Detlefs, D., Flood, C.H., Garthwaite, A., Martin, P., Shavit, N., Steele Jr., G.L.: DCAS-based concurrent deques. In: ACM Symposium on Parallel Algorithms and Architectures. (2000) 137–146
8. Detlefs, D., Flood, C.H., Garthwaite, A., Martin, P., Shavit, N., Steele Jr., G.L.: Even better DCAS-based concurrent deques. In: International Symposium on Distributed Computing. (2000) 59–73
9. Martin, P., Moir, M., Steele, G.: DCAS-based concurrent deques supporting bulk allocation. Technical Report TR-2002-111, Sun Microsystems (2002)
10. Valois, J.D.: Lock-Free Data Structures. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York (1995)

11. Michael, M.M.: CAS-based lock-free algorithm for shared deques. In: Proceedings of the 9th International Euro-Par Conference. Lecture Notes in Computer Science, Springer Verlag (2003)
12. Sundell, H., Tsigas, P.: Lock-free and practical deques using single-word compare-and-swap. Technical Report 2004-02, Computing Science, Chalmers University of Technology (2004)
13. Herlihy, M., Wing, J.: Linearizability: a correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems **12** (1990) 463–492
14. Detlefs, D., Martin, P., Moir, M., Steele Jr, G.: Lock-free reference counting. In: Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing. (2001)
15. Harris, T., Fraser, K., Pratt, I.: A practical multi-word compare-and-swap operation. In: Proceedings of the 16th International Symposium on Distributed Computing. (2002)
16. Harris, T.L.: A pragmatic implementation of non-blocking linked lists. In: Proceedings of the 15th International Symposium of Distributed Computing. (2001) 300–314
17. Michael, M.M.: Safe memory reclamation for dynamic lock-free objects using atomic reads and writes. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing. (2002) 21–30
18. Herlihy, M., Luchangco, V., Moir, M.: The repeat offender problem: A mechanism for supporting dynamic-sized, lock-free data structure. In: Proceedings of 16th International Symposium on Distributed Computing. (2002)
19. Michael, M.M., Scott, M.L.: Correction of a memory management method for lock-free data structures. Technical report, Computer Science Department, University of Rochester (1995)
20. Sundell, H.: Efficient and Practical Non-Blocking Data Structures. PhD thesis, Department of Computing Science, Chalmers University of Technology (2004)
21. Sundell, H., Tsigas, P.: NOBLE: A non-blocking inter-process communication library. In: Proceedings of the 6th Workshop on Languages, Compilers and Runtime Systems for Scalable Computers. (2002)

A Dynamic Reconfiguration Tolerant Self-stabilizing Token Circulation Algorithm in Ad-Hoc Networks

Hi o K w[1,⋆] n M i Y i[2,⋆⋆]

[1] Faculty of Engineering, Hiroshima University,
1-4-1 Kagamiyama, Higashi Hiroshima, Hiroshima, Japan
h.kakugawa@computer.org

[2] Graduate School of Information Science and Electrical Engineering,
Kyushu University, 6-10-1 Hakozaki, Fukuoka, Japan
mak@csce.kyushu-u.ac.jp

Abstract. Ad-hoc networks do not provide an infrastructure for communication such as routers and are characterized by 1) quick changes of communication topology and 2) unstable system behaviors. Self-stabilizing algorithms have been studied well to design stable distributed algorithms on unstable systems, but they are not requested to be adaptive to dynamic topology changes. We in this paper propose a new concept of dynamic reconfiguration tolerant (DRT for short) self-stabilizing algorithm, which is a self-stabilizing algorithm that is also robust against dynamic changes of topology. We next propose a DRT self-stabilizing token circulation algorithm. It deterministically circulates a token through a spanning tree edges in an asymptotically optimal time $O(n)$, once the system is stabilized. The spanning tree will converge to the minimum spanning tree, if the network remains static.

Keywords: token circulation, self-stabilization, ad-hoc network, spanning tree.

1 Introduction

A - o n wo on i o o il in l wi wi l o ni ion - vi . T i no - xi in in o o ni ion, n in l i onn l o n - o n wo wi o on in i . T i i in in o n - , i o ollowin ni l i l i , w n o i l n li ion in - o n wo ; ( ) in no oin o on o il in l , o il in l o

[⋆] This work is partially supported by the Ministry of Education, Culture, Sports, and Technology, Grant-in-Aid 15700017.

[⋆⋆] This work is partially supported by the Ministry of Education, Culture, Sports, and Technology, Grants-in-Aid 14085204 and 14380145.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 256–266, 2005.

y lv , ( ) in n wo o olo y i ly n “ o-il ” in l i , o ni ion o o ol iv o yn i n o o olo y, n ( ) in in l y join o v n l v -o n wo w il i i in in n li ion jo , o ni ion o o ol o in o ni ion l n wo i ion. Al o-i on - o n wo n o iv o o ni ion o olo y n , in i ion o l ol n ili y x o n l i i l o i .

y , w i ol ny ni n o n i n l , v n i n i l o l o l ol n [1,]. How v , y no n ily o in yn i n o o olo y n y no o ly wo on - o n wo . W n in o n w on o l ol n : Fo iv n on in C on yn i n o n wo o olo y, w y y i (D T o o) l - ili in n C, i y wo o l - ili in y lon n wo n o no viol C. W n C l i no o olo i l n n, D T l - ili in l o i o C i j l - ili in l o i , n on o D T l - ili ion i n x-n ion o l - ili ion. No in o ion o on i ion C o li i ion o iv n x li i i o o o o l, in ny l o i wo l l o o li o olo i l n , non-ivi l l o i wo l n l o o x ly i n . W n o o o - o n wo , n o o n wo on ion on in C.

S l - ili in o n i l ion l o i v n x n iv ly i [3, , 5, 6]. A on i wo y n n W l [ ] lo o o wo . T y o o ini i i i l x l ion l o i o - o n wo , w i i in n l - ili in o n i l ion l o i . T i l o i ow v i y o in i w il y i onv in o l i i on ion; i i no D T l - ili in l o i o non- ivi l C.[1] T i i n n l iff n w n i n o ; o l o i onv o l i i on ion, v n i y i no i (lon y i o on i ion C).

l o i o o v n ; on o i o i i l in n non-ini i o o no n o in in o o o ol. L Δ n D xi n i o n wo , iv ly. T n in n n W l ’ l o i , v y o in in lo l v i l o $O($lo $(n\Delta^D))$ i , w il in o l o i , non-ini i o o no n o in in lo l v i l . n i on i o o n wo l n o y o in o ion n i . A o , o o n i in o ion o $O(n$ lo $n)$ i , w il i n i o $O($lo $(n^2\Delta^D))$ i , o $O(n^2$ lo $n)$ i l o in wo . Ano

[1] Note that mutual exclusion is guaranteed in the presence of arbitrary mobility, once the system is stabilized.

v n o o o n i i l lon o nnin . T i l o i i $O($lo $(n\Delta^D))$ i o n o i l ion, w i i $O(n$ lo $n)$ in wo , w il o i $O(n)$.

Fin lly o l o i n onv n i i ini i lly o n y $O(n)$, n nnin lon w i o n i i l on- v o ini nnin in $O(n^2)$- i , i n wo in i .

2 The Model

W on i y on i in o n o il in l wi wi l o ni- ion vi . W o l y y o o $V = \{p_1, p_2, ..., p_n\}$ $(n \geq$) wi ni i n i . Fo o p_i, l N_i o n i o o p_i n o ni wi . W v y o - ni ion nn l i i i ion l; $p_j \in N_i$ iff $p_i \in N_j$. N_i' o ll $p_i \in V$ n n wo $G = (V, E)$, w $(p_i, p_j) \in E$ i n only i $p_j \in N_i$. Fo $(p_i, p_j) \in E$, w i n o i iv (o o) no y $w_{i,j} (= w_{j,i})$. W y p_j i n p_k (o p_j ov p_k) i $w_{i,j} < w_{i,k}$. T w i o n n n on n wi , i n , o li ili y, o x l .

Sin o (i. ., in l) y n i lo ion , N_i y yn - i lly n n o y G o in ly. T w i $w_{i,j}$ y l o yn i lly n . W n ow v w i $w_{i,j}$ ni wi o lo o n li y, in o wi , w n i l $(w_{i,j}, p, q)$ ni w i in- , w $p =$ in$\{p_i, p_j\}$ n $q =$ x$\{p_i, p_j\}$. T ini nnin i ni ly in . W p_i now - o- v l o N_i n $w_{i,j}$ o ll $p_j \in N_i$. W N_i n v n w il o n i vi i in p_i.

S o y i i ion in o v l -n wo n i i ion on in o v . T n ll w o in -n wo n o i o i l o n on o in -n wo . A n o n wo o olo y y vi w o o join o l v o no o o o o (-)n wo . W on i V o ll o v n o i i in y , n o n o n wo i $|V| = n$ i l o nown o o . n wo i yn ono in n 1) lo l lo o o ow ,) i n o n δ on o ni ion l y w n wo n i o in o , 3) δ i nown o o , n) o in i o i n li i l . Wi o lo o n li y, w $\delta = 1$ (ni i).

A y i . (D T o o) l- ili in y wi o i ion P n yn i n wo on - ion on in C i ollowin on i ion i .

(1) onv n : Fo ny ini i l on ion n o ny o ion - in o i , y v n lly i P, lon n wo on ion n ollow C.

( ) S y: Fo ny ini i l on ion i P n o ny o- ion in o i , y in o i y P, lon n wo on ion n ollow C. I w on in “n i n i n o no n wo on ion o ” o C, D T l- ili in y wi o P n C i onv n ion l l- ili in y wi o P.

3 The Algorithm

T i ion n l n D T l- ili in l o i o i l- in o n lon ini nnin in n - o n wo . T l o i M, α n τ, w i ff o n n o n . S ion ow o v l i n o l o i o o , n n w n ly o n .

A o i in in o n i l ion o n ini i o . H n o n on o y o n ini i o . T l o i on i o wo , o n ini i o (Fi 1) n o ll o w o iv o n (Fi 3). H n wo x in in l o o ini i o .

W n o o n ini i o , ini i o n o n n n i in - nn o on o i n i o . A o i in o ion, o n i n o i vi i in o o . H n i i n y , i. ., y , ini i ion i . I o p_j iv o n o o p_i, o n in o n y in n (p_i, p_j) n o w i o n i o p_ℓ o p_j. No p_ℓ i l o i ion o (p_j, p_ℓ) o no y l in (i y o n). A w il , o n will y nnin T, w i ow v y no ini nnin . T o o nnin i lly i ov , n w v n lly o in ini nnin .

Sin o l o i i l , o n t n o y ll o ollowin o i l ion, o o w i y o ily in on i n o il n /o o olo y n .

- t ∈ {probe, echo}: T i ion o v l. I t i in n ow l (. oo), t = probe (. echo).
- t : A o o n oo , lon w i t i i l . T oo o t i ini i o o t (= t).
- t : T w i o in t .
- t : T o t, w o v l i ini i lly 0 n i in n y on w n v t ov .
- t : T i n i o t i n y ini i o l o n in $\{0, 1, \ldots, M-1\}$.[2]

[2] Ideally, M should be selected so that more than M tokens never exist in the network at a time. However, this assumption is removable and M can be set any value ≥ 2, at the expense of the convergence time.

```
**Variables of an initiator** $p_i$ **:**
  $m$ : **integer initially** 0;        — *Token identifier.*

**Code for an initiator** $p_i$ **:**
while {        — Initiate new circulation by generating a new token.
    try {
        wait;        — Wait for a token to arrive (with timeout). Token is handled by the token thread.
    } catch (Signal) {        — A token visits this process. This event is notified by the token thread.
        ;        — Do nothing. Wait for next arrival of a token.
    } catch (TimeoutException) {
        $m := (m+1) \bmod M$;        — Assign new token identifier.
        $t := \langle \mathsf{probe}, \emptyset, \emptyset, 0, m, p_i, \perp, \infty \rangle$
        Let $p_k$ be a process in $N_i$ such that $w_{i,k}$ is the smallest;
        send $t$ to $p_k$;
    }
}
```

Fig. 1. Initiator thread: the code for an initiator

```
macro UpdateToken ≡
{
        // IMPROVE THE SPANNING TREE.
        if ($t.alte \neq \perp$) {        — There is an edge to improve the spanning tree.
                $t.tree := t.tree \cup \{t.alte\}$;        — Temporarily t.tree has a cycle.
                Find an edge $e$ in $t.tree$ such that
                        $t.tree - \{e\}$ is a spanning tree and its weight is the smallest;
                $t.tree := t.tree - \{e\}$;
                Delete from $t.wgt$ the weight of edge $e$;
                Add into $t.wgt$ edge $t.alte$ with weight $t.altw$;
        }
        // REFRESH THE TOKEN FOR THE NEXT ROUND OF TOKEN CIRCULATION.
        if ($p_i = t.ini$) {
                $m := (m+1) \bmod M$;
                $t.id := m$;        — Assign new token identifier.
                $t.age := 0$;        — Reset token age.
        }        — If $p_i$ (= the root of t.tree) is not the initiator of t, t.age and t.id are unchanged.
        $t := \langle \mathsf{probe}, t.tree, t.wgt, t.age, t.id, t.ini, \perp, \infty \rangle$;
                — Assign new token identifier and reset the token age.
}

macro FindCandidate ≡
{
        if ($t.alte = \perp$)
                Let $T$ be $t.tree$;
        else
                Let $T$ be the spanning tree with the smallest weight among subgraphs of $t.tree \cup t.alte$;
        Let $T'$ be the spanning tree with the smallest weight among subgraphs of
           $T \cup \{(p_i, p_\ell) : p_\ell \in N_i - TreeNeighbors(t)\}$;
        if (weight of $T'$ < weight of $T$){
                Let $p_\ell$ be a process that yields $T'$;
                return $p_\ell$;
        }
        return $\perp$;
}
```

Fig. 2. Macro definitions for token thread

- $t.$: T i n i o ini i o .
- $t.$: T n n i e o i ovin w i o $t.$.
- $t.$: T w i o $t.$.

```
When a token $t$ arrives at $p_i$ from $p_j$ :
1:  $t :=$ receive;
2:  $t.age := t.age + 1$;    — Increment the age by one.
3:  if ($p_i$ is an initiator) $\wedge ((t.ini > p_i) \vee ((t.ini = p_i) \wedge (t.id \neq m)))$ {
4:      Discard $t$;    — Discard the token based on priority.
5:  } else if $(\neg Alive(t))$    — The token is too old to alive.
6:      Discard $t$.
7:  } else {
8:      if ($p_i$ is an initiator)
9:          notify;    — Restart timeout timer of the initiator thread.
10:     // EXTEND THE SPANING TREE IF $p_i$ IS NOT YET INCLUDED.
11:     if $(t.type =$ probe$) \wedge ((p_j, p_i) \notin t.tree)${    — This is the first visit to $p_i$.
12:         $t.tree := t.tree \cup \{(p_j, p_i)\}$;  $t.wgt := t.wgt \cup \{(p_j, p_i, w_i(p_j))\}$;    — Extend the tree.
13:         $t.alte = \perp$;  $t.altw = \infty$;    — Reset the candidate for improving the spanning tree.
14:         if $(t.ini = p_i)$    — The token visits initiator $p_i$ which was disconnected from $t.tree$.
15:             $t := \langle$probe$, \emptyset, 0, m, p_i, \perp, \perp, \perp, \infty\rangle$    — Reset $t$ and start a new round.
16:     }
17:     // CHECK IF NETWORK TOPOLOGY AND EDGE WEIGHTS ARE CHANGED.
18:     for each $p_k \in (Children(t, p_i) - N_i)$    — A child $p_k$ is disconnected from $p_i$.
19:         Delete a subtree rooted at $p_k$ from $t.tree$, and update $t.wgt$ accordingly;
20:     if $(Parent(t, p_i) \notin N_i)$    — Parent process is disconnected from $p_i$.
21:         $t.tree :=$ a subtree of $t.tree$ rooted at $p_k$, and update $t.wgt$ accordingly;
22:     for each $p_k \in TreeNeighbors(t, p_i)$ {
23:         if (the weight of $(p_k, p_i)$ in $t.wgt$ is different from $w_i(p_k)$)
24:             Update the weight of $(p_k, p_i)$ in $t.wgt$ to be $w_i(p_k)$;
25:     }
26:     // FIND A CANDIDATE EDGE TO IMPROVE THE SPANNING TREE.
27:     if $(N_i - Procs(t) = \emptyset)$ {
28:         $p_\ell := FindCandidate$;    — $p_\ell$ is in $N_i - TreeNeighbors(t)$ or equals $\perp$.
29:         if $(p_\ell \neq \perp)$ {    — Better candidate is found.
30:             $t.alte = (p_i, p_\ell)$;  $t.altw = w_i(p_\ell)$;
31:         }
32:     }
33:     // FIND A DESTINATION OF THE TOKEN.
34:     if $(N_i - Procs(t) \neq \emptyset)$ {    — There is a neighbor process not in the spanning tree.
35:         $t.type :=$ probe;  $p_k :=$ a process such that $w_i(p_k)$ is the smallest among $p_k \in N_i - Procs(t)$;
36:     } else if ($p_i$ is a leaf process of $t.tree$){
37:         $t.type :=$ echo;  $p_k := Parent(t, p_i)$;    — $t$ will be sent back to the parent ($p_k = p_j$).
38:     } else {
39:         if $(t.type =$ probe$)$ {    — $t$ was received from the parent.
40:             $p_k := FirstChild(t, p_i)$;    — $t$ will be sent to the first child.
41:         } else {    — $t$ was received from a child ($t.type =$ echo).
42:             $p_k := NextChild(t, p_i, p_j)$;    — $t$ will be sent to the next child.
43:         }
44:         if $(p_k \neq \perp)$ {    — There is a child to forward.
45:             $t.type :=$ probe;    — $t$ will be sent as a probe token to the child.
46:         } else {    — No more child to forward $t$ ($p_k = \perp$).
47:             if $(p_i = Root(t))$ {    — The end of a round. $p_i$ may not be $t.ini$. (See lines 20–21.)
48:               $UpdateToken$;    — Improve the spanning tree, and prepare for the next round.
49:               $p_k := FirstChild(t, p_i)$;    — $t$ will be sent to the first child.
50:             } else {
51:               $p_k := Parent(t, p_i)$;    — $t$ will be sent back to the parent.
52:             }
53:         }
54:     }
55:     // FORWARD THE TOKEN.
56:     send $t$ to $p_k$;
57: }
```

Fig. 3. Token thread: the code for a process who receives a token

T l o i o ollowin n ion :

– (t): T o o t.. .

– (t): T oo o t.. .

- (t, p_i) : T n o p_i in t.. . I p_i i oo n $(t, p_i) = \bot$.
- (t, p_i): T o il n o p_i in t.. .
- (t, p_i): T ll il o p_i in t.. . I p_i no il n in t.. , n $(t, p_i) = \bot$.
- (t, p_i, p_j): T ll il on o o p_i in t.. l n p_j.
- (t, p_i): T o n i o o p_i in t.. .
- (t): T i n **true** i n only i t. $\leq \alpha$.

3.1 Initiator Thread

T ol o ini i o o n w o n n o i w n i o no n wi in τ i n i o lo . W will ow in S ion i o v l $\tau = 6(n-1)$ i i n o n i o i li no o n in i l . T ini i o in in lo l in v i l $m \in \{0, \ldots, M-1\}$ o o n i n i . W n i o o , ini i o in n m y on (o lo M), n n w o n yin ini i l . Ini i o n n i o ll n i o o i i l ion. T ini i o o no in , lon i o o no o . No n iv l o o n i y o n on n ly x in ini i o , n i no i o ini i o y in o ni ion i i iv **notify** n **wait**.

3.2 Token Thread

T o n i on i l o ollowin o n ion (A)-(D):

(A) Token Elimination: A o n t on in o v l on no in - ion, nl i i li in wi i ion o i in n n , in 1) i t. x on n α, o) t iv n ini i o p w o io i y. A o 1), α o l l o o i l ion lw y ni l n α v l (w n n wo i i). W i l ly $\alpha =$ $(n-1)$ in S ion . A o), io i y o t n y $(t.$. $, t.$ $)$ i l i lly o wi o p n y (p, m), w m i n v ion n .

(B) Token Circulation: A o n t i i l in - nn . Ini i lly t.. i y, n i ow i vi i n w o . A lon n wo i i , n i l n ion o i l ion i i o w , l o i on i o . In , n - l o i i x , n t will n o i ini i o wi nnin t.. .

In on , t ly o v l o nnin t.. , w il n v in o o i w i . D in v l, t in o ion o i ov t.. , w l v i i in I ( ) n (D).

ll ow v n wo y no i . W n o n no p_i, o o n y in on i n wi i n n i o N_i n /o w i $w_{i,j}$. W ov on i n y i ly y in on i - n wi N_i n $w_{i,j}$ n y i in . Fo x l , i w i ov $p_k \notin N_i$ o o $(p_i, p_k) \in t.$, n $t.$ y ovin (p_i, p_k), n n ov ll onn o on n in $t.$ o no on in p_i. W n $t.$ o in ly.

No oo o $t.$ n i n o p_i l v n wo . v n o, o n on in o v l, lon i i l n α. I t n o n i ini i o $t.$, n ini i o ini i n w o n , o wi , t i li in y .

(C) Searching for Candidate Edge: T nnin $t.$ o t i no lw y ini w n n . In o n o v l , w y o i ov $t.$ y l in n e' in $t.$ wi no $t.$ n o n in l v l, o v n lly in ini nnin . H e' i no wi l w i in ni y l in $t.$ $\cup\{t.$ $\}$. No i i y o in in $t.$, in i i o vio o no p_i o i i n e' o i l e in i n on i .

(D) Initiating Next Traversal: W n o n t n o ini i o , ini i o i ov nnin $t.$ y l in e' in $t.$ wi $t.$, n ini i li t y in n in v ion n $t.$ n in $t.$ o 0.

4 Correctness and Performance

on i n i y ini i l on ion o ini i o in n wo . S o will no n i n il n no o o will no o n ini i o . W ow v n lly n x ly on o n i i l on o lon nin , lon n wo o olo y n i y o on i ion iv n low. B o li i , w only i n io o oo . ll α i li i o o n, n τ i i o in v l o o n ion.

Sin $t.$ o in o n o in o ion i y o n t, w o vio ly v :

Theorem 1. O $(n \text{lo} \ n)$ □

l o ow n wo o olo y n (lon on i ion i in S ion ol), w n ow ollowin :

Lemma 1. □

L p^* ll ini i o . ll To n li in ion oli y o o n . Sin p^* lw y li in o n n y o ini i o , p^* v n- lly ini i o n. I o n on o n ini i y p^*, ll on wi $p^*.m = t.$ li in . H n w n wi o lo o n li y i in l o n t in n wo ini i y p^* $p^*.m = t.$, n t i n v li in , nl i α o i o i o p^* x i τ.

S o **the network is static**. Sin in n x ly on o n wi $p^*.m = t.$ y ov o v ion n $(n-1)$ i ov n o $t..$ lw y $t..$ ini , w o vio ly v :

Theorem 2. t $t. = p^*.m$ p^* $t..$ $t..$ $(n-1)$ $(n-1)^2$ t □

N x o **the network topology may change only by edge augmentations**. Sin n wo will no i ion , w v :

Theorem 3. $\alpha = (n-1)$ $\tau = (n-1)$ □

By o inin i wi T o , w n on l t v n lly i ini nnin in $(n-1)^2$ i .

Fin lly, w on i **general network topology changes**. L i w nno v n i o n o y nnin o o i l ll o in n l, in n wo y i ion , n w l o i n n v y, n in on on i ion on o i l o olo y n .

T on in C_1 on n wo n w on i i ollowin : 1) no wo i onn ion o wi in $6(n-1)$ i , n) n wo i no i ion .

Theorem 4. $\alpha = 6(n-1)$ $\tau = 1 (n-1)$ C_1 $6(n-1)$

P oo (S). on i ollowin wo n io: S o i i ly i ini i ion y n ini i o p^*, o n t l v p^* o il q, n e w n p^* n q i onn i i ly i ov . T o n i l ion n on in n il t n e i onn ion l o n o i i l ion w n i i o o n o p^* o q.

T n q on i l ion o t o lly oo . No i i l ion y i $(n-1)$. Sin n wo i no i ion n no (x e) i onn y $6(n-1)$ i , t lw y n o p^* y i $(n-1)$. I i wo no in y o t no vi i y . I i ow v l i i l ion will vi i ll o in no x e ill will no i onn n x $(n-1)$ i , n $(n-1)$ i ion l i n o o l i l ion.

H n ov i i $6(n-1)$, n $\tau = 1$ $(n-1)$ i i n o o ini i o no o i n w o n y i o . □

L C_2 on i ion i C_1 x ini i in v l w n wo i onn ion i o $(n-1)$.

Theorem 5. $\alpha = 6(n-1)$ $\tau = 1$ $(n-1)$ C_2 ( ) □

i in in n wo i ion. S o n wo i i ion in o wo -n wo , o w i on in n ini i o . T n o n will i l o v in o -n wo , o l o i nno ili i i ion. T i ow on i ion ( ) o C_1 n C_2 in vi l .

5 Conclusion

In i , w o o on o yn i on ion ol n (D T) l- ili ion o i l wo o i i l o i o yn i -o n wo . W n o o ini i n l D T l- ili in o n i l ion l o i n o n wo o olo y n on in . W l o i n n n on w w n o n wo o olo y n . I n wo i l , n x ly on o n i i- l lon ini nnin in $O(n)$ i . vio ly i n no in i n wo o olo y i ly n . B i n n in l o n i l ion v n i i onn ion o (lon i o n no o n).

In i , w v p_i now - o- v l o N_i n $w_{i,j}$ o ll $p_j \in N_i$. T i ion ow v i no o li i ; in l y , n o n i o i ly il i n i o i no lon wi in n i ion n . W ow v li v w n o i y o l o i o i wo n ov o li i ion y ly o in in ion o o n n il o n in .

A ll n in o l w l v wo i o x n ni ion o o n i l ion o n o o n i l ion l o i , v n i n wo i i ion , n o o o D T l- ili in o n i l ion l o i on n w ni ion.

References

1. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM **17** (1974) 643–644
2. Dolev, S.: Self-stabilization. The MIT Press (2000)

3. Datta, A.K., Johnen, C., Petit, F., Villan, V.: Self-stabilizing depth first token circulation in arbitrary rooted networks. In: Proceedings of the 5th International Colloquium on Structual Information and Communication Complexity (SIRROCO). (1998) 119–131
4. Malpani, N., Vaidya, N.H., Welch, J.L.: Distributed token circulation on mobile ad hoc networks. In: Proceedings of the 9th International Conference on Network Protocols (ICNP). (2001)
5. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self stabilizing mutual exclusion. In: Proceedings of the 9th ACM Symposium on Principles of Distributed Computing, ACM (1990) 119–131
6. Dolev, S., Schiller, E., Welch, J.: Random walk for self-stabilizing group communication in ad-hoc networks. In: the 21st IEEE Symposium on Reliable Distributed Systems (SRDS). (2002) 70–79
7. Chen, Y., Welch, J.L.: Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks. In: Proceedings of the Sixth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM). (2002)

Snap-Stabilizing Depth-First Search on Arbitrary Networks[⋆]

Al in o ni , S n D vi , F n P i , n Vin n Vill in

LaRIA, CNRS FRE 2733,
Université de Picardie Jules Verne, Amiens (France)

Abstract. A *snap-stabilizing protocol*, starting from any arbitrary initial configuration, always behaves according to its specification. In this paper, we present a snap-stabilizing depth-first search wave protocol for arbitrary rooted networks. In this protocol, a wave of computation is initiated by the root. In this wave, all the processors are sequentially visited in depth-first search order. After the end of the visit, the root eventually detects the termination of the process. Furthermore, our protocol is proven assuming an unfair daemon, i.e., assuming the weakest scheduling assumption.

keywords: Distributed systems, fault-tolerance, stabilization, depth-first search.

1 Introduction

A i i y i n wo w o o x lo l o ion o in o i n o i n i o . In y , *wave protocol* [1] i o o ol w l on o o (ll *initiator*) ini i y l o o ion (l o ll *wave*). A n in o y l , ini i o i l o in l n in on o in l on ion n i o y o y l ' o ion.

In n i y oo n wo , D -Fi S (*DFS*) w v i ini i y oo . In i w v , ll o o n i lly vi i in - o . T i ny li ion in i i y . Fo x - l , ol ion o i o l n o olv l x l ion, nnin o ion, on in o in , o in , o yn oni ion.

T on o *self-stabilization* [ ] i o n l ni o i n y o ol i y n i n l . A l- ili in y , l o ini i l o o o n ini i ly in lin , i n- o onv o in n vio in ni i . *Snap-stabilization* w in o in [3]. A *snap-stabilizing* o o ol n i lw y v o in o i i ion. In o wo , n - ili in o o ol i l o

[⋆] The full version is available at www.laria.u-picardie.fr/∼devismes/tr2004-9.ps

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 267–282, 2005.

l- ili in o o ol w i ili in 0 . vio ly, *snap-stabilizing* o o ol i o i l in ili ion i .

T xi v l (non l- ili in) i i l o i olvin i o l , . ., [,5]. In o l- ili in y , il n l o i (i. ., in i l o i , y onv o x oin) w i o *DFS* nnin o i y oo n wo i iv n in [6]. S v l l- ili in (no n - ili in) w v l o i on *depth-first token circulation* (*DFTC*) v n o o o i y oo n wo . T on w o o in [ ]. I i $O(\text{lo} (N) + \text{lo} (\Delta))$ i o o w N i n o o o n Δ o n wo . S n ly, v l o l- ili in o o ol w o o , . ., [,9,10]. All o o y i n o $O(\text{lo} (\Delta))$ i o o . T l o i o o in [ ] off o l xi y. All ov ol ion [, ,9,10] v ili ion i in $O(N \times D)$ o n w D i i o n wo . T ol ion o o in [11] ili in $O(N)$ o n in $O(\text{lo} (N) + \text{lo} (\Delta))$ i o o . Un il now, i i ol ion (o i y n wo) in o -off w n i n o l xi i . T o n o ov l o i i ov n in (w ly) i on. o ly in , on i on i n v y w i i o v n o o ol o v x , n i n n on nno v n o v o o o x n n l ion. T n - ili in *DFTC* n o o in [1] o n wo . In i y n wo , ovi in n - ili in v ion o ny (n i l- no n -) o o ol i iv n in [13]. vio ly, o inin i o o ol wi ny *DFTC* l o i , w o in n - ili in *DFTC* l o i o i y n wo . How v , l in o o ol wo in w ly i on only. In , i n n in ni n o n o , in n n ly o o n o . T o , n o w v nno o n .

In i , w n n - ili in - w v o o ol o i y oo n wo in n n i on, i. ., in w lin ion. In , in o o o ol, x ion o *DFS* w v i o n y $O(N^2)$. T o o ol o no ny - o nnin i i n i i on o o . T n - ili in o y n oon o o ol i ini i y oo , v y o o o n wo will vi i in *DFS* o . A n o vi i , oo v n lly in ion o o .

T o i o ni ollow : in S ion , w i i i y n o l in w i o o o ol i w i n. Mo ov , in ion, w iv o l n o D -Fi S W v P o o ol olv in i . In S ion 3, w n D -Fi S W v P o o ol. In ollowin ion (S ion), w iv oo o n - ili ion o o o ol n o o l xi y l . Fin lly, w on l in in S ion 5.

2 Preliminaries

W on i n n i onn $G = (V, E)$ w V i o ($|V| = N$) n E i o . W on i n wo w i n ,i. ., on o o ,w i in i i l o o ll *root*. W no oo o o y . A o ni ion lin (p, q) xi i n only i p n q ni o . v y o o p n i in i ll i lin . To i li y n ion, w o lin (p, q) o o o p y q. W l l o p, o in $Neig_p$[1], lo lly o y $\prec_p$. W $Neig_p$ i on n , $Neig_p$ i own n in o y . Mo ov , w n wo i i n i , i. ., v y o o x ly on i n i y w i i ni in n wo . W no i n i y o o o p y Id_p. W Id_p i on n . Id_p i l o own n in o y .

In o ion o l w , o o x o x . W on i lo l o y o l o o ni ion. T o o v y o o on i o o (n o , o v i l) n ni o ion . A o o n only w i o i own v i l , n i own v i l n v i l own y ni o in o o . ion i on i ollow :

$$< label > :: < guard > \to < statement > .$$

T o n ion in o o p i ool n x ion involvin v i l o p n i ni o . T n o n ion o p on o o v i l o p. An ion n x only i i i i . W ion o i lly x , nin , v l ion o n x ion o o on in n o n ion, i x , on in on o i .

T o o o i n y v l o i v i l . T o y i o o o ll o o ($\in V$). W will o o o o n y ( ) n ( ) , iv ly. L $\mathcal{C}$, o ll o i l on ion o y . An ion A i i o in $\gamma \in \mathcal{C}$ p i o A i p in γ. A o o p i i o in γ ($\gamma \in \mathcal{C}$) i xi n n l ion A in o o p in γ.

L i i o o ol $\mathcal{P}$ oll ion o in y n i ion l ion no y $\mapsto$, on $\mathcal{C}$. A o o o ol $\mathcal{P}$ i n o on ion $e = (\gamma_0,\gamma_1,...,\gamma_i,\gamma_{i+1},...)$, o $i \geq 0$, $\gamma_i \mapsto \gamma_{i+1}$ (ll) i γ_{i+1} xi , l γ_i i in l on ion. n n i i ni (n no ion o $\mathcal{P}$ i n l in in l on ion) o in ni . All o ion on i in i o xi l. T o ll o i l o ion o $\mathcal{P}$ i no $\mathcal{E}$.

[1] Every variable or constant X of a processor p will be noted X_p.

W on i ny o o p x in o - ion $\gamma_i \mapsto \gamma_{i+1}$ i p w in γ_i n no n l in γ_{i+1}, i no x ny ion w n wo on ion . (T i lin ion n ollowin i ion: l on n i o o p n i w n γ_i n γ_{i+1}, n i n ff iv ly o ll ion o p l .)

In o o ion, , ll o o o i ion . T n, o o o o n y . Fin lly, “ l ” o o x on o o o i ion . T xi v l in o . H , w , i. ., in o - ion , i on o o o o n l , on oo l on (o i ly o) o n l o o o x n ion. F o , on n , i. ., i o o p i on in o ly n l , p will v n lly o n y on o x n ion. I on i , i n o v v n o o o x n ion x i i i only n l o o .

In o o o i o l xi y, w ni ion o [1]. T i ni ion x ion o low o o in ny o ion. iv n o ion e ($e \in \mathcal{E}$), o e (l ll i e') i ini l x o e on inin x ion o on ion (n ion o o o ol o i lin ion) o v y n l o o o on ion. L e'' x o e $e = e'e''$. T o e i o n o e'', n o on. W y o n i i i i on i o ni n o .

T on o *snap-stabilization* w in o in [3]. In i , w i i on o w v o o ol only.

Definition 1 (Snap-stabilization for Wave Protocols). $\mathcal{T}$ $\mathcal{SP_T}$ $\mathcal{T}$ $\mathcal{P}$ $\mathcal{SP_T}$

(initiator) $\mathcal{P}$ *(initialization action)* $\mathcal{P}$ *initialization action* $\mathcal{SP_T}$

Theorem 1. $\mathcal{T}$ $\mathcal{SP_T}$ $\mathcal{T}$ $\mathcal{P}$ $\mathcal{P}$ $\mathcal{SP_T}$ $\mathcal{P}$ $\mathcal{P}$ $\mathcal{SP_T}$

L e n x ion o $\mathcal{P}$ in n n i on. By ion, v y o n o e i ni . T n, v y o n o e i ni , n l o o (in e) x n ion (i i lin ion o n ion o $\mathcal{P}$) in ni n o . In i l , v y on in o ly n l o o x n ion o $\mathcal{P}$ in ni n o . So, e i l o n x ion o

$\mathcal{P}$ in w ly i on. Sin $\mathcal{P}$ i l- ili in o $\mathcal{SP_T}$ in w ly i on, e ili o $\mathcal{SP_T}$. H n , $\mathcal{P}$ i l- ili in o $\mathcal{SP_T}$ v n i on i n i . □

B o ivin i-ion o D -Fi S W v P o o ol, w o o o ni ion .

Definition 2 (Path). p_1, p_k ($\forall i \in [1$ $k]$, $p_i \in V$) $G = (V, E)$ $\forall i \in [1$ $k-1]$, $(p_i, p_{i+1}) \in E$ p_1, p_k $\forall i, j$ $1 \leq i < j \leq k$, $p_i \neq p_j$ p_1 p_k

Definition 3 (First Path). (p_1), p_i, p_k, l_1, l_i, l_{k-1} ($word(P)$) $\forall i \in [1$ $k-1]$, p_i p_{i+1} l_i p_i $\prec_{lex}$ p, p $\prec_{lex}$ p ($fp(p)$)

U in i no ion, w n n *first DFS order*:

Definition 4 (First DFS Order). $p, q \in V$ $p \neq q$ $\prec_{dfs}$ $p \prec_{dfs} q$ $word(fp(p))$ $\prec_{lex}$ $word(fp(q))$

Specification 1 (fDFS Wave). *Visited* $\in \mathcal{E}$ (DFS).

$\forall p \in V$, p

In o o ov o o o ol i n - ili in o S i -ion 1, w ow v y x ion o o o ol i o on i ion :

1. v n lly ini i DFS w v .
. F o ny on ion w ini i DFS w v , y lw y i S i ion 1.

3 Algorithm

In i ion, w n *DFS* w v o o ol o Al o i *snap$\mathcal{DFS}$* (Al o i 1. n .). W n no l vio . W n x l in o o o o ion.

Algorithm 1. Al o i $\mathit{snap}\mathcal{DFS}$ o $p =$

Input: $Neig_p$: set of neighbors (locally ordered); Id_p: identity of p;
Constant: $Par_p = \bot$;
Variables: $S_p \in Neig_p \cup \{idle, done\}$; $Visited_p$: set of identities;
Macros:
$Next_p = (q = \min_{\prec_p}\{q' \in Neig_p :: (Id_{q'} \notin Visited_p)\})$ **if** q **exists**, $done$ **otherwise**;
$ChildVisited_p = Visited_{S_p}$ **if** $(S_p \notin \{idle, done\})$, $\emptyset$ **otherwise**;
Predicates:
$Forward(p) \equiv (S_p = idle)$
$Backward(p) \equiv (\exists q \in Neig_p :: (S_p = q) \wedge (Par_q = p) \wedge (S_q = done))$
$Clean(p) \equiv (S_p = done)$
$SetError(p) \equiv (S_p \neq idle) \wedge [(Id_p \notin Visited_p) \vee (\exists q \in Neig_p :: (S_p = q) \wedge (Id_q \in Visited_p))]$
$Error(p) \equiv SetError(p)$
$ChildError(p) \equiv (\exists q \in Neig_p :: (S_p = q) \wedge (Par_q = p) \wedge (S_q \neq idle) \wedge \neg(Visited_p \subsetneq Visited_q))$
$LockedF(p) \equiv (\exists q \in Neig_p :: (S_q \neq idle))$
$LockedB(p) \equiv [\exists q \in Neig_p :: (Id_q \notin ChildVisited_p) \wedge (S_q \neq idle)] \vee Error(p) \vee ChildError(p)$
Actions:
$F :: Forward(p) \wedge \neg LockedF(p) \rightarrow Visited_p := \{Id_p\};\ S_p := Next_p;$
$B :: Backward(p) \wedge \neg LockedB(p) \rightarrow Visited_p := ChildVisited_p;\ S_p := Next_p;$
$C :: Clean(p) \vee Error(p) \rightarrow S_p := idle;$

F o no l on ion, w i in i wo in o o o ol: w o o ol vi i ll o o in *first DFS order* n w i l n o o oo i v n lly y o ini i n w . T o wo in ll l. In i no l vio , Al o i $\mathit{snap}\mathcal{DFS}$ v i l o o o p:

1. S_p i n o p in vi i in , i. ., i xi $q \in Neig_p$ $S_p = q$, n q (. p) i i o o p (. o q),
. $Visited_p$ i o o o w i v n vi i in vi i in ,
3. Par_p i n o o w i oin o p on o i o in vi i in (no o , Par_r i on n $\bot$).

on i on ion w $[(S_r = idle) \wedge (\forall p \in Neig_r\ , S_p = idle) \wedge (\forall q \in V \backslash\ (Neig_r \cup \{\ \}), S_q \in \{idle, done\})]$. W o on ion . In on ion , v y o o $q \neq$ $S_q = done$ i n l o o i l nin (P i $Clean(p)$). P o o q o i l nin y x in A ion C, i. ., i i n $idle$ o S_q. Mo ov , in i on ion, oo ( ) i n l o ini i vi i in (A ion F). P o o n ini i vi i in y ini i li in $Visited_r$ wi i i n i y (Id_r) n oin in o (wi S_r) i ini l n i o in lo l o $\prec_r$ (M o $Next_r$). In wo , v y o o q, $S_q = done$, x i l nin , , i only n l o o n ini i vi i in . F o i oin on, i only o o .

W n o o $p \neq$ $S_p = idle$ i oin o wi S_q y n i o in o o q, n p w i n il ll i n i o p', $S_{p'} = done$ n $Id_{p'} \notin PredVisited_p$ (, $Visited_q$), x i l nin . A , p n x A ion F. T n, p l o i n q wi Par_p n i n $PredVisited_p \cup \{Id_p\}$ (, $Visited_q \cup \{Id_p\}$) o $Visited_p$. In o lly, $Visited$ o l vi i o o on in i n i i o ll vi i o o . Fin lly, p oo n w o , i ny. Fo i li , wo o i l (M o $Next_p$):

1. $\forall\, p' \in Neig_p$, $Id_{p'} \in Visited_p$, i. ., ll n i o o p v n vi i ; vi i in o p i now in , o, S_p i o *done*,

. o wi , p oo o ini l o o y $\prec_p$ in $\{p' :: p' \in Neig_p \wedge Id_{p'} \notin Visited_p\}$ n p i now in vi i in .

In o , p i now on i .

W n q i o o p n $S_q = done$, p now vi i in o q i in . T , p on in vi i in in no n i o in o o w i i ill no vi i , i ny: p x A ion B n i i n $ChildVisited_p$ o $Visited_p$. H n , i now x ly w i o o v n vi i n i n i n no o , i ny, in A ion F (M o $Next_p$). P o o q i , now, n l o x i l nin (A ion C).

Fin lly, $S_r = done$ n vi i in i in o ll o o n o, n x i l nin . T , y v n lly in.

Fi , o no l vio , w n , i $p \neq$ i in vi i in n vi i in o p i ill no in , n p v o n i n i wi i v i l Par_p, i. ., o o $p \neq$ i y: $(S_p \notin \{idle, done\}) \Rightarrow (\exists q \in Neig_p :: S_q = p \wedge Par_p = q)$. T i $NoRealParent(p)$ llow o in i i on i ion i no i y p. T n, in no l vio , o o in in o i on v l o i $Visited$ n o i o , i ny. T , in ny on ion, p ollowin on i ion (A ion F):

1. $(S_p \neq idle) \Rightarrow (Id_p \in Visited_p)$ w n p i vi i , i in l i i n i y in i $Visited$.

. $(S_p \in Neig_p) \Rightarrow (Id_{S_p} \notin Visited_p)$, i. ., p no oin o vio ly vi i o o .

3. $((p \neq \ ) \wedge (S_p \neq idle) \wedge (\exists q \in Neig_p :: (S_q = p) \wedge (Par_p = q))) \Rightarrow (Visited_q \subsetneq Visited_p)$ w il $p \neq$ i in vi i in , $Visited_p$ i ly in l $Visited$ o i n .

Algorithm 2. Al o i *snap$\mathcal{DFS}$* o $p \neq$

```
Input: $Neig_p$: set of neighbors (locally ordered); $Id_p$: identity of $p$;
Variables: $S_p \in Neig_p \cup \{idle, done\}$; $Visited_p$: set of identities; $Par_p \in Neig_p$;
Macros:
$Next_p$ = ($q = \min_{\prec_p}\{q' \in Neig_p :: (Id_{q'} \notin Visited_p)\}$) if $q$ exists,
           $done$ otherwise;
$Pred_p$ = $\{q \in Neig_p :: (S_q = p)\}$;
$PredVisited_p$ = $Visited_q$ if $(\exists!\ q \in Neig_p :: (S_q = p))$, $\emptyset$ otherwise;
$ChildVisited_p$ = $Visited_{S_p}$ if $(S_p \notin \{idle, done\})$, $\emptyset$ otherwise;
Predicates:
$Forward(p)$ ≡ $(S_p = idle) \wedge (\exists q \in Neig_p :: (S_q = p))$
$Backward(p)$ ≡ $(\exists q \in Neig_p :: (S_p = q) \wedge (Par_q = p) \wedge (S_q = done))$
$Clean(p)$ ≡ $(S_p = done) \wedge (S_{Par_p} \neq p)$
$NoRealParent(p)$ ≡ $(S_p \notin \{idle, done\}) \wedge \neg(\exists q \in Neig_p :: (S_q = p) \wedge (Par_p = q))$
$SetError(p)$ ≡ $(S_p \neq idle) \wedge [(Id_p \notin Visited_p)$
    $\vee (\exists q \in Neig_p :: (S_p = q) \wedge (Id_q \in Visited_p))$
    $\vee (\exists q \in Neig_p :: (S_q = p) \wedge (Par_p = q) \wedge \neg(Visited_q \subsetneq Visited_p))]$
$Error(p)$ ≡ $NoRealParent(p) \vee SetError(p)$
$ChildError(p)$ ≡ $(\exists q \in Neig_p :: (S_p = q) \wedge (Par_q = p) \wedge (S_q \neq idle)$
    $\wedge \neg(Visited_p \subsetneq Visited_q))$
$LockedF(p)$ ≡ $(|Pred_p| \neq 1) \vee (\exists q \in Neig_p :: (Id_q \notin PredVisited_p) \wedge (S_q \neq idle))$
    $\vee (Id_p \in PredVisited_p)$
$LockedB(p)$ ≡ $(|Pred_p| \neq 1) \vee (\exists q \in Neig_p :: (Id_q \notin ChildVisited_p) \wedge (S_q \neq idle))$
    $\vee\ Error(p) \vee ChildError(p)$
Actions:
$F$ :: $Forward(p) \wedge \neg LockedF(p)$ → $Visited_p := PredVisited_p \cup \{Id_p\}$;
                                             $S_p := Next_p$; $Par_p := (q \in Pred_p)$;
$B$ :: $Backward(p) \wedge \neg LockedB(p)$ → $Visited_p := ChildVisited_p$; $S_p := Next_p$;
$C$ :: $Clean(p) \vee Error(p)$ → $S_p := idle$;
```

I on o on i ion i no i y p, p i $SetError(p)$. So, Al o i *snap$\mathcal{DFS}$* i p i in n no l , i. ., $(((p \neq\ ) \wedge NoRealParent(p)) \vee SetError(p))$ wi i $Error(p)$. In o , w ll o o p i yin $Error(p)$. I p i n no l o o , n w o p n ll o o vi i o p. W i ly o p y in S_p o $idle$ (A ion C). So, i , o p x A ion C, xi o o q $(S_p = q \wedge Par_q = p \wedge S_q \notin \{idle, done\} \wedge \neg Error(q))$, n p x A ion C, q o n no l o o oo (l in p). T o ion o n il vi i in o p i o l ly o . How v , in o ion , vii in o p n o y x ion o A ion F n B. B , w n *Visited* o l o o o vi i in ow y x ion o A ion F n B n l o o o vi i in n only x n o ion in o o w i no in i $Visited$. T , vi i in o n no l o o nno n in ni ly. H n , w will l vi i in o n no l o o will v n lly o .

Fin lly, w o on iff n w y o o (o low own) o ion o on o vio . A ion F n B llow o o p o x i vi i in . How v , y o vin i n o i n i o , p n o y vio n o : i o l o i $LockedF(p)$ n $LockedB(p)$ in A ion F n B, iv ly. A o o p i (i. ., p nno x A ion B o A ion F) w n i i l on o v ollowin on i ion :

1. p v l o .
. p i n no l o o .
3. p o q $((S_q{\neq}idle){\wedge}(Par_q{=}p){\wedge}\neg(Visited_p{\subsetneq}Visited_q))$, i. ., q i no l.
. p $(S_p = idle)$ i i n o y q Id_p i in $Visited_q$, i. ., q i no l.
5. o non-vi i n i o q o p no l n , i. ., $S_q \neq idle$ (l o in no l vio).

4 Correctness and Complexity Analysis

4.1 Basic Definitions and Properties

L $p \in V$. p i i n only i $(Clean(p) \vee (S_p = done \wedge Error(p)))$. W ll p i i n only i i i $Error(p)$. A o o p i o o o q i n only i $(S_p = q) \wedge (Par_q = p) \wedge \neg SetError(q) \wedge (S_q \neq idle)$. In i p i ll . o q n q o p. W n l o S_p (. Par_q) n q (. p) i only il (. n) o p (. q). A $Par_r = \perp$, o vio ly, n v ny n .

A o G i $P = p_1, ..., p_k$ $S_{p_1} \notin \{idle, done\}$ n $\forall$ i, $1 \leq i \leq k-1$, p_i i lin o p_{i+1}. W will no $IE(P)$ o P (i. ., p_1) n $FE(P)$ o P (i. ., p_k). Mo ov , o P (no $length(P)$) i l o k. vio ly, in ny on ion, v y lin o G i l n y. So, o now on n n il n o , w only on i xi l non- y lin . T n x l iv n i o n o y o lin .

Lemma 1. P $Visited_{FE(P)} \supseteq \{Id_p :: p \in P \wedge p \neq IE(P)\}$

W ll no l lin , lin P i yin $Error(IE(P))$. - iv ly, w ll no l lin , v y lin w i i no no l. vio ly, no l lin P i $IE(P) =$.

Lemma 2. P $Visited_{FE(P)} \supseteq \{Id_p :: p \in P\}$

Now, w in o no ion o o lin . W ll o lin P vol ion o P in o ion. In i l , o P i n o ion o y P . No ,

, P y i . T , y onv n ion, w no y $Dead_P$ P i .

Definition 5 (Immediate Future of a Linked Path). $\gamma_i \mapsto \gamma_{i+1}$ P γ_i $F(P)$ P γ_{i+1}

If P' γ_{i+1} (a) $P \cap P' \neq \emptyset$ (b) γ_i $S_{FE(P)} = IE(P')$ $IE(P')$ F $\gamma_i \mapsto \gamma_{i+1}$ **then** $F(P) = P'$
else $F(P) = Dead_P$

$F(Dead_P) = Dead_P$

Fi 1 i wo y o i i . on i on ion *i* n *ii*. on ion *i* on in on lin only: $P =$, 1, . Mo ov , P o o 3 A ion F n l in *i* n x i in $i \mapsto ii$ (i. ., 3 oo on o P). T , $i \mapsto ii$ ill 1.(a) o D ni ion 5: in i x ion, $F(P) =$, 1, , 3. on ion *iii* l o on in on lin only: $P' = 1$. T n, in *iii*, P o o 1 A ion C n l n P o o A ion F n l . T wo o o x C n F iv ly in $iii \mapsto iv$ (1 n oo o P' n oo on o P'). So, w o in on ion *iv* w i ill 1.(b) o D ni ion 5: in i x ion, $F(P') =$. No i only P o o 1 x A ion C o *iii*, P' i , i. ., $F(P') = Dead_P$.

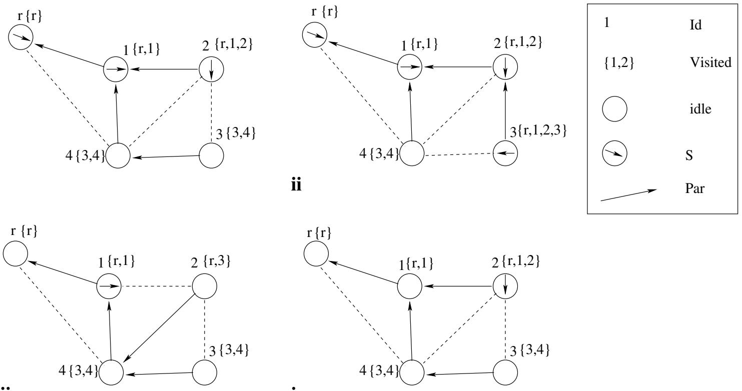


Fig. 1. Instances of Immediate Futures

Definition 6 (Future of a Linked Path). $e \in \mathcal{E}$ $\gamma_i \in e$ $F^k(P)$ $(k \in \mathbb{N})$ P e k γ_i

$F^0(P) = P$,
$F^1(P) = F(P)$ (P),
$F^k(P) = F^{k-1}(F(P))$ (P k), **if** $k > 1$

T ollowin n l iv o o i o lin n i .

. L $\gamma_i \mapsto \gamma_{i+1}$. L P lin in γ_i. $\forall p \in V$, p oo on o P in $\gamma_i \mapsto \gamma_{i+1}$ i n only i p x A ion F in $\gamma_i \mapsto \gamma_{i+1}$ n $p = FE(F(P))$ in γ_{i+1}. A Par_r i on n l o $\perp$, nno oo on o ny lin .

. L $\gamma_i \mapsto \gamma_{i+1}$ xi lin P in γ_i. A o o p n oo o P in $\gamma_i \mapsto \gamma_{i+1}$ in ollowin only:

1. P i n no l lin , $IE(P) = p$ n p x A ion C,
. $S_p = done$ n i n in P x A ion B ($p \neq$),
3. $p =$, i il q i $S_q = done$, n S_r o $done$ y x in A ion B. In i , q i l o n oo o P (.); o ov , in n v ny n , $IE(P) =$ n in S_r o $done$ involv P i , i. ., $F(P) = Dead_P$.

T ollowin l llow o l i , in o ion, i n i i o o o w i oo on o lin P n i in l in o $Visited$ o n l x i y o o P. By in A ion B n F o Al o i 1. n ., i l i y o v i y:

Lemma 3. P $F^k(P) \neq Dead_P$ ($k \in \mathbb{N}$), $Visited_{FE(F^k(P))}$ $Visited_{FE(P)}$ P $F^k(P)$

By in A ion F o Al o i 1. n ., ollow :

Lemma 4. P, $\forall$ $p \in V$ $Id_p \in Visited_{FE(P)}$, p P

By L 3 n , w n x l .

Lemma 5. P, $p \in V$ P, p $F^k(P)$, $\forall k \in \mathbb{N}^+$

In o , w y vol ion o . So, lo o l on n P n $F^k(P)$ wi $k \in \mathbb{N}$. F o now on, w n i no i i y, w l "P n $F^k(P)$, $\forall k \in \mathbb{N}$" y P only.

4.2 Proof Assuming a Weakly Fair Daemon

Now, w w ly i on. Un i ion, n o o ny o n i ni . So, w v n o lin in o , w n l o v l o lin in o o n . L $e \in \mathcal{E}$. L P lin in γ_i ($\in e$). W no $F_R^K(P)$ o P, in e, K o n o γ_i.

W now ow n wo on in no no l lin in o N o n , i. ., v y no l P o ini i l on ion i $F_R^N(P)$ $= Dead_P$.

Theorem 2. N

I i y o n o no l lin nno in . Mo ov , i A ion C i n l p, n i in n l n il p x i . So, l P n no l lin . A on i w ly i , o n , l on o o n oo o P (w il P xi). By L 1, n 5 n , n o o o w i n oo on o P i o $N - length(P)$. So, in wo , N o n n y o n oo o o o P n o w i will oo on. T , $F_R^N(P) = Dead_P$. □

T ollowin l n o llow o ov v n lly x A ion F.

Lemma 6. P P $Dead_P$ $N -$

L $e \in \mathcal{E}$. L $\gamma_i \in e$. A xi no l lin P in γ_i. Fi , w n o P i i no l lin o $Dead_P$. Mo ov , o vio ly, ion on P i i A ion F o A ion B. By L n 5, only o o p $Id_p \notin Visited_{FE(P)}$ (in γ_i) n oo on o P o on in x ion. By L , in wo , n o o o w i oo on o P in x ion i $N - length(P)$. T n, $N -$ o o n oo o P (i. ., $length(P)$ + $(N - length(P))$ − ion B on P), P i $length(P) =$. In i , only on ion n x on P: n o $FE(P)$ (i. ., $IE(P)$) n x A ion B. Now, y L 3, $Visited_{FE(P)} = \{Id_q :: q \in V\}$. So, y x in A ion B, $IE(P)$ $S_{IE(P)}$ o $done$ $(Next_{IE(P)})$. T , x l in in 3, P i . H n , in wo , o P i $Dead_P$ $N - length(P) + (N -) + 1$ ion w i i xi l i ini i ly $length(P) = 1$, i. ., $N -$ ion . □

I xi no no l lin , w n , o on o n , lw y xi l on on in o ly n l ion on no l lin . T , y L 6 ollow :

Lemma 7. P $F_R^{2N-1}(P) = Dead_P$

T o n L ov ollowin o .

Theorem 3. P, $F_R^{3N-1}(P) = Dead_P$

Theorem 4. F $3N$

By T o n 3, o ny ini i l on ion, y n o $3N-1$ o n o on ion γ_i i yin $\forall p \in V$, $S_p \in \{idle, done\}$. In γ_i, $\forall p \in V$ $S_p = done$, w v , $S_{Par_p} \neq p$. So, v y p A ion C on in o ly n l . A on i w ly i , on o n , $\forall p \in V$, $S_p = idle$. T , i only n l o o n A ion F i only n l ion o . H n , o ny ini i l on ion, oo x A ion F o $3N$ o n . □

F o x l n ion ovi in S ion 3, i i y o v i y w n y o on ion w $\forall p \in V$, $S_p = idle$ (l ll i *idle* on ion) i o v l o n wo o in o S i ion 1. Now, i y o n i y on ion, n i n on in o - l n o o n no l lin . W n - l n o o n no l lin n only low own o ion o no l lin . B y v n o no l vio no l lin o in w y n i i o *idle* on ion. So, no l lin v n lly vi i ll o o in *first DFS* o n , , v n lly in ion o w v w n S_r o *done* ($\forall p \in V, Id_p \in Visited_r$). H n :

Theorem 5. F,

F o 1, T o n 5, ollow :

Theorem 6. *snapDFS*

4.3 Proof Assuming an Unfair Daemon

F o now on, w o no ny i n ion. T wo n x l llow o ov , in ny x ion o Al o i *snapDFS*, o n i ni .

Lemma 8. P $Dead_P$ $N-1$

T onnin i i il o oo o L 6. □

Lemma 9. *snapDFS*

L $e \in \mathcal{E}$. A o n R o e n in ni n o . L γ_R on ion o R.

Fi , o no l lin o γ_R n v i . So, y v n lly on ion $\gamma_i \in R$ in w i xi only no l lin w i n v i . Now, v y no l lin i ni n o ion on i (L), xi on ion γ_j $(j \geq i)$ o w i no ion will x on no l lin . T n, v y - l n o o i l n on A ion C n no l lin n only n ni n o - l n o o . In , - l n o o n y no l lin lon o i o n , n il no l lin i , only ni n o o o n oo on o i (L 5). T n, - l n o o nno v n o v ion o x on no l lin . Now, y L 6, v y no l lin i ni n o . So, oo o o x A ion F in ni iv ly o n o no l lin . B , i x A ion F, n, y T o 5, n w no l lin P n v y o o ($\neq$) v n lly oo on o P in x ion (in i l , o o o no l lin). Now, o o p n oo on o P i $S_p = idle$ (n P i $Forward(p)$). T , P i v n lly lo o o o no l lin n v oo on o i . So, nno x A ion F in ni iv ly o n, on i ion. T , xi $\gamma_{j'} \mapsto \gamma_{j'+1}$ wi $j' \geq j$ in w i l on ion i x on n no l lin , on i ion. H n , no l lin v n lly i .

So, xi on ion γ_k in w i xi no no l lin . F o i on ion, lw y xi o on lin , no l lin . A xi no no l lin in γ_k. T n, ni n o , x A ion F n no l lin P (in wo , $O(N)$ A ion C, v y - l n o o o i l n i only n l o o). A x l in ov , - l n o o nno v n o v ion o x on P. By L 6, o P i $Dead_P$ ni n o ion on i . Now, y T o 5, o i in , v y o o oo on o i y x in A ion F. So, o n R i v n lly on , on i ion. Fin lly, i xi no l lin P' in γ_k, y i il onnin , ni n o , o P' i $Dead_{P'}$ n w i v vio , on i ion.

H n , ni n o , v y n l o o o γ_R x on ion. □

By T o 1 n 6, n L 9, ollowin o ol .

Theorem 7. *snap$\mathcal{DFS}$*

4.4 Complexity Analysis

By in Al o i 1. n ., ollow :

Lemma 10. *snapDFS* $O(N \times$ lo (N) + lo $(\Delta))$

Lemma 11. F $O(N^2)$

In ini i l on ion, y n on in $O(N)$ - l n o- o n $O(N)$ lin . T n, v y lin n n $O(N)$ - l n o o . In , - l n o o n y lin lon o i o n , n il lin i , v y o o n oo on o i o on (L 5). Fin lly, v y - l n o o l n i y x in A ion C. An , v y lin i $O(N)$ ion on i (L 6 n). H n , in wo , $O(N^2)$, i only n l o o n x A ion F in n x . □

T ollowin l n o L 11.

Lemma 12. *fDFS* $O(N^2)$

By L , n T o n , w n ollowin l .

Lemma 13. *fDFS* $5N - 1$

5 Conclusion

W n n - ili in - w v o o ol o i y oo n wo . T o o ol o no ny - o nnin i i n i i on o o . T n - ili in o y n oon oo ini i o o ol, v y o o o n wo will vi i in *DFS* o . A n o vi i , oo v n lly in ion o o . F o , o o o ol i n - ili in , y ni ion, i i l o l - ili in o o ol w i ili in 0 o n (. 0). vio ly, o o o ol i o i l in ili ion i . W l o ow o o o o ol wo o ly in n n i on, i. ., in w lin ion. Fin lly, no o o o ol x o l v l o n wo in $O(N)$ o n n $O(N^2)$, iv ly. T o y i n o o ol ion i $O(N \times$lo $(N) +$ lo $(\Delta))$ i o o . In wo , w wo l li o i n n - ili in *DFS* w v o o ol (o i y oo n wo) wi o y i n in n n o N.

Acknowledgements. W wo l li o n nony o o i ion n on iv o n on li v ion o . T i ion v ly n n ili y o .

References

1. Tel, G.: Introduction to distributed algorithms. Cambridge University Press (Second edition 2001)
2. Dijkstra, E.: Self stabilizing systems in spite of distributed control. Communications of the Association of the Computing Machinery **17** (1974) 643–644
3. Bui, A., Datta, A., Petit, F., Villain, V.: State-optimal snap-stabilizing PIF in tree networks. In: Proceedings of the Forth Workshop on Self-Stabilizing Systems, IEEE Computer Society Press (1999) 78–85
4. Awerbuch, B.: A new distributed depth-first-search algorithm. Information Processing Letters **20** (1985) 147–150
5. Cheung, T.: Graph traversal techniques and maximum flow problem in distributed computation. IEEE Transactions on Software Engineering **SE-9(4)** (1983) 504–512
6. Collin, Z., Dolev, S.: Self-stabilizing depth-first search. Information Processing Letters **49(6)** (1994) 297–301
7. Huang, S., Chen, N.: Self-stabilizing depth-first token circulation on networks. Distributed Computing **7** (1993) 61–66
8. Datta, A., Johnen, C., Petit, F., Villain, V.: Self-stabilizing depth-first token circulation in arbitrary rooted networks. Distributed Computing **13(4)** (2000) 207–218
9. Johnen, C., Beauquier, J.: Space-efficient distributed self-stabilizing depth-first token circulation. In: Proceedings of the Second Workshop on Self-Stabilizing Systems. (1995) 4.1–4.15
10. Petit, F., Villain, V.: Color optimal self-stabilizing depth-first token circulation. In: I-SPAN'97, Third International Symposium on Parallel Architectures, Algorithms and Networks Proceedings, IEEE Computer Society Press (1997) 317–323
11. Petit, F.: Fast self-stabilizing depth-first token circulation. In: Proceedings of the Fifth Workshop on Self-Stabilizing Systems, Lisbonne (Portugal), LNCS 2194 (October 2001) 200–215
12. Petit, F., Villain, V.: Time and space optimality of distributed depth-first token circulation algorithms. In: Proceedings of DIMACS Workshop on Distributed Data and Structures, Carleton University Press (1999) 91–106
13. Cournier, A., Datta, A., Petit, F., Villain, V.: Enabling snap-stabilization. In: 23th International Conference on Distributed Computing Systems (ICDCS 2003). (2003) 12–19
14. Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. IEEE Transactions on Parallel and Distributed Systems **8** (1997) 424–440

A Self-stabilizing Link-Coloring Protocol Resilient to Byzantine Faults in Tree Networks

Y S i[1], F i o o i [2], n To i i M w [2]

[1] Information and Communication Systems Group,
Sharp Corporation Yamatokoriyama-shi, 639-1186 Japan
sakurai.yuhsuke@sharp.co.jp

[2] Graduate School of Information Science and Technology,
Osaka University, Toyonaka-shi, 560-8531 Japan
{f-oosita, masuzawa}@ist.osaka-u.ac.jp

Abstract. Self-stabilizing protocols can tolerate any type and any number of transient faults. But self-stabilizing protocols have no guarantee of their behavior against permanent faults. Thus, investigation concerning self-stabilizing protocols resilient to permanent faults is important.

This paper proposes a self-stabilizing link-coloring protocol resilient to (permanent) Byzantine faults in tree networks. The protocol assumes the central daemon, and uses $\Delta + 1$ colors where Δ is the maximum degree in the network. This protocol guarantees that, for any nonfaulty process v, if the distance from v to any Byzantine ancestor of v is greater than two, v reaches its desired states within three rounds and never changes its states after that. Thus, it achieves fault containment with radius of two. Moreover, we prove that the containment radius becomes $\Omega(\log n)$ when we use only Δ colors, and prove that the containment radius becomes $\Omega(n)$ under the distributed daemon. These lower bound results prove necessity of $\Delta + 1$ colors and the central daemon to achieve fault containment with a constant radius.

1 Introduction

[5] i on o o ff iv n o i in i o
l-ol n i i o in [6]. A l- ili in o o ol i n
o i v i i vio v n lly l o ini i l n wo on-
ion (i. ., lo l). T i i li l- ili in o o ol i ili n
o ny n n ny y o n i n l in i n onv o i -
i vio o ny on ion l y n i n l . How v
onv n o i vio i n only on ion
no l o in onv n . T , l- ili in o o ol
i no n o i v i i vio in n o n n
l . T , i i on ly i o i n l- ili in o o ol ili n o
n n l .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 283–298, 2005.

T o o l- ili in o o ol ili n o -
n n l [1, ,10,16,3,15,1 ,19]. Mo o only
l , n l- ili in o o ol n non l y o-
i v i i vio l o ini i l n wo on ion.
N n o l.[1] By n in l n n l . T in i-
l y in ol in By n in l i y i y n n o n
n o By n in o : o o n By n in o y
n i in on o n o By n in o ,
n o n x o o n in i y l o n i
. T i i li infl n o By n in o x n o
w ol y , n n no o n i v i i vio . N n o
l.[1] iv nov l ni ion o l- ili in o o ol ili n o By n in
l . T o o ol n , y on inin infl n o By n in o-
o only o n , o o n i v i i
vio v n lly. T y in o on in n i i n -
w n By n in o n o o ff y By n in o .
T y l o o o l- ili in o o ol ili n o By n in l o
v x olo in o l n inin ilo o o l . T on in n
i i on o v x olo in o l n wo o inin ilo o
o l .

T on o l on in n i v y o l in l o l- ili in
o o ol[, ,9,13,1 ,]. How v , i o on in infl n o
n i n l , n y o no on i By n in l .

In i , w on i l- ili in lin - olo in o o ol ili n o
By n in l in n wo . Lin - olo in o i i y i n
i n n o olo o o ni ion lin no wo o ni ion
lin wi olo o in o on. Lin - olo in ny
li ion in i i y , . ., lin n n i nin
n y n in wi l n wo . T , ny i i o o ol o lin -
olo in o o [11,1 ,1]. How v , l ol n i no on i
in o o ol .

In i , w o o l- ili in lin - olo in o o ol ili n o
By n in l . T o o ol n l on, i. ., x ly on o-
n x n o ion i , n $\Delta+1$ olo , w Δ i
xi o n wo . T o o ol n ny non l y
o v i i wi in o n n n v n i
i v no By n in n o wi i n o wo o l .
Mo ov , w ow , o ny l- ili in lin - olo in o o ol ili n
o By n in l , w n i only Δ olo , on in n i o
$\Omega($lo $n)$ i $\Delta \geq 3$, n $\Omega(n)$ i $\Delta =$, w n i n o o .
T , o o o o o ol in ini li y in n o olo o
i vin l on in n o By n in o wi on n on in n
i . N x , o ny l- ili in lin - olo in o o ol i-
i on, i. ., n i y n o o n x o ion
i , v n w n i n i y n o olo , w ow

on in n i o $\Omega(n)$. T i low o n l i li ion o n l on i on l o in l on in n in By n in l wi on n on in n i .

2 Preliminaries

2.1 Distributed System

A $S = (P, L)$ on i o $P = \{v_1, v_2, \ldots, v_n\}$ o o n L o o ni ion lin (i ly ll lin). A lin i n no i o i in o n o v n w ll i $(v, w) \in L$. A i i y S n wi v x P n lin L, n , w o inolo i o i i y S.

W on i in i . Fo o $v \in P$, N_v no o n i o o v, prt_v no n o v, n Ch_v no o il n o v. W o no xi n o ni i n i o o . In w o n i n i y i n o , n i in i o i il n y lo l o in on i il n. T x-il o o v i no y $ch_v(x)$ $(1 \leq x \leq |Ch_v|)$. T i n o oo o o o v i ll o v. T xi o n wo i no y Δ, i. ., oo o o Δ il n n ny o o o $\Delta - 1$ il n.

o i o l y in n o ni wi i n i o o lin i . Fo i o n i o in o , u n v, wo lin i $r_{u,v}$ n $r_{v,u}$. M n i ion o u o v i li ollow : u w i o lin i $r_{u,v}$ n n v i o $r_{u,v}$.

Fo o v, l $In_v = \{r_{u,v} | u \in N_v\}$ in i o v n $Out_v = \{r_{v,u} | u \in N_v\}$ o i o v. Fo onv ni n , w v i l o no o o n lin i , n (i ly ll) o no n i ion n ion o o . ion i o ollowin o :

$< guard > \rightarrow < statement >$

T o n ion o o v i ool n x ion on i in o v i l o v n ll in i r $(r \in In_v)$. T n o n ion o v on o o v i l o v n ll o i r $(r \in Out_v)$. T v l i n o v i l o v n r $(r \in Out_v)$ n only on v l o v i l o v n r $(r \in In_v)$. T n o n ion n x only i i i v l o . W n o l i l ion v l o , on o ion i ini i lly l n x . W ion i o i lly x : v l ion o n x ion o o on in n o ion, i x , on in on o i . T x ion o n ion o v i ll o v.

A lo l o i i y i ll n i no y o o o ll o n ll lin i . W n C o ll o i l on ion o i i y S. Fo on ion $\rho \in C$, $\rho|v$ n $\rho|r$ no o o v n lin i r in on ion ρ iv ly.

W n o v ion w o i on ion ρ, w y v i ρ. L $En(\rho, v)$ i $En(\rho, v) = true$ iff v i n l ρ. L in Q ny o o , w n on ion ρ n o on ion ρ' y x in ion o v y n l o in Q, w no $\rho \overset{Q}{\mapsto} \rho'$.

A o i i y i n in ni n o o o . L $\mathcal{Q} = Q_1, Q_2, \ldots$ l . An in ni n o on ion $e = \rho_0, \rho_1, \ldots$ i ll n o n ini i l on ion ρ_0 y l $\mathcal{Q}$, i e i $\rho_i \overset{Q_{i+1}}{\mapsto} \rho_{i+1}$ o i $(i \geq 0)$. No i x ion e i ni ly in o i ini i l on ion n l $\mathcal{Q}$ in x ion o o i ini i lly l (v n w n o wo o o ion wi). T o o i l l in i i y i o i o l y l ll on. In i , w on i wo in o on , n . Un i i on, Q_i n n i y o o . T i , i i on llow wo o o o x i ion i l n o ly. By on , n l on i i l o i i on. Un n l on, $|Q_i| = 1$ ol o i, i. ., no wo o x i ion i l n o ly. Un n l on, w n $Q_i = \{q_i\}$ o i, w i ly i l $\mathcal{Q} = q_1, q_2, \ldots$ n i on ion n i ion $\rho_i \overset{q_{i+1}}{\mapsto} \rho_{i+1}$ in o $\rho_i \overset{Q_{i+1}}{\mapsto} \rho_{i+1}$. T o ll o i l x ion o n ini i l on ion $\rho_0 \in C$ i no y E_{ρ_0}. T o ll o i l x ion i no y E, i , $E = \bigcup_{\rho_0 \in C} E_{\rho_0}$.

W on i i i y w w n no - ion on l x ny l i : v y o in l in ni ly o n.

In i , w on i wo in o n n l : l n By n in l .

- : A o (i. ., o wi l) ly o x ion o i ion . I v i o , v o no n o v n r $(r \in Out_v)$ in i v n w n i n ion wi . B o i , v non l y o n o ly x i ion .
- : A By n in o (i. ., o wi By n in l) n i ily v in n ly o i ion . I v i By n- in o , v n ly n o v n r $(r \in Out_v)$ i ily.

W n BF n CF o By n in o n o iv ly. Sin l n i l o By n in l , $BF \supseteq CF$ ol . How v , in w ollow , w wi o lo o n li y $BF \cap CF = \emptyset$ ol y x l in o o BF.

L $CF = \{f_1, f_2, \ldots, f_c\}$. In i i y w l n o , n in ni n o on ion $e = \rho_0, \rho_1, \ldots$ i ll n y l $\mathcal{Q} = Q_1, Q_2, \ldots$, i xi $t_1, t_2, \ldots, t_c$ ollowin on i ion ol o ny i ($i \geq 0$):

- Fo ny $v \in Q_{i+1} - (BF \cup CF)$, x ion o n ion o v n v' o $\rho_i|v$ o $\rho_{i+1}|v$ (o i ly $\rho_i|v = \rho_{i+1}|v$) n n o v y $r \in Out_v$ o $\rho_i|r$ o $\rho_{i+1}|r$ (o i ly $\rho_i|r = \rho_{i+1}|r$).
- Fo ny $f_j \in Q_{i+1} \cap CF$, i $i \geq t_j$, o f_j n o i $r \in Out_{f_j}$ in n n o ρ_i o ρ_{i+1}: $\rho_i|f_j = \rho_{i+1}|f_j$ n $\forall r \in Out_{f_j} : \rho_i|r = \rho_{i+1}|r$ ol . I $i < t_j$, x ion o n ion o f_j n i o $\rho_i|f_j$ o $\rho_{i+1}|f_j$ (o i ly $\rho_i|f_j = \rho_{i+1}|f_j$) n n o v y $r \in Out_{f_j}$ o $\rho_i|r$ o $\rho_{i+1}|r$ (o i ly $\rho_i|r = \rho_{i+1}|r$). No i t_j i li o f_j o w n ρ_{t_j-1} n ρ_{t_j}.
- Fo ny $v \notin Q_{i+1}$, $\rho_i|v = \rho_{i+1}|v$ n $\forall r \in Out_v : \rho_i|r = \rho_{i+1}|r$ ol .

No i , o ny o $v \in Q_{i+1} \cap BF$, $\rho_{i+1}|v$ n $\rho_{i+1}|r$ ($r \in Out_v$) n i y .

In yn ono i i y , i i lly y (i ly ll). L $e = \rho_0, \rho_1, \ldots$ n x ion o on ion ρ_0 y l $\mathcal{Q} = Q_1, Q_2, \ldots$. T o n o e i n o ini x o e, $e' = \rho_0, \rho_1, \ldots, \rho_k$, $\bigcup_{i=1}^{k} Q_i = P$. o n t ($t \geq$) i n iv ly, y lyin ov ni ion o o n o $e'' = \rho_k, \rho_{k+1}, \ldots$. In i iv ly, v y o n o i in v y o n .

2.2 Self-stabilizing Protocol Resilient to Byzantine Faults

In i , w only , i. ., on y i on ion, on ion in n n o v . Fo x l , nnin - on ion o l i i o l , n l xl ion o l i no i o l [6]. A i o l n n y , *spec*(v), o o v, w i i on iion v o l i y i on ion. A i ion i *spec*(v) i ool n x ion on i in o v i l o $P_v \subseteq P$ n lin i $R_v \subseteq R$, w R i o ll lin i .

A l- ili in o o ol i o o ol n o v i*spec*(v) v n lly l o ini i l on ion. By i o y, l- ili in o o ol n ol ny n n ny y o n i n l . How v , in w on i n n l By n in l n l , l y o nno i y *spec*(v). In i ion, non l y o n l y o n infl n y l y o n nno i y *spec*(v). T , N n o l.[1] n l- ili in o o ol ili n o l . In o lly, o o ol i non l y o v o ny l y o o i y *spec*(v) v n lly. T y l o o o ono i ol n n i ili ion o n o l o o ol ili n o By n in l . W o in ov wo on , n o o $(\mathcal{B},\mathcal{C})$- l- ili ion wi i (τ,μ), w $\mathcal{B}$ n $\mathcal{C}$ n By n in l n l iv ly. In ollowin ni ion, l $\Gamma(v,l)$ o o w o i n o v i o l.

Definition 1. [illegible] ρ_0 [illegible] $(\mathcal{B},\mathcal{C})$ [illegible] (τ,μ) [illegible] $e = \rho_0, \rho_1, \ldots$ [illegible] v, [illegible]

[illegible] v [illegible] τ $(\forall w \in \Gamma(v,\tau) : w \notin BF)$ [illegible] v [illegible] μ $(\forall u \in \Gamma(v,\mu) : u \notin CF)$, [illegible] i,) v [illegible] $spec(v)$ [illegible] ρ_i,) $\rho_i|v = \rho_{i+1}|v$ [illegible]) $\rho_i|r = \rho_{i+1}|r$ $(r \in Out_v)$ [illegible]

D ni ion 1 , on y l on ion, o v i n ly o ny l y o i $spec(v)$ n n v n o v n r $(r \in Out_v)$ o v .

Definition 2. [illegible] A [illegible] $(\mathcal{B},\mathcal{C})$ [illegible] (τ,μ) [illegible] $e = \rho_0, \rho_1, \ldots$ A [illegible] ρ_0, [illegible] ρ_i [illegible] $(\mathcal{B},\mathcal{C})$ [illegible] (τ,μ) [illegible] A [illegible] k [illegible] k [illegible] k [illegible] A

D ni ion $(\mathcal{B},\mathcal{C})$- l- ili in o o ol n y- v n lly $(\mathcal{B},\mathcal{C})$- l on ion o ny ini i l on -ion. I o o ol A i $(\mathcal{B},\mathcal{C})$- l- ili in o o ol wi i (τ,μ), w τ n μ on n , A i i ly $\mathcal{C}$- ol n [1] n i ly ili in [1].

2.3 Link-Coloring Problem

A lin - olo in o l i o n n i n n o olo o lin no wo lin wi olo o in o on. In ollowin , l $CSET$ iv n o olo , n l $Color((u,v)) \in CSET$ olo o lin (u,v). W n i i lin - olo in o l ollow .

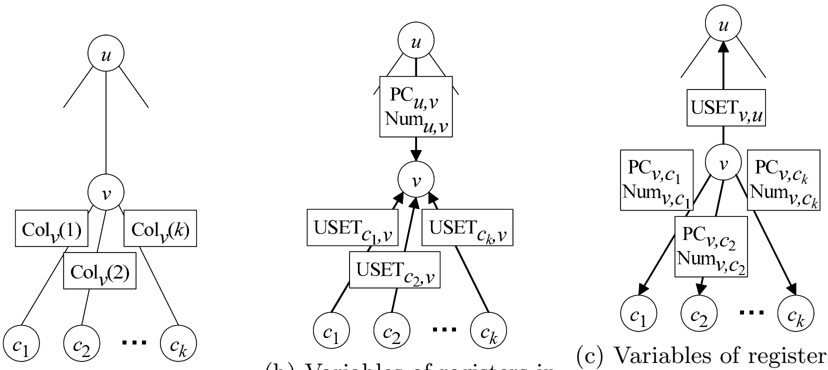
Definition 3. [illegible] $spec(v)$ [illegible] v [illegible]

$$\forall x, y \in N_v : x \neq y \Longrightarrow Color((v,x)) \neq Color((v,y))$$

In ollowin , w n b-[illegible] lin - olo in o o ol in b olo .

3 Link-Coloring Protocol Under the Central Daemon

In i ion, w o o $(\mathcal{B},\mathcal{C})$- l- ili in $(\Delta+1)$-lin - olo in o o ol wi i (,1). o o ol o $\Delta+1$ olo o lin - olo in , n , w $CSET = \{1, \ , \ldots, \Delta+1\}$.



(a) Variables of v.

(b) Variables of registers in In_v.

(c) Variables of registers in Out_v.

Fig. 1. Variables of v, link registers in In_v, and link registers in Out_v, where $u = prt_v$ and $c_x = ch_v(x)$ $(1 \le x \le |Ch_v|)$

L v ny o , $u = prt_v$, n x_v n in i yin $v = ch_u(x_v)$. Fi , w x l in v i l on o n lin i (S Fi . 1).

- P o v v i l $\mathtt{Col}_v(x)$ $(1 \le x \le |Ch_v|)$. V i l $\mathtt{Col}_v(x)$ no olo o lin $(v, ch_v(x))$. No i v o no v v i l o o olo o lin (u, v) o i n u. T olo o lin i o in $\mathtt{Col}_u(x_v)$ o u.
- Lin i $r_{u,v}$ v i l $\mathtt{Num}_{u,v}$ n $\mathtt{PC}_{u,v}$. P o u i n x_v o $\mathtt{Num}_{u,v}$, n i n $\mathtt{Col}_u(x_v)$ o $\mathtt{PC}_{u,v}$. P o v n l n olo o lin (u, v) y in $\mathtt{PC}_{u,v}$. T v l o $\mathtt{Num}_{u,v}$ i o in $\mathtt{Col}_v(x)$ $(1 \le x \le |Ch_v|)$.
- Lin i $r_{v,u}$ v i l $\mathtt{USET}_{v,u}$. P o v i n $\left\{\mathtt{Col}_v(x) \middle| 1 \le x \le |Ch_v|\right\}$ o $\mathtt{USET}_{v,u}$. P o u n l n olo i n o lin $(v, ch_v(x))$ $(1 \le x \le |Ch_v|)$ y in $\mathtt{USET}_{v,u}$.

Fo i li i y, w $\mathtt{Col}_v(x) \in CSET$ $(1 \le x \le |Ch_v|)$, $1 \le \mathtt{Num}_{u,v} \le \Delta$, $\mathtt{PC}_{u,v} \in CSET$, n $\mathtt{USET}_{v,u} \subseteq CSET$ lw y i ny on - ion v n w n xi o By n in o .

P o v x ollowin o i lly:

1. P o v v i l on ll lin i in In_v.

. P o v lo lly in olo $\mathtt{Col}_v(x)$ o ll x $(1 \le x \le |Ch_v|)$.

3. Fo x $(1 \le x \le |Ch_v|)$, l in $w = ch_v(x)$, o v w i x n $\mathtt{Col}_v(x)$ o $\mathtt{Num}_{v,w}$ n $\mathtt{PC}_{v,w}$ on lin i $r_{v,w}$ iv ly.

. L in $u = prt_v$, o v w i $\left\{\mathtt{Col}_v(x) \middle| 1 \le x \le |Ch_v|\right\}$ o $\mathtt{USET}_{v,u}$ on lin i $r_{v,u}$.

W ow o o ol LinkColoring in Fi . . Fo i li i y, w ow o o ol y ivin o- o . T n ion LinkColoring i x in on o i .

```
function LinkColoring {
  // $v$ is the root process
  if $v = root$ then
    for $k := 1$ to $|Ch_v|$ {
      $\texttt{Col}_v(k) := k$
    }

  // $v$ is not the root process
  else
    // assign colors to $\texttt{Col}_v(k)$ $(k < \texttt{Num}_{u,v})$
    $u := prt_v$
    $c := 1$
    for $k := 1$ to $\min(\{\texttt{Num}_{u,v} - 1, |Ch_v|\})$ {
      if $\texttt{PC}_{u,v} = c$ then
        $c++$
      endif
      $\texttt{Col}_v(k) := c$
      $c++$
    }

    // assign colors to $\texttt{Col}_v(k)$ $(k > \texttt{Num}_{u,v})$
    $c := |Ch_v|$
    for $k := |Ch_v|$ downto $\texttt{Num}_{u,v} + 1$ {
      if $\texttt{PC}_{u,v} = c$ then
        $c--$
      endif
      $\texttt{Col}_v(k) := c$
      $c--$
    }

    // assign colors to $\texttt{Col}_v(k)$ $(k = \texttt{Num}_{u,v})$
    if $|Ch_v| \geq \texttt{Num}_{u,v}$ then
      $k := \texttt{Num}_{u,v}$
      if $\texttt{Col}_v(k) \notin \{k, k+1, k+2\}$ or
           $\texttt{PC}_{u,v} = \texttt{Col}_v(k)$ or
           $\texttt{Col}_v(k-1) = \texttt{Col}_v(k)$ or
           $\texttt{Col}_v(k+1) = \texttt{Col}_v(k)$ then
        $C := \{k, k+1, k+2\} - \{\texttt{PC}_{u,v}\} - \texttt{USET}_{ch_v(k),v}$
        if $C \neq \emptyset$ then
          $\texttt{Col}_v(k) := \min(C)$
        else
          $\texttt{Col}_v(k) := \min(\{k, k+1, k+2\} - \{\texttt{PC}_{u,v}\})$
        endif
      endif
    endif
  endif

  // write colors to link registers
  for $k := 1$ to $|Ch_v|$ {
    $\texttt{Num}_{v,ch_v(k)} := k$
    $\texttt{PC}_{v,ch_v(k)} := \texttt{Col}_v(k)$
  }
  if $v \neq root$ then
    $\texttt{USET}_{v,u} := \left\{ \texttt{Col}_v(k) \middle| 1 \leq k \leq |Ch_v| \right\}$
  endif
}
```

Fig. 2. The protocol LinkColoring: the action of v

To x l in ow o v in $\texttt{Col}_v(x)$, w n n i olo n n olo . Fo o v n x $(1 \leq x \leq |Ch_v|)$, w n $CCol_v(x)$ ollow :

$$CCol_v(x) = \{x, x+1, x+ \ \}$$

Fo o v (v i no oo) n x ($1 \leq x \leq |Ch_v|, x \neq \mathtt{Num}_{prt_v,v}$), w n $PCol_v(x)$ ollow :

$$PCol_v(x) = \begin{cases} x & (x < \mathtt{Num}_{prt_v,v}) \\ x+ & (x > \mathtt{Num}_{prt_v,v}) \end{cases}$$

In o o ol LinkColoring, non l y o v i n olo o lin wi ollowin oli i : 1) v lw y i n olo o lin o no wo lin wi olo v,) v lw y i n olo $c \in CCol_v(x)$ o $\mathtt{Col}_v(x)$, n 3) i o i l , v i n olo $PCol_v(x)$ o $\mathtt{Col}_v(x)$. A o in o oli i , v i n olo o $\mathtt{Col}_v(x)$ ollow :

- w v i oo o . P o v i n x o $\mathtt{Col}_v(x)$ ($1 \leq x \leq |Ch_v|$) (S lin o 6).
- w v i no oo o . L $u = prt_v$ n $w = ch_v(\mathtt{Num}_{u,v})$.
  1. w $\mathtt{PC}_{u,v} \in CCol_v(\mathtt{Num}_{u,v}) = \{\mathtt{Num}_{u,v}, \mathtt{Num}_{u,v}+1, \mathtt{Num}_{u,v}+ \ \}$
     - Fo x ($x < \mathtt{Num}_{u,v}$), v i n x o $\mathtt{Col}_v(x)$ (S lin 10 o 19).
     - L $C = CCol_v(\mathtt{Num}_{u,v}) - \{\mathtt{PC}_{u,v}\}$. I xi c $c \in C - \mathtt{USET}_{w,v}$, v i n c o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$. wi , v i n ny olo $c \in C$ o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$ (S lin 30 o).
     - Fo x ($x > \mathtt{Num}_{u,v}$), v i n $x+$ o $\mathtt{Col}_v(x)$ (S lin 1 o 9).
  . w $\mathtt{PC}_{u,v} < \mathtt{Num}_{u,v}$.
     - Fo x ($x < \mathtt{PC}_{u,v}$), v i n x o $\mathtt{Col}_v(x)$ (S lin 10 o 19).
     - Fo x ($\mathtt{PC}_{u,v} \leq x < \mathtt{Num}_{u,v}$), v i n $x+1$ o $\mathtt{Col}_v(x)$(S lin 10 o 19).
     - L $C = CCol_v(\mathtt{Num}_{u,v}) - \{\mathtt{Num}_{u,v}\}$. I xi c $c \in C - \mathtt{USET}_{w,v}$, v i n c o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$. wi , v i n ny olo $c \in C$ o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$ (S lin 31 o 5).
     - Fo x ($x > \mathtt{Num}_{u,v}$), v i n $x+$ o $\mathtt{Col}_v(x)$ (S lin 1 o 9).
  3. w $\mathtt{PC}_{u,v} > \mathtt{Num}_{u,v} +$.
     - Fo x ($x < \mathtt{Num}_{u,v}$), v i n x o $\mathtt{Col}_v(x)$ (S lin 10 o 19).
     - L $C = CCol_v(\mathtt{Num}_{u,v}) - \{\mathtt{Num}_{u,v} + \ \}$. I xi c $c \in C - \mathtt{USET}_{w,v}$, v i n c o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$. wi , v i n ny olo $c \in C$ o $\mathtt{Col}_v(\mathtt{Num}_{u,v})$ (S lin 31 o 5).
     - Fo x ($\mathtt{Num}_{u,v} < x \leq \mathtt{PC}_{u,v} - \ $), v i n $x+1$ o $\mathtt{Col}_v(x)$ (S lin 1 o 9).
     - Fo x ($\mathtt{PC}_{u,v} - \ < x$), v i n $x+$ o $\mathtt{Col}_v(x)$ (S lin 1 o 9).

A v x n ion, o ion o v o l n $spec(v)$ o . No i o ion o v n $spec(v)$ o no in l v i l $\mathtt{USET}_{w,v}$ ($w \in Ch_v$). T , on v x n ion, v n w n $w \in Ch_v$ i By n in o n n $\mathtt{USET}_{w,v}$ i ily, o ion o v in l n $spec(v)$ in nl prt_v n $\mathtt{PC}_{prt_v,v}$.

A v n $u = prt_v$ non l y o , n prt_u i By n in o . In w ollow , w x l in ow infl n o By n in o prt_u i on in in n i olo n n olo .

Sin u i non l y o , y o o ol, u i n olo in $CCol_u(x)$ o $\texttt{Col}_u(x)$ in o n . T , o u infl n o By n in o prt_u in n n o $\texttt{Col}_u(x)$ i on in in $CCol_u(x)$ l o n o $\texttt{PC}_{prt_u,u}$ i o l ly n on in .

A i ov , u lw y i n olo in $CCol_u(x)$ o $\texttt{Col}_u(x)$ $(1 \le x \le |Ch_u|)$. on n ly, l in $C = \{\texttt{Num}_{u,v}, \texttt{Num}_{u,v}+1, \texttt{Num}_{u,v}+ \}$, $\texttt{PC}_{u,v} = \texttt{Col}_u(\texttt{Num}_{u,v}) \in CCol_u(\texttt{Num}_{u,v}) = C$ ol ny on ion o n . T n, w n v x o n , v i n $PCol_v(x)$ o $\texttt{Col}_v(x)$ o x $(x \neq \texttt{Num}_{u,v})$, n i n olo in $CCol_v(\texttt{Num}_{u,v})$ o $\texttt{Col}_v(\texttt{Num}_{u,v})$. T i i li v n v n $\texttt{Col}_v(x)$ $(x \neq \texttt{Num}_{u,v})$ v n w n u n $\texttt{PC}_{u,v}$. How v , v y v o n $\texttt{Col}_v(\texttt{Num}_{u,v})$ in on o n o $\texttt{PC}_{u,v}$. L in $w = ch_v(\texttt{Num}_{u,v})$, in w l o i n $PCol_w(x)$ o $\texttt{Col}_w(x)$ o x $(x \neq \texttt{Num}_{v,w})$ on o n , lin n i n olo in C o n onn o i v o w only (u,v), (v,w), n $(w, ch_w(\texttt{Num}_{u,v}))$. T , v n i n olo in C o $\texttt{Col}_v(\texttt{Num}_{u,v})$ o no wo lin wi olo i v o w. T o , w, o o By n in n o y i n o , i no ff y By n in o , n w in l on in n in By n in l . A o o o ol LinkColoring, w v ollowin o .

Theorem 1. LinkColoring $(\Delta + 1)$

- $e = \rho_0, \rho_1, \ldots$ $v \in P$ ρ_s v t $(t \geq s)$) v $spec(v)$ ρ_t) $\rho_t|v = \rho_{t+1}|v$ $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$

Corollary 1. LinkColoring $(\mathcal{B},\mathcal{C})$ $(\Delta + 1)$ (,1) LinkColoring

In i ion, w n o v i in i $\texttt{PC}_{v,ch_v(x)} = \texttt{Col}_v(x)$ ol o ny x $(1 \le x \le |Ch_v|)$. No i , o ny o $v \notin BF$, on v x n ion, v i in on i n o v . Fo non l y o v o ny By n in n o y i n o , w n prt_v i o , w nno n v i $spec(v)$. How v , i prt_v in on i n , w n n .

Theorem 2. LinkColoring $(\Delta + 1)$

- $e = \rho_0, \rho_1, \ldots$ $v \in P$ v, ρ_s v

t $(t \geq s)$,) v $spec(v)$ ρ_t,) $\rho_t|v = \rho_{t+1}|v$ $\forall r \in Out_v : \rho_t|r = \rho_{t+1}|r$

4 Impossibility of Link-Coloring Using Δ Colors Under the Central Daemon

In i ion, w on i l- ili in Δ-lin - olo in o o ol. W n lin - olo n wo wi Δ olo . How v , w ow , o ny l- ili in Δ-lin - olo in o o ol, on in n i i $\Omega(\text{lo } n)$ i $\Delta \geq 3$, n $\Omega(n)$ i $\Delta =$, w n i n o o . T , o o ol LinkColoring in ini li y in n o olo o i vin l on in n o By n in o wi on n on in n i . To ow low o n , w n o v o v n ll lin i in In_v, n $view(\rho, v)$ no vi w o o v in on ion ρ.

Theorem 3. $\Delta \geq 3$ $(\mathcal{B},\mathcal{C})$ Δ $(\tau(n), \mu(n))$, $\tau(n) = \Omega(\text{lo } n)$ n

W olo o lin (u, v) i in (o o) y o u, v, $r_{u,v}$, n $r_{v,u}$. A A i $(\mathcal{B},\mathcal{C})$- l- ili in Δ-lin - olo in o o ol wi i $(\tau(n), \mu(n))$.

L y $S = (P, L)$ o l $(\Delta - 1)$- y non- l o $\Delta - 1$ il n n ll l o v , y h. T n, $n = \sum_{k=0}^{h}(\Delta - 1)^k$ ol . L $P = \{v_1, v_2, \ldots, v_n\}$, n l $ch_{v_i}^S(x)$ x- il o v_i in S. L v_l o wi o $\lceil h/ \ \rceil$, $v_m = prt_{v_l}$, n c n in i yin $v_l = ch_{v_m}^S(c)$.

W o By n in o $BF = \{v_l\}$ n o o $CF = \emptyset$. Fi , w xi $(\mathcal{B},\mathcal{C})$- l on ion ρ i yin ollowin on i ion $\mathcal{A}$. (W ow xi n o ρ in l o i oo .)

on i ion $\mathcal{A}$: 1) Fo ny $v_i \in P - \{v_m, v_l\}$, ll lin in i n o v_i v iff n olo ,) l in E_l n E_m o ll lin in i n o v_l n v_m iv ly, $\Big\{Color(e)\Big|e \in (E_l \cup E_m) - \{(v_l, v_m)\}\Big\} = CSET$.

No i , in ρ, w v olo i i n o lin (v_l, v_m), lin in i n o v_l o v_m olo lin (v_l, v_m).

W on i x ion o ρ By n in o v_l v non l y o , i , ll o v o ly. T x ion i x ion o ρ in ll o non l y. T , y on ion ρ' w ny o v i $spec(v)$. in , in ρ, w v olo i i n o lin (v_l, v_m), lin in i n o v_l o v_m olo lin (v_l, v_m). T , xi $v_{a_1} \in \{v_l, v_m\}$ n $v_{a_2} \in N_{v_{a_1}} - \{v_l, v_m\}$ olo o lin (v_{a_1}, v_{a_2}) in ρ' i iff n o in ρ. Sin ll lin in i n o v_{a_2} v iff n olo in ρ, i o v_{a_2} i Δ, xi n i o v_{a_3} o v_{a_2} olo o lin (v_{a_2}, v_{a_3}) in ρ' i iff n o in ρ. In i il w y, w n on n o lin , $(v_{a_1}, v_{a_2}), (v_{a_2}, v_{a_3}), \ldots, (v_{a_{f-1}}, v_{a_f})$, w o olo in ρ' iff n o o in ρ. No i o v_{a_f} i no Δ. on n ly, o i $v_{a_{f-1}}$, v_{a_f}, r_{a_{f-1},a_f}, o $r_{a_f,a_{f-1}}$ in ρ i iff n o in ρ'. Sin o v_{a_f} i no Δ n y S i o l $(\Delta - 1)$- y , v_{a_f} i oo o o l o . T , i n o $v_{a_{f-1}}$ (o v_{a_f}) o By n in o v_l i $\Omega(h) = \Omega(\text{lo}\ n)$. Sin ρ i $(\mathcal{B}, \mathcal{C})$- l on ion wi i $(\tau(n), \mu(n))$, o w o i n o v_l i o n $\tau(n)$ o no n ny . T o , $\tau(n) = \Omega(\text{lo}\ n)$.

In ollowin , w ov xi n o $(\mathcal{B}, \mathcal{C})$- l on ion i yin on i ion $\mathcal{A}$. To ov i , w on y $T = (P', L')$ ollow . L $P' = P \cup \{u_1, u_2, \ldots, u_{\Delta-1}\}$, w $u_i \notin P$ $(1 \le i \le \Delta - 1)$, n L' i n ollow (S Fi . 3):

1. $ch^T_{v_i}(x) = ch^S_{v_i}(x)$ $(i \neq m)$

. $ch^T_{v_m}(x) = \begin{cases} u_1 & (x = c) \\ ch^S_{v_m}(x) & (x \neq c) \end{cases}$

3. $ch^T_{u_1}(x) = \begin{cases} u_{x+1} & (x < c) \\ v_l & (x = c) \\ u_x & (x > c) \end{cases}$

Fo y T, l $\mathcal{Q}_T = p_1, p_2, \ldots$ l , n l $e_T = \sigma_0, \sigma_1, \ldots$ n x ion y l $\mathcal{Q}_T$ in $BF = \emptyset$ n $CF = \emptyset$. T n, in

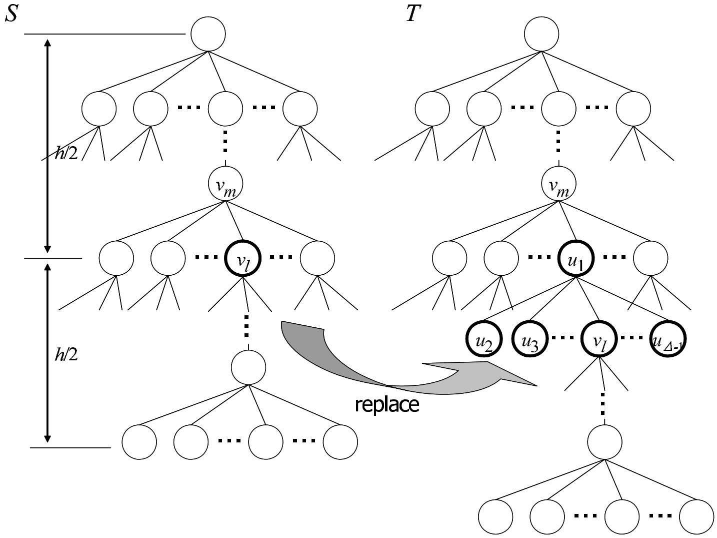


Fig. 3. Two systems

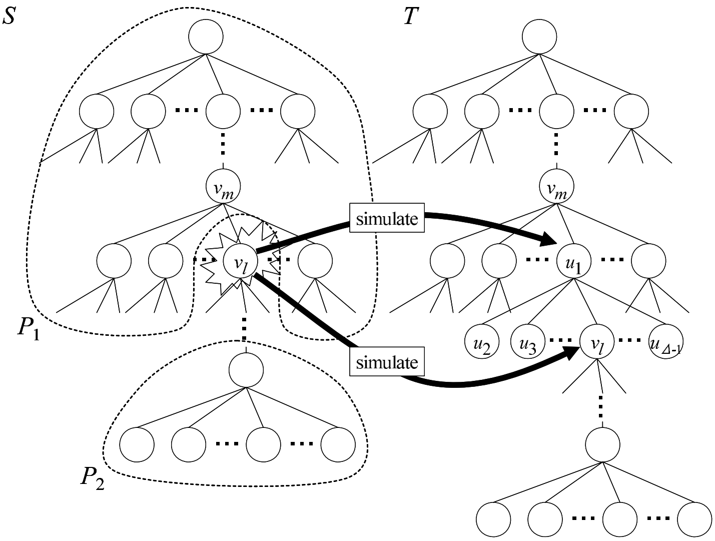


Fig. 4. Byzantine process v_l in S behaves as u_1 for P_1 and as v_l for P_2

e_T, xi on ion σ_s ny o v i $spec(v)$ n n v n ny σ_s.

N x , o y S, w on l $\mathcal{Q}_S = q_1, q_2, \ldots$ n n x ion $e_S = \rho_0, \rho_1, \ldots$ in $BF = \{v_l\}$ n $CF = \emptyset$. W n ini i l on ion ρ_0 o $view(\rho_0, v_i) = view(\sigma_0, v_i)$ ol o ny v_i. W on l $\mathcal{Q}_S$ in w i v_i $(i \neq l)$ x in o in $\mathcal{Q}_T$, n v_l x i i ly o o n i o o v_l. In e_S, By n in o v_l i l vio o u_1 n v_l in e_T (S Fi .). T i , i $q_\alpha = v_l$, $q_{\alpha+1} = v_m$, n $q_{\alpha+1}$ i k- o v_m in e_S, v_l n o i l n lin i r_{v_l,v_m} o $\rho_\alpha|v_l = \sigma_\beta|v_l$ n $view(\rho_\alpha, v_m) = view(\sigma_\beta, v_m)$, w σ_β i on ion $p_{\beta+1}$ i k- o v_m in e_T. An , i $q_\alpha = v_l$, $q_{\alpha+1} = ch^S_{v_l}(x)$, n $q_{\alpha+1}$ i k- o $ch^S_{v_l}(x)$ in e_S, v_l n o i l n i o i o $\rho_\alpha|v_l = \sigma_\beta|v_l$ n $view(\rho_\alpha, ch^S_{v_l}(x)) = view(\sigma_\beta, ch^T_{v_l}(x))$, w σ_β i on ion $p_{\beta+1}$ i k- o $ch^T_{v_l}(x)$ in e_T. T n, ny v_i $(i \neq l)$ n o i l n i o i in w y in e_T. By ni ion o e_T, xi on ion ρ_t , ny o v_i $(i \neq l)$ n v n i ρ_t. Sin A i $(\mathcal{B},\mathcal{C})$- l - ili in o o ol wi i $(\tau(n), \mu(n))$, y v n lly $(\mathcal{B},\mathcal{C})$- l on ion $\rho_{t'}$ wi i $(\tau(n), \mu(n))$. L t'' in $t'' \geq$ x$\{t, t'\}$ n $q_{t''+1} \in Ch_{v_l}$. By ni ion o t'', $\rho_{t''}$ i $(\mathcal{B},\mathcal{C})$- l on ion wi i $(\tau(t), \mu(t))$.

Sin olo o lin x o (v_m, v_l) in $\rho_{t''}$ o in σ_s, $\rho_{t''}$ i on i ion o $\mathcal{A}$.

In ollowin , w ow $\rho_{t''}$ l o i on on i ion o $\mathcal{A}$. L $E_S(v)$ n $E_T(v)$ o ll lin in i n o v in S n in T iv ly.

L $COL(\rho, E)$ o olo lin in E v in on ion ρ. L $U = COL(\rho_{t''}, (E_S(v_l) \cup E_S(v_m)) - \{(v_l, v_m)\})$, $V = COL(\sigma_s, E_T(v_l) - \{(v_l, u_1)\})$, n $W = COL(\sigma_s, E_T(v_m) - \{(v_m, u_1)\})$. By ni ion o $\rho_{t''}$, $U = V \cup W$ ol . T n, in o v_l n v_m in T Δ, l in χ_1 n χ_2 olo o (v_l, u_1) n (v_m, u_1) in on ion σ_s, $V = CSET - \{\chi_1\}$ n $W = CSET - \{\chi_2\}$ ol . Sin $\chi_1 \neq \chi_2$, $V \cup W = CSET$ ol , n , $U = CSET$ ol . T o , $\rho_{t''}$ i on on i ion o $\mathcal{A}$. □

Si il ly, w n ollowin o .

Theorem 4. $\Delta =$ $(\mathcal{B}, \mathcal{C})$ Δ $(\tau(n), \mu(n))$ $\tau(n) = \Omega(n)$ n

5 Impossibility of Link-Coloring Under the Distributed Daemon

In i ion, w on i l- ili in lin - olo in o o ol n i i on. T i i on llow wo o o o o x i ion i l n o ly, w il n l on o no . T , i i on i lly o i l o l. How v in i ion, w ow , o ny l- ili in lin - olo in o o ol, infl n o By n in o x n o o w o i n o By n in o i $\Omega(n)$ v n w n i n i ily l n o olo . T i low o n l i li ion o n l on i on l o in l on in n in By n in l wi on n on in n i .

Theorem 5. n A A $(\mathcal{B}, \mathcal{C})$ $(\tau(n), \mu(n))$ $\tau(n) = \Omega(n)$

W olo o lin (u, v) i in y o u, v, $r_{u,v}$, n $r_{v,u}$. A A i $(\mathcal{B}, \mathcal{C})$- l- ili in o o ol wi i $(\tau(n), \mu(n))$.

W on i i i y $S = (P, L)$ in o o lin : L $P = \{v_1, v_2, \ldots, v_n\}$ n $L = \{(v_i, v_{i+1}) | 1 \leq i \leq n - 1\}$, w v_1 i oo o . L $BF = \{v_1, v_n\}$ n $CF = \emptyset$.

W on i n x ion $e = \rho_0, \rho_1, \ldots$ y l $\mathcal{Q} = Q_1, Q_2, \ldots$, w $Q_i = P$ o ny i. W $view(\rho_0, v_2) = view(\rho_0, v_3) = \cdots = view(\rho_0, v_{n-1}) = s_0$. Sin o $v_2, v_3, \ldots, v_{n-1}$ x , $view(\rho_1, v_3) = view(\rho_1, v_4) = \cdots = view(\rho_1, v_{n-2}) = s_1$. T n, w By n in o v_1 n v_n n o $view(\rho_1, v_2) = view(\rho_1, v_{n-1}) = s_1$ n ol . T i i li $view(\rho_1, v_2) = view(\rho_1, v_3) = \cdots = view(\rho_1, v_{n-1}) = s_1$. W n By n in o x i il ly, w v $view(\rho_i, v_2) = view(\rho_i, v_3) = \cdots = view(\rho_i, v_{n-1}) = s_i$ o ny i. I ow , o ny ρ_i, $Color((v_2, v_3)) = Color((v_3, v_4)) = \cdots = Color((v_{n-2}, v_{n-1}))$ ol . T , l in $h = \lceil \frac{n}{2} \rceil$, o v_h nno i y $spec(v_h)$. Sin i n o v_h o ny By n in o i $\Omega(n)$, $\tau(n) = \Omega(n)$. □

6 Conclusion

In i , w on i l- ili in lin - olo in o o ol ili n o By n in l in oo n wo . Fi , n n l on, w o o l- ili in lin - olo in o o ol ili n o By n in l . T o o ol $\Delta + 1$ olo , w Δ i xi o n wo , n n ny non l y o v i i wi in o n n n v n i i v no By n in n- o wi i n o wo o l . F o , w ow , o ny l- ili in lin - olo in o o ol in Δ olo , on in n i - o $\Omega($lo $n)$ i $\Delta \geq 3$, n $\Omega(n)$ i $\Delta =$, w n i n o o . T , o o o o o ol in ini li y in n o olo o i vin l on in n o By n in o wi on n on in n i . N x , n i i on, w ow , o ny l- ili in lin - olo in o o ol, on in n i o $\Omega(n)$ v n w n i n i y n o olo . T i low o n l i li - ion o n l on i on l o in l on in n in By n in l wi on n on in n i .

Acknowledgement

T i wo i o in y JSPS, n -in-Ai o S i n i ((B)( )1530001), n “T 1 n y n o x ll n P o ” o Mini y o ion, l , S o , S i n n T nolo y, J n.

References

1. E. Anagnostou and V. Hadzilacos. Tolerating transient and permanent failures. *Lectures Notes in Computer Science, Vol 725 (Springer-Verlag)*, pages 174–188, 1993.
2. A. Arora and H. Zhang. Lsrp: Local stabilization in shortest path routing. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pages 139–148, 2003.
3. J. Beauquier and S. Kekkonen-Moneta. Fault-tolerance and self-stabilization: impossibility results and solutions using self-stabiling failure detectors. *International Journal of Systems Science*, 28(11):1177–1187, 1997.
4. J. Beauquier and S. Kekkonen-Moneta. On ftss-solvable distributed problems. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, page 290, 1997.
5. E. W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

6. S. Dolev. *Self-Stabilization*. MIT Press, 2000.
7. S. Ghosh and A. Gupta. An exercise in fault-containment: self-stabilizing leader election. *Information Processing Letters*, 59(5):281–288, 1996.
8. S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 45–54, 1996.
9. S. Ghosh and S. V. Pemmaraju. Tradeoffs in fault-containing self-stabilization. In *Proceedings of the 3rd Workshop on Self-Stabilizing Systems*, pages 157–169, 1997.
10. A. S. Gopal and K. J. Perry. Unifying self-stabilization and fault-tolerance. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pages 195–206, 1993.
11. D. A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in o(log log n) rounds. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 278–285, 1997.
12. Y. Katayama and T. Masuzawa. A fault-containing self-stabilizing protocol for constructing a minimum spanning tree. *IEICE Transactions*, J84-D-I(9):1307–1317, 2001.
13. S. Kutten and B. Patt-Shamir. Stabilizing time-adaptive protocols. *Theoretical Computer Science*, 220(1):93–111, 1999.
14. M. V. Marathe, A. Panconesi, and L. D. Risinger. An experimental study of a simple, distributed edge coloring algorithm. In *Proceedings of the 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 166–175, 2000.
15. T. Masuzawa. A fault-tolerant and self-stabilizing protocol for the topology problem. In *Proceedings of the 2nd Workshop on Self-Stabilizing Systems*, pages 1.1–1.15, 1995.
16. H. Matsui, M. Inoue, T. Masuzawa, and H. Fujiwara. Fault-tolerant and self-stabilizing protocols using an unreliable failure detector. *IEICE Transactions on Information and Systems*, E83-D(10):1831–1840, 2000.
17. M. Nesterenko and A. Arora. Tolerance to unbounded byzantine faults. In *Proceedings of 21st IEEE Symposium on Reliable Distributed Systems*, pages 22–29, 2002.
18. A. Panconesi and A. Srinivasan. Fast randomized algorithms for distributed edge coloring. In *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing*, pages 251–262, 1992.
19. S. Ukena, Y. Katayama, T. Masuzawa, and H. Fujiwara. A self-stabilizing spanning tree protocol that tolerates non-quiescent permanent faults. *IEICE Transaction*, J85-D-I(11):1007–1014, 2002.

A Hierarchy-Based Fault-Local Stabilizing Algorithm for Tracking in Sensor Networks

M D i[1], Ani A o[1], Tin Nol[2], n N n y Lyn[2]

[1] Computer Science & Engineering, The Ohio State University,
Columbus, OH 43210, USA
[2] MIT Computer Science & Artificial Intelligence Laboratory,
Cambridge, MA 02139, USA

Abstract. In this paper, we introduce the concept of *hierarchy-based fault-local stabilization* and a novel self-healing/fault-containment technique and apply them in Stalk. Stalk is an algorithm for tracking in sensor networks that maintains a data structure on top of an underlying hierarchical partitioning of the network. Starting from an arbitrarily corrupted state, Stalk satisfies its specification within time and communication cost proportional to the size of the faulty region, defined in terms of levels of the hierarchy where faults have occurred. This local stabilization is achieved by slowing propagation of information as the levels of the hierarchy underlying Stalk increase, enabling more recent information propagated by lower levels to override misinformation at higher levels before the misinformation is propagated more than a constant number of levels. In addition, this stabilization is achieved without reducing the efficiency or availability of the data structure when faults don't occur: 1) Operations to *find* the mobile object distance d away take $O(d)$ time and communication to complete, 2) Updates to the tracking structure after the object has moved a total of d distance take $O(d * log$ network diameter) amortized time and communication to complete, 3) The tracked object may relocate without waiting for Stalk to complete updates resulting from prior moves, and 4) The mobile object can move while a *find* is in progress.

Keywords: Sensor networks, self-stabilization, fault-containment, tracking, distributed data structures.

" v y in i l o v y in l , n in o l n i n in ".

1 Introduction

In i i y , l n o i o o o y . In o y , i o ion o o i n l .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 299–315, 2005.

F l - on in n o o on n n i on n wi v n in i o - ion o l yon ll ion. x ly w i n y “ ll” i n olyno i l o , n o v i y . P - vio ly, ion i o il w n in o n o o o . T i i onv ni n o x in io n o o o l in x ion o n l o i lon l o i o no in o o o o o i i .

Hi i v lon n i o on n wo o o o o ili i n o i n n l l o o ol . Fo x l , Aw n P l ’ in [6] i i i i o y v o o lo ion in o - ion o o il o j . T i o y v w o o o i y o o i lly n ion l i o i w i o i i l v l o i y w on i l o in inin in o ion o l ion o n wo .

Ano x l o o i lly n i i in i i y l in on . In y , ll o ivi in o l v l o l . o l on in - lo o on no o i lly n v n l . T l v l o l n i ion in o l v l on l , in on inin lo o on no , n o on.

U in i ion l ni ion o ion i , l o in l l v l o o in x ion o i y- l o i i o l o in l l v l n o . A l , l - on inin l o i wo l v o v n o ion o in o ion yon n w o i i olyno i l on ion i on . T i in o l v l- lin l - on in n i no lw y o i l . In , i n l o n ion i n l - on in n in o i y. P ion i wo l n in o l v l w o o , n l - on inin l o i wo l i o no o l o n ll n o l v l in i y. In i , w n no ion n i o v l n l o i o in o il o j .

B o n ow o li ion in o il o in , ll l l ony, n ili y on x , in o o il o j n ly - iv i ni n n ion [6, 0, , ,1]. T DA PA N wo So w T nolo y (N ST) o o in ll n o l in wi l n o n wo , n v l o v liv ll- l (100 no n wo) in on ion : - v in wi on - n on oll v n ono o i ow in [1], n ion, l i ion, n in o v io in , on n , on in [3].

In i ion o o o ni i y ovi o in o o j , wi - l n o n wo l o i o i ion l ll n . S n o no n y- on in , n l o i i x iv o ni ion n - l in y in y ow i ly. S n o n wo l - on , lo n o ion n no il n , no n lo yn ony n o n i y [1]. n- i in n n i in i l n n n o n wo o l l- lin . Mo ov l- lin o l i v l - on in n o v n l in on ion o n wo o on in in n i n wo n i in lo l o ion, w in n y o no n in v il ili y o in vi .

Contributions. nov l on i ion i o n i y- l- lin / l - on in n ni n n on on wi n l o i o in in n o n wo , w i w ll STALK (S ili in T in viA L y linK). To i v l ili y, STALK loy i i- l in . T in i i o on n n lyin i i l i ionin o n o n wo in o l , o o- vi y l- ili in l o i i in [1]. W i l n o in y n o wo lo l ion , . n . . . . T ow ion n l o ow o n w lo ion o o il o j o in in ly i l v l o i y n onn o o i in l . T in ion l n n y o j . S in in l o low l v l n li o in in ly i l v l . D i ow n in o on n ly, w o l ov o ion lly y in i ly- o n i o in w n ion o .

STALK i i y- l - on inin , v n in o ion o l in in yon ll n o l v l in i y. S in o n i ily o , i i i i ion in i n wo o o ion l o ion i , n in o l v l (n y n lyin i y) w l v o . W i v l - on in n y lowin o ion o in o ion l v l o i y n lyin STALK in , n lin o n in o ion o y low l v l o ov i i in o ion i l v l .

STALK ovi oo lo li y n ; . . . o o j in o i n d w y i $O(d * logD)$ i n o ni ion (wo) o in , w D i n wo i . In ll v ion o o [11] w l o i . . o ion in in . A n o ion invo o i n i o in o in in ly i l v l o l in i y n il i n o n o on in . n i o n , n o ion ollow i o i l o o il o j . In ll v ion w l o ow . . invo wi in i n d o o il o j i $O(d)$ wo o o j n w n no l o , o o i vin l - on in n o no in o l xi y o in o n in . F o , w ow STALK i v l in o on in o ly ovin o j y llowin on n in n n in o ion . Fo on , w o ll v ion [11] o l n in on n on in o ion o STALK n l - on in n .

Related work. T i o loyin i i l o i v- in l ili y o in n x n iv ly . T i o in i l in o ion y o o i i o n n ov in l iv ly i oin - o- oin n wo w inv i in [6]. In [6], i y o - ion l i o i i on o l v l l l i o y n l no o n o il o j wi in ^{l} i n o i l . T o ni ion o o n o n o j d w y i $O(d * log^2 N)$ n o ov o i - n d i $O(d * logD * logN + log^2 D/logN)$ (w N i n o no n D i n wo i). How v , o olo y n , no il- , n i lo l o y in ion l i o i - n on non-lo l l in o [5] on ov o .

In [9], in o l i on i o o i n wo o l i il o o , n o o l xi y i il o o i i v . How v , in in in i no v il l in ov o o il o j n o o n in o il o j i only i li i ly n . T i l o i i l o no l - ol n . P [, 3] on n wi non- ili in ol ion o on l o ni ion y n o il In n P o o ol, no n o n wo . A lo ion vi o o n wo i i in [1] n ovi iv wo n v o n ovi o l - ol n , o i i no l - on inin .

T n wo on l - ili in , o no l - on inin , in l o i [15, 1 , 10]. T i i ow o o ol [15] i on l o i ﬀ o i in o l —w n o j ovin n o o l i-l v l i y o n y y l o nonlo l . T o- o ol in [10] o no x loi i y i n no l l o l n wo . In [1], in i y o lo ion v , ili in lo ion n n o o ol i n . How v , o o ol in [1] o no n- lo li y o n . In [1] no l - ili in l o i in i i o olv o l lo o in i n , o i oo i no l - on inin .

F l - on in n o l - ili in l o i in n l iv ow- in in [13, 19, ,], o non o l o i i y- on o l - on in n . T no ion o l on in n wi in on- x o ili ion w o li in [13]; l o i w o o o on in - o ion o in l no in ili in nnin o o ol. In [19] l - on in n o By n in no w i in inin ilo o n olo in l o i ; i wo i n o on in ion o on n n i oo li i in o o l in n o in w lo li y i no on n . In [ ], o o o ol w o o o on- in o v l v i l in n o o ion , o o ol llow o lo l o ion o in n l o o ol v i l . Ano o o ol i v l -lo l ili ion in o o in w n in [ ]. To i v l - on in n o o ol ivil on in n ion w on n i n l -in ol n o - ion .

Organization of the paper. A n in o l in n x ion, w n i ion o Stalk n ni ion o i y- l-lo li ion in S ion 3. In S ion , w n ov o ion. F l lo l ili ion ion o in i in S ion 5. Fin lly w on l o in S ion 6. Fo on , w l il oo o T ni l o [11].

2 Model

W on i n o n wo on i in o l i l n o lo ion . n o lo ion l y o o (o i ly) l i l o wi i n i o P. In i , onv n ion, i n j o o i n i , n $i.x$ o v l o v i l x i.

W no lo ion o o i wi $loc(i)$ (n o onv ni n o lo ion o o I wi $loc(I)$). T li n i n w n lo ion o i n j i no y $dist(i,j)$.

Hierarchical partitioning. A i i l i ionin o o ov lo ion . on i wi l v l 0 o MAX o ll o P. Fo o i w n :

1. $lvl(i)$, l v l o o i in ,

. $h(i)$, i' n in (o onv ni n , w n $h(i)$ o i i $lvl(i) = MAX$),

3. $h^n(i)$, i n , n $h(i)$ i $n = 1$ n $h(h^{n-1}(i))$ o wi ,

. $children(i)$, i' il n in . W on - o-on o on n

w n l v l 0 o in n n o lo ion . Fo lo ion v w no l v l 0 o i in v $proc_0(v)$. W l o o ny i $lvl(i) > 0$, i' lo ion $loc(i)$ i l o $loc(j)$ o on o i il n j.

T i i ionin yi l . Fo i $lvl(i) = k{+}1$, $0 \leq k < MAX$, $children(i)$ o o l C l v l k w o l , $head(C)$, i i. $Radius(C)$ i xi i n o $head(C)$ o ny o in C.

N x w in o y i n i o l ion. Fo l v l 0 o i, j, $i \neq j$, $j \in nbr(i) \iff dist(i,j) \leq 1$. Fo l v l $k > 0$ o i, j, l o l v l $k-1$ l C_i n C_j, i n j n i o i C_i n C_j on in wo o n i o .

Geometry assumptions. W x ollowin ion o i - i l i ionin :

1. W n l on n $r \geq 3$ o no l il ion o ; i o l v l l l i l r^l,

. W n l xi l i on n $m \geq /\sqrt{3}$ o o n i o l v l l l o o mr^l,

3. W n l ini l on n q i yin $\frac{2m+r-1}{r-1} \leq$ $q \leq$ m o i lo ion in l v l l l o n qr^l .

T on in i ly o n , ω, on n o n i o ny l v l $l > 0$. T y l o i ly , o $l > 0$, i n w n wo n i o in l v l l o i wi in r^{l-1}- o- mr^{l-1}, n i n w n l v l l o n i il n in i y i o mr^{l-1}. T i l in o no n ily i ly ni o ilin o n wo , ii o l l v l no i o . T n wo i , D, i xi i n w n ny wo lo ion in n wo . no in n wo i loy wi $O(MAX)$ o w $MAX \leq log_r D$.

An x l o l in o y wi $r = 3$ n o n in S ion . i i l i ionin on in n li y in i i n l -lo l ili in l in o o ol, Loci [1].

3 System Specification

H w i i ion o y .

Mobile object. T o il o j **Evader** i x ly on n o lo ion. W o l **Evader** in **object** n **no_object** in o : An **object**$_i$ o ll o i in o j ' n lo ion n **no_object**$_j$ o o ll o lo ion . W n ovin , o j non ini i lly ov o n i o in lo ion.

STALK. Stalk on i o wo , **Tracker** n **Finder**, n in Fi 1. **Tracker** in in in y o in o il o j in o ion o in o **object** n **no_object** in . **Finder** n w li n **find** y o in **found** o il o j ' n lo ion. **Finder** wo l y **Tracker** o lo ion in o ion o **cpq** n **Tracker** wo l n w wi **cpointer** on .

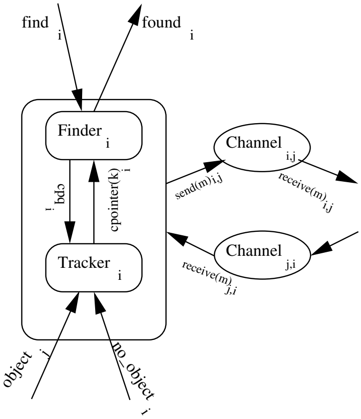


Fig. 1. Stalk architecture at process i

STALK i i l n i i iv ly y in ivi l o o ni - in o nn l . o i o v o i own lo l i , v n ll o . W o no i yn oni ion o o .

Channels. W o ni ion ion o (o i ly) l i- o nn l **Channel**$_{i,j}$ w n ny wo o i n j. S nn l in **send(m)**$_{i,j}$ o n o i n **receive(m)**$_{i,j}$ o iv j. T o o n in o **Channel**$_{i,j}$ i $dist(i,j)$, n in n o l i ov o nn l y o $\delta * dist(i,j)$ i w δ i nown l y o .

Fault model and tolerance specification. P o n ff o i- y o ion. T l y o ny i n in ny ni n n o . nn l y ff l o , n , li- , o lo .

W y y i iff in o n i y y - v n lly ov o on i n , o w i i ion i i . In S ion w i on i n o o i l n ion. A ion o n o iv n y i ini n o o w o n o i v on i n o y . Fo wo n i l l ion l v l o “ ” o i o n ; l i in l v l l o ff n i l v l l l n n i i i r^l. W n o y o w i o i o o . A ili in y i i i n wo i o ili ion o n y n ion o ion i n y i .

Complete system. T o l y i o o i ion o ll nn l , **Evader** n STALK.W i y l -lo l ili in o on- i n . S in o on i n w n i i o j ov d i n , o i i n wo o in - i $O(d * log(D))$. (n n i n l in o n on n in n n in i in T o [11].)

4 Tracker

H w i ow **Tracker** in ov , in o il o j o no lo n il o l . In [11], w l x i i ion n llow o j o lo w il ff o i vio ov ill i lin o .

U o in i l n y wo lo l ion , ow n in . T ow ion n l n w o ow o in in ly i l v l o l in i y n onn o o i in l o l v l. T in ion l n ol n y o il o j in o low l v l .

A i i l i ionin o n wo in vi ly l in l i-l v l l - o n i : v n o wo o n i o y i on in in iff n l ll l v l (x o) o i y. I o w o lw y o ow n in o i l , ll ov n o o j n o o l i-l v l l o n y o l l in wo o o ion l o i o n wo n i n o ov . To olv i " i in " o l , w llow on l v l in o in . A o o ion lly onn o o i in l wi l l lin o n i o in o n y o in lin o i n in i y.

To i l n **Tracker**, o *i* in in il oin *c*, n oin *p*, ow i *gtime*, n in i *stime*. In ini i l , $i.c = i.p = \bot$ n $i.gtime = i.stime = \infty$ o ll *i*. W o ow n in on n *g* n *s* i y:

$$s \geq 10.5\delta m \quad (1)$$

$$\frac{s + \delta m}{r} < g \leq s - \delta m \quad (\ )$$

A ow o in i i *i* o $g * r^{lvl(i)}$ o $s * r^{lvl(i)}$ i iv ly. T v l o i o n o i y i n on o wo l l ion in S ion . n l - on in n oo in S ion 5.

Signature:	State:
Input: **object**$_i$	$c \in P \cup \{\bot\}$, initially $\bot$
no_object$_i$	$p \in P \cup \{\bot\}$, initially $\bot$
cpq$_i$	$gqack \in P \cup \{\bot\}$, initially $\bot$
receive$(msg^*)_{j,i}, j \in P$	$gnbrquery \subseteq P$, initially $\emptyset$
	update, a Boolean, initially *false*
Output: **send**$(msg^*)_{i,j}, j \in P$,	$gtime \in \Re$, a timer, initially ∞
cpointer$(j)_i, j \in P \cup \{\bot\}$	$stime \in \Re$, a timer, initially ∞
$^*msg \in$ {gquery, ack_gquery, grow, shrink}	$now \in \Re$, a timer indicating current time

Fig. 2. Signature and state of **Tracker**$_i$

Tracker$_i$ n w **cpq**$_i$ in (n in o ion o **Finder**$_i$) wi **cpointer**$(i.c)_i$ o , ovi in v l o i il oin . T **send** n **receive** o ow n in x l in in il low o o *i*.

4.1 Grow Action

A ow o oin o n w lo ion o o j .

I *i* i l v l 0, o j i lo ion *i*, n *i*' il oin *c* o no oin o i l , n *i* o l o in y in *c* o *i* n in i ow i , *gtime*, lin **grow** o n w n *gtime* x i .

```
Input: object_i
eff:   if $c \neq i \;\wedge\; lvl(i) = 0$ then
           $c := i$
           $gtime := now + g$

Output: send (gquery)_{i,j}
pre:   $j \in gnbrquery$
eff:   $gnbrquery := gnbrquery - \{j\}$
       if $gnbrquery = \emptyset$ then
           $gtime := now + g * r^{lvl(i)}$

Input: receive (gquery)_{j,i}
eff:   if $p = h(i)$ then
           $gqack := j$

Output: send (ack_gquery)_{i,j}
pre:   $gqack = j$
eff:   $gqack := \perp$

Input: receive (ack_gquery)_{j,i}
eff:   if $c \neq \perp \;\wedge\; p = \perp$ then
           $p := j$

Output: send (grow)_{i,j}
pre:   $now = gtime \;\wedge\; c \neq \perp \;\wedge$
         $((j = p \;\wedge\; p \in nbr(i)) \;\vee$
          $(j = h(i) \;\wedge\; p = \perp))$
eff:   if $p = \perp$ then
           $p := h(i)$
       $gtime := \infty$

Input: receive (grow)_{j,i}
eff:   $c := j$
       if $lvl(i) = MAX$ then
           $p := i$
       if $p = \perp$ then
           $gnbrquery := nbr(i)$
```

Fig. 3. Grow actions at process i

I i i ov l v l 0 n iv **grow** , i i c oin o n , *gtime* lin **grow** o n o i o iv n . i l o n **gquery** o i n i o o i in i l o n i o . T in llow o on l l lin l v l. A n i o j iv **gquery** n n **ack_gquery** i j i on in n i n' l y l l lin oin in o j, i. ., i $j.p$ oin o i own l , $h(j)$. I i iv n **ack_gquery** o j n i p o oin o j, in ion o in l l lin j.

W n *gtime* x i , i c i ill non-$\perp$, nin no n w il i' ow i w o n in own, n **send (grow)** i o o x n in . I $i.p$ oin o n i o j n ow i n o j, in in l l lin . wi , i $p = \perp$, i p o oin o i own l $h(i)$ n n **grow** o $h(i)$, o in ow on l v l in i y. In i *gtime* i o ∞, n i' ol in in in i o l .

I **grow** i iv i i l y n in in o i MAX l v l o , n i o no o ow (i i l y on in).

4.2 Shrink Action

A in l n ol , n o in .

I i i l v l 0 n non-$\perp$ il oin , o il o j i no i' lo ion, n i ov i l o l o in . I i il oin c o $\perp$ n in i *stime*, lin **shrink** o n on x i ion o *stime*.

```
Input: no_object_i                              Input: receive(shrink)_{j,i}
eff:   if $lvl(i) = 0 \;\wedge\; c \neq \bot$ then    eff:   if $c = j$ then
           $c := \bot$                                        $c := \bot$
           $stime := now + s$                                 $stime := now + s * r^{lvl(i)}$

Output: send (shrink)_{i,j}
pre:   $now = stime \;\wedge\; c = \bot \;\wedge\; j = p$
eff:   $p := \bot$
       $stime := \infty$
```

Fig. 4. Shrink actions at process i

I i iv **shrink** o no o j, i o w i il oin c oin o j (c i no oin o j; i y v n o oin o o on n w). I $c = j$ n i ov i l o y in c o ⊥ n n i in i , lin **shrink** o n o i n p. wi , i $c \neq j$, i i no , n in in ion l n only woo n no n i in .

W n *stime* x i , i c i ill ⊥, nin no n w onn i w il *stime* w o n in own, i n **shrink** o i n p in n n p o ⊥.

Example. Fi 5 i l in . T i n oin in o l v l l , w i oin o on o i i y il n, l v l 1 l . T l l l lin o no l v l 1 l oin o l v l 0 l w o j **e** i lo . D woo i no y o .

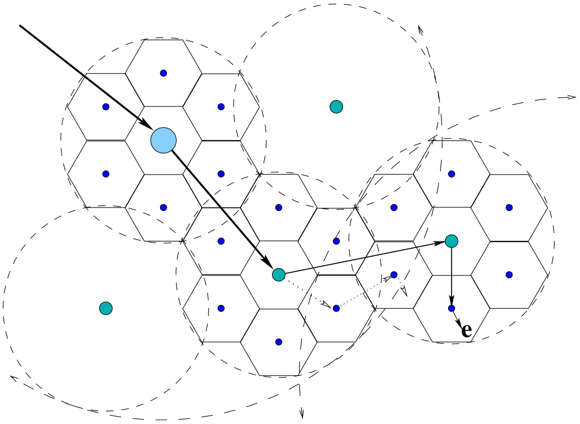


Fig. 5. Tracking path example

4.3 Correctness

H w n y inv i n n n on i n o y .

In n o l , v y o i i I, ollowin v on iion , ll i :

I0. I $lvl(i) = 0$ n **object**$_i$ o n $i.c = i$,

I1. I $i.c \neq \bot$ n on o ollowin ol :

( ) $i.c = i$ n o j i i,

( ) $i.c$ oin o on o i il n in l in i y, o

( ) $i.c$ oin o n i o n $i.p$ oin o i n in l in i y,

I . I $i.p \neq \bot$ n i $i.c \neq \bot$ o i i x in in ion n will n **shrink** o $i.p$,

I3. T l: i $i.c \neq \bot$ n $i.p \neq \bot$ o i i x in ow ion n will n **grow** o i o iv n ,

I . I $i.c \neq i$ n $i.c \neq \bot$ n $(i.c).p$ i i i o $\bot$. In l **shrink** o $i.c$ i in n i o i. □

A i n $\{i_x, \ldots, i_1\}$ w

1. i_1 i l n on in o j ,

. v y o i_1 oin o n x o i il , n

3. I i i ll o in n .

A i in $\{i_x, \ldots, i_1\}$ w $lvl(i_x) = MAX$ n $i_x.p = i_x$.

A i w o l in xi n $i.c = i.p = \bot$ o v y o i no in in .

U in inv i n I i ollow o o ion n x ion in o n ini i l v n lly on i n n on i n lo n ov o o j .

In w v n lo o v n o l i i n y o l x ni ion o in n in n o n l i in n i y in ili y on i ion . D il n o n in T ni l o .

4.4 Work

In o o ov o wo l i , w ow i in o n o n w n ol in i y in l ion i o n ol i (vi in ion o l l lin) o x n o i l . Mo i lly, i ollow o ion on i on n s n g n ol in l n o o - o l v l 0 will no l n on o i l v l l oin o ow in l v l 0 in n w l v l l n n o o ni y o y on o o oin , llowin o i ion o l l lin .

T i llow o on n w (w i ow y o in oin i i y n il i onn o ol) onn o - in ol low l v l o i i n i l o n w o j lo ion o n i o o l i no i l onn o in vi l l lin . In l , n w wo l onn vi l l lin .

W n ov ollowin o .

Theorem 1. d $O(d * \omega mr * MAX)$ $O(d * gr^2 * MAX)$

Proof sketch. T ov onin i li l v l l oin in i o n v y $\sum_{j=1}^{l-2} qr^j$ i n o i o l l lin ll l v l low l (no qr^l i ini i n w n wo non-n i o in l v l l l). An $O(mr^{l-1})$ wo n $O(gr^l)$ i o i in i l v l l oin i . T o , l i li y n y o , o l v l o l . □

5 Fault-Containment

A o ion o ion o (o n i lly ll) o , o in l i l in l -lo l nn wi in wo o o ion l o ion i . H w i o ion ion n lin l -lo l ili ion o .

T o l in ion n i nly ini i . Fo x l , w n o ion o in i i y l , i l v l o o , n w l y low xi , in ion. I “ ow ” low l v l l in “ in in ” o l v l , l n o o n i . Fo l - on in n , ow ion low l v l on in in ion .

Si il ly, ow ion n i nly ini i . on i wi no o j i l . T o o o o i , n w o no l o o j , ow ion. I “ in in ” o low l v l l in “ owin ” o l v l , l n on in n i n wo . T in low l v l on in ow .

T ov i n o i y ivin io i y o ion wi o n in o ion in ; ion o low l v l iv- il ov on i l v l . W i v i y l yin in / ow o lon io l v l o o x in ion in . T i w y, o ion ion o in o low j o l l y n n i nly ini i o ion ion ; i y- l -lo l ili ion i i v . W no l n y i o y l yin i on- n o o o ni ion l y o i l v l n o no ff li y o in .

Stabilization. H w n o ion ion o - li in - in inv i n I in o n i ily o .

```
Internal: start-grow_i
pre: $(c \neq \bot \ \wedge \ p = \bot \ \wedge$
        $gtime \notin [now, now + g * r^{lvl(i)}])$
eff: if $lvl(i) = MAX$ then
         $p = i$
     if $p = \bot$ then
         $gnbrquery := nbr(i)$

Internal: start-shrink_i
pre: $(c = \bot \ \wedge \ p \neq \bot \ \wedge$
        $stime \notin [now, now + s * r^{lvl(i)}])$
      $\vee \ [p \in nbr(i) \ \wedge \ c \in nbr(i)]$
eff: $c := \bot$
     $stime := now + s * r^{lvl(i)}$
```

Fig. 6. Starting grow/shrink at process i

```
Output: send (heartbeat)_{i,j}
pre:  $now = next \ \wedge \ j = p$
eff:  $next := now + b * r^{lvl(i)}$

Input: receive (heartbeat)_{j,i}
eff:  if $c = \bot$ then $c := j$
      if $c = j$ then $timeout :=$
        $now + (b + 2\delta m/r) * r^{lvl(i)}$

Internal: timeout_expire_i
pre:  $now = timeout \ \wedge \ c \neq \bot \ \wedge \ c \neq i$
eff:  $c := \bot$

Internal: heartbeat_set_i
pre: $p \neq \bot \ \wedge \ next \notin [now, now + b * r^{lvl(i)}]$
eff: $next := now + b * r^{lvl(i)}$

Internal: timeout_set_i
pre: $(c \neq \bot \ \wedge \ c \neq i \ \wedge \ timeout \notin$
        $[now, now + (b + 2\delta m/r) * r^{lvl(i)}])$
eff: $timeout := now + (b + 2\delta m) * r^{lvl(i)}$
```

Fig. 7. Heartbeat actions at process i

$I0$ $I1$ $I0$ i li ivi lly y **object** n **no_object** in . T o ion o $I1$ ollow o o in ion w on non-$\bot$ c, p n $gnbrquery$ v i l o $i \in P$. W i $i.c \neq \bot \Rightarrow i.c \in \{nbr(i) \cup children(i)\}$: $i.c$ oin o i n i o o i o o il o i. Si il ly, w i o in o non-$\bot$ $i.p$ v i l o $\{nbr(i) \cup \{h(i)\}\}$ n $i.gnbrquery$ o o $nbr(i)$. T ion on l in l in ovi o wi i n i o i n i o , il n, n l ; o n lo lly n v i l o $\bot$ i i v l o i i iv o in .

I I i v li n no v li il , n I i o i y in $i.c = \bot$ n lin **shrink** o n o $i.p$.

$I3$ I i v li il no n , n **gquery** i n o i' n i o n **grow** i l o n o n o i.

I To o I w n wo i : $next$ o io i lly n in o n n $timeout$ o i o i in il i no i . T o ion ion on n b o l l in n y o , w o io i i y n l o i v l o ni ion o ion. W i b i o n wi s, in i on n :

$$b \geq \; s \tag{3}$$

In i iv ly, i on i ion v o v n n io w iv ly - l in o i in l o n w owin n onn o o i in l.

v y i wi non-⊥ v l n n **heartbeat** o i n v y $b * r^{lvl(i)}$ i y in *next.* v y i i iv **heartbeat** o **grow** o i il , *i.c*, i i *timeout* v i l o $(b + \; \delta m/r) * r^{lvl(i)}$ (i i l o on i o **grow** o v n n io w i o o i x i lin in j i iv **grow** o o in n wly owin). I i iv o j $i.c = \perp$ n i $i.c := j$. wi , **heartbeat** iv o o o n $i.c$ i i no .

I i non-⊥ v l il , i no l , n no iv **heartbeat** in $(b + \; \delta m/r) * r^{lvl(i)}$ i in v l, n $i.c$ i o ⊥.

S ili ion o *next* n *timeout* v i l o o o i n y in i v l wi in i iv o in .

U in o ion ion i ov , w ov in T o , Stalk i l- ili in o on i n , w o l in xi .[1]

Theorem 2. Stalk □

Fault-local stabilization. To ov i y- l -lo l ili ion w iv o n on in i n o ow/ in ion in L 1 n . In l , $l_1 + 1$ n l_2 iv ly low n i l v l : l o only o l v l $l_1 + 1$ o l v l l_2. W ov l on in n y owin o o i in ion , o ion o o l_1 o ion o in o ion l v l $l > l_2$, l vin l v l ov l n o y l . T oo o o l on y o in xi i o ion o low w v o l v l l v ini i i w v o i .

Lemma 1. $l_1 + 1$ l_2 l

$$l = l_2 + \lceil \mathrm{lo} \;\, _r \, \frac{br - b + sr + gr - 2s + 3\delta m}{gr - s - \delta m} \rceil. \qquad \square$$

Lemma 2. l_1 l_2 l

$$l = l_2 + \lceil \mathrm{lo} \;\, _r \, \frac{br - b + sr^2 - gr - \delta m}{sr - gr - 3\delta m} \rceil. \qquad \square$$

[1] In the case where the evader can relocate before updates are completed, the algorithm self-stabilizes to a state where a more general tracking structure exists, as mentioned in Section 4.3.

T i , $l - l_2$, o on in ion o l o ion i in n n o n wo i n i n l vi ow n in i in . In [11] w ovi v l i y i n , w ll n o o $(g = 5\delta m, s = 11\delta m, b = 11\delta mr)$.

Fin lly, ov wo l llow o ov ollowin o :

Theorem 3 (Fault-local stabilization). S L $O(S)$ $O(r^L)$ □

Proof sketch. v n o y ny iff n n io o o - ion, in y ll l o i i o ion o in o ow, y ll n o low wo o o i: 1) i n o o in i il n i ow ,) i n o o in i no il n i in .

In i i l n o in o ion wi in o $O(r^{lvl(i)})$ i n o on in n n in L 1 n i o ion w v on in vio i in o w v wi in on n n o l v l in i y, o $O(r^{lvl(i)})$ i n wo .

T wo o l - on in n i i iv : ion o wo o ll - o iv wo o y . How v , in l - on in n l on n ly o ll o , l - on in n i $O(r^L)$ o i l v l o (l v l L) o in , ivin o $O(r^L)$ i . □

6 Concluding Remarks

W n Stalk, i y- l -lo l ili in in vi o n o n wo . W wo on o i v i y- l lo li y: i i l i ionin n l v l- i o o x ion o ion . T y i i o w i lon o in wi ion' vi w y loyin l i o w n o in n o i l v l o i y. T i w y, o n o low l v l n - o n ov i i in o i l v l wi in on n n o l v l ov l . W il i vin l -lo l ili ion Stalk l o o lo li y o in o ion . Mo ov , y n lin on n ov n on n n o ion Stalk i v l n on in o in o o il o j . T i l oin i i o lly in o T ni l o [11].

Stalk li ion in o in o o il ni n in - / v . A o o ffo o v lo n o n wo vi in DA PA/N ST o , w i l n in STALK on Mi o l o [16]. Fo wo , w x inin o o l o l n o o i y- lo l ili ion ni .

References

1. I. Abraham, D. Dolev, and D. Malkhi. LLS: a locality aware location service for mobile ad hoc networks. *Manuscript*, 2004.
2. I.F. Akyildiz, J. McNair, J.S.M. Ho, H. Uzunalioglu, and W. Wang. Mobility management in next-generation wireless systems. *Proceedings of the IEEE*, 87:1347–1384, 1999.
3. A. Arora and et. al. Line in the sand: A wireless sensor network for target detection, classification, and tracking. *To appear in Computer Networks (Elsevier)*, 2004.
4. A. Arora and H. Zhang. LSRP: Local stabilization in shortest path routing. In *IEEE-IFIP DSN*, pages 139–148, June 2003.
5. B. Awerbuch and D. Peleg. Sparse partitions (extended abstract). In *IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.
6. B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the Association for Computing Machinery*, 42:1021–1058, 1995.
7. Y. Azar, S. Kutten, and B. Patt-Shamir. Distributed error confinement. In *ACM PODC*, pages 33–42, 2003.
8. A. Bar-Noy and I. Kessler. Tracking mobile users in wireless communication networks. In *INFOCOM*, pages 1232–1239, 1993.
9. Y. Bejerano and I. Cidon. An efficient mobility management strategy for personal communication systems. *MOBICOM*, pages 215–222, 1998.
10. M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *Sixth Symposium on Self-Stabilizing Systems(SSS'03)*, 2003.
11. M. Demirbas, A. Arora, T. Nolte, and N. Lynch. STALK: A self-stabilizing hierarchical tracking service for sensor networks. Technical Report OSU-CISRC-4/03-TR19, The Ohio State University, April 2003.
12. S. Dolev, D. Pradhan, and J. Welch. Modified tree structure for location management in mobile environments. In *INFOCOM (2)*, pages 530–537, 1995.
13. S. Ghosh, A. Gupta, T. Herman, and S.V. Pemmaraju. Fault-containing self-stabilizing algorithms. *ACM PODC*, pages 45–54, 1996.
14. M. Herlihy and Y. Sun. A location-aware concurrent mobile object directory for ad-hoc networks. *Manuscript*, 2004.
15. M.P. Herlihy and S. Tirthapura. Self-stabilizing distributed queueing. In *Proceedings of 15th International Symposium on Distributed Computing*, pages 209–219, oct 2001.
16. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS*, pages 93–104, 2000.
17. M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. *ACM Symposium on Principles of Distributed Computing*, 1996.
18. V. Mittal, M. Demirbas, and A. Arora. LOCI: Local clustering in large scale wireless networks. Technical Report OSU-CISRC-2/03-TR07, The Ohio State University, February 2003.

19. M. Nesterenko and A. Arora. Local tolerance to unbounded byzantine faults. In *IEEE SRDS*, pages 22–31, 2002.
20. E. Pitoura and G. Samaras. Locating objects in mobile computing. *Knowledge and Data Engineering*, 13(4):571–592, 2001.
21. B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. Sastry. Distributed control applications within sensor networks. *Proceeding of the IEEE, Special Issue on Sensor Networks and Applications*, August 2003.
22. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.
23. J. Xie and I.F. Akyildiz. A distributed dynamic regional location management scheme for mobile ip. *IEEE INFOCOM*, pages 1069–1078, 2002.

The Quorum Deployment Problem
(Extended Abstract)

S il [1] n o M l wi [2]

[1] Massachusetts Institute of Technology,
Computer Science and Artificial Intelligence Laboratory, Cambridge, MA
sethg@mit.edu
[2] University of Alabama, Computer Science Department, Tuscaloosa, AL
greg@cs.ua.edu

Abstract. Quorum systems are commonly used to maintain the consistency of replicated data in a distributed system. Much research has been devoted to developing quorum systems with good theoretical properties, such as fault tolerance and high availability. However, even given a theoretically good quorum system, it is not obvious how to efficiently deploy such a system in a real network. This paper introduces a new combinatorial optimization problem, the *Quorum Deployment Problem*, and studies its complexity. We demonstrate that it is NP-hard to approximate the Quorum Deployment Problem within any factor of n^{δ}, where n is the number of nodes in the distributed network and $\delta > 0$. The problem is NP-hard in even the simplest possible distributed network: a one-dimensional line with metric cost. We begin to study algorithms for variants of the problem. Some variants can be solved optimally in polynomial time and some NP-hard variants can be approximated to within a constant factor.

Keywords: quorum systems, combinatorial optimization, fault-tolerance.

1 Introduction

T o o on ni o n in l - ol n in i i y i li ion: o o i li l n o no in n wo , n in no ll n o il n il o - ion. T i y i l y wi i o i n in on i n y o li , wi o in in o o in oo . T i n n l -off w n l - ol n o n o o in inin on i n y.

Q o y v lon n (. ., [1, ,3,]) o olv o l o li on i n y. A . . . . . , q, i o no in n wo , n

[*] This work is supported by MURI–AFOSR SA2796PO 1-0000243658, USAF–AFRL #FA9550-04-1-0121, NSF Grant CCR-0121277, NSF-Texas Engineering Experiment Station Grant 64961-CS, and DARPA F33615-01-C-1896.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 316–330, 2005.

i o o , Q, v y wo o in Q l on no . T i , iv n wo o , $q, q' \in Q$, xi o no $i \in q \cap q'$; in ion o wo o i non- y.

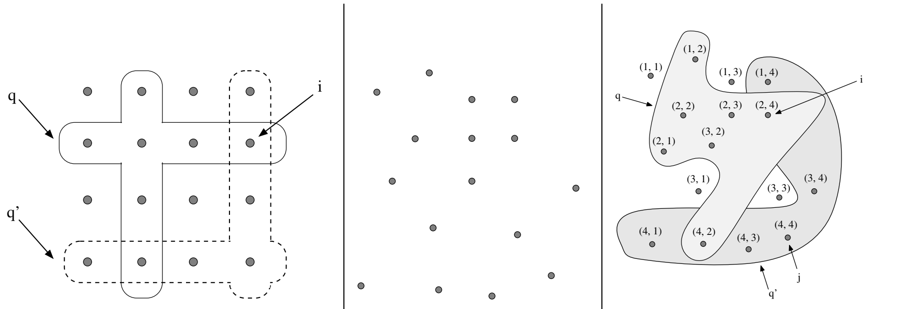
In o o n on i n y o , w n no oo o o i y , i no i o o , y, $q \in Q$, o o i ion; w n no w n o , i on o o , y, $q' \in Q$. Sin wo o , q n q', in o no , w n o ion l n o li o i ion. V i ion on i ni n ly o i l n li ion (. ., [5, 6, ,]). Fo x l , A iy l. i ni o on /w i o y ([9]), n i i l x n o on on l /w i o y ([10, 11]). A i il ni n o l x l ion o o ol (. ., [3, 1]) n o o ol (. ., [13]).

M o o i in l wo on o y o on i o jo i y o no in n wo . In i w y, in ion o y i i i ly n , n o i l l - ol n i i v . (S , o x l , [1 , 1, 15].) Mo n ly, ow v , n - v lo in o o li o y wi v i y o in in o i , i ov v il ili y, on , n o fl xi ili y o on o yn i y . (S , o x l , [16, 1 , 1 , 19, 0].)

Ty i lly, n l o i i n on o wi y o oo o i , n only n i w i n wo no will w i o o o i v low o o n wo o ni ion. T iy l. [1] n F [ ], on o n , v n iff n o ; i l o i in wi n wo , n in o y i o i i n in o n i . Un o n ly, l in o y o no n ily n oo l ol n , v il ili y, . By i n- in o y , n n inin oo loy n , i o - i l o o in o oo n wo o n w ll oo o y o i .

L ill i o in n x l . on i o y in Fi 1( ) (o i in lly i in [16]). T no in n wo n in i wi $\sqrt{n}$ no in ow n ol n. o on i o on ow n on ol n. Any wo o , n, in wo no ; o x l , in Fi 1( ) o q n q' in no i. Fi 1( ) n n i y n wo in wo- i n ion l l n in w i o o o ni ion w n ny wo no i o o ion l o i n w n no . In o o o y , no in l n wo o no in i , in Fi 1( ). T n, no oo on o o o . Fo x l , no i i oo o o q, w il no j i oo o o q'. In n o i l wo l , no i lo o ll no in o i oo .

I o y i ly loy , o o in inin on i n li y o i i iv ly x n iv . I n o o o l ly n l o y – n l wo l n wo – iff n w n n o i l



(a) Grid quorum system

(b) Network of real nodes

(c) Mapping of real nodes to quorum system

Fig. 1. Figure 1(a) represents an abstract quorum system of 16 nodes, where q and q' are two possible quorums, and i is a node in the intersection. Figure 1(b) is an example of a network of nodes, embedded in a two-dimensional plane; communication time between two nodes is proportional to their distance. Figure 1(c) is a mapping of the real nodes in the network onto the abstract nodes in the quorum system

loy n n -o i l loy n n i l . In , w n ow o non- ivi l o y , i o n wo in w i n o i l loy n i n loy n . I wo no onn y n x n iv o ni ion lin (o x l , n wo i o ion lly i ion), -o i l loy n y i no o o ni w il n o i l loy n y no .

In i , w in o , o l o in o y o i lly. W o o i x , n o o n in w n ny wo no i nown in v n . T o o o no , i, o in o y i n o o o n in o v y no in o o . o l i o in in o l no in n wo o no in o i ion, n oi o w i o no o l in n o ion. W n o l o o lly in S ion .

Summary of Results

o l in i i o in w n Q o D loy n P o l n n nno i n ly olv . W no i o on in v - ion o o l , , i olv l in olyno i l i (S ion 3). T n l v ion o Q o D loy n P o l , o , i i . v n in i l o i l i i n wo – w no n in lin – o l i NP- .

T n l ion, n, i w i i o i l o in n o i l loy n . W ow in S ion i i NP- o oxi n o i l loy n wi in ny on n o . In , i i o oxi n o i l loy n wi in ny o o n^δ o ny $\delta > 0$, w n i n o no in n wo .

Fin lly, in S ion 5, w x lo i l (ill NP-) in w i o l n oxi ly olv , n in S ion 6, w on l n i wo .

2 The Quorum Deployment Problem

In i ion, w o lly n . T o l o Q o D loy n P o l i o in , iv n o y n i i n wo , ow o o o y .

Mo o lly, w iv n i i n wo on i in o n no , onn y - in n wo . W iv n n n y n ix, C, i o o n in o no i o no j: $C_{i,j}$ i o l n y o n wo onn in i n j. In i , w o ni ion n wo i x . Any i n wo n , loy n l l , l in in o on ion.

W l o iv n o y , Q. Fo on n , w Q on i o x ly n o , on o no in n wo . W il o y wi o − n w − o y in in , w i ov o l i i v n wi i i ion. W o y i i n n y n ix, w ol n n no in o n ow n o . n y in ix i i 0 o 1. Q o p on in no j i (n only i) $Q_{p,j} = 1$. (S Fi 3( ) o n x l o o y in ix o .)

ll o i in l no ion o o y v y i o o in . ion lly in i , w l x i i ion, n llow ix Q o on in o o no no . I n o l x v ion o o l i olyno i lly iv l n o i v ion o o l .

A o loy n , n on i o wo o on n . Fi , ll ol n in o ix n no ; o ol n in o ix i n o no in n wo . T i in w i l no in o . I no i i i n o ol n j, n $Q_{p,j}$ in w no i i in o p. (ll ow o Q n o .)

S on , no i i n o o . Ty i lly w n in oy , no o in n o ion n o v y no in o o , o iv o v y no in o o . I , o x l , no i i i n o p, n w n v n o ion o no i, i o on o p. I i il (o il o no in o p, o x l), n no i y on o o . (I i − n − o l o in n o o o on.) In i , w o o i i o o on , w o p no il . Fo no i, o o loy n i in y o o in no ini i i n o . Fo x l , i no i i i n o p, n o o o loy n o i i :

$$\sum_{j \in p} C_{i,j}$$

W x o wo o on n o o loy n -ion on $[1, n]$. W o o on n , i n n o no o ol n in o ix, ion β. T i , no i i i n o ol n j i $\beta(i) = j$. T o , i no i i i n o p, n o o o loy n o i i :

$$\sum_{j=1}^{n} C_{i,j} \cdot Q_{p,\beta(j)}$$

T in o o in no j, n on in w no j i in o p: $Q_{p,\beta(j)}$ i 1 i ol n i n o j i o o p.

W o on o on n o o loy n , i n n o o o no , ion α. No i i i n o p i $\alpha(i) = p$. T o , o o o loy n o no i i :

$$\sum_{j=1}^{n} C_{i,j} \cdot Q_{\alpha(i),\beta(j)}$$

T o l o o o loy n i o l o o loy n o ll no in n wo . T o , o l o o loy n , $D(C, Q, \alpha, \beta)$ i :

$$D(C, Q, \alpha, \beta) = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{i,j} \cdot Q_{\alpha(i),\beta(j)}$$

o l i o ini i i o : iv n i C n Q, n wo -ion α n β on $\{1, \ldots, n\}$ ini i $D(C, Q, \alpha, \beta)$ o ll o i l oi o α n β. W ll i o i i ion o l Q o D loy n P o l .

T o o , w o ion lly on i v i n n i v -ion o Q o D loy n P o l . W i in o il y i . T ollowin i i vi w o v i n :

– : In i v i n , “ o ” no -i o in[1]. W y i o o i in l o l loy n o l .

[1] In this case, referring to the sets as “quorums” is a misuse of terminology, since the defining features of a set of quorums is that they intersect. For simplicity, however, we continue to use this term.

- : In i v i n , on o wo ion , α o β, i iv n in v n o o l in n .
- : In i v i n , o ni ion n wo i i o lin n wo . T i , ll no in i i n wo on lin .
- : In i v i n , o ix n i . In i l , i n w n no i i n l in li y.

3 Partial Quorum Deployment

W on i i o l o . In n l Q o D loy n P o l , w iv n o , Q, n i- i n wo , C, n o o l i o in loy n , $\langle\alpha, \beta\rangle$, o i l o . In o l , w on o wo ion in loy n i x . T i , w i α o β i iv n.

In on , ion α y x in v n . Fo x l , α y x i n i y: no 1 o 1, no o , . T o l i o in ion β, i n n o no o ol n o o ix.

In on , ion β i x in v n . T o l, n, i o in ion α, i n n o w i o no o l .

Bo o P i l D loy n P o l n o , w i n w ll i n n olv in olyno i l i (, o x l , [3]).

In , w iv n w i i i , on i- in o n no $-$ n l no , L, n n i no , R $-$ n w i n ion $w_{i,j} \geq 0$ o ll $i \in L$ n $j \in R$. T o l i o oo in on i in o n wi ini w i .

Theorem 1. C Q α ($O(n^2)$) β β α

A ion α i iv n. W on i i o A i n n P o l w l no , L, n no n i no , R, n ol n in o ix, Q. T w i o n onn in $i \in L$ n $j \in R$ i o o i nin i o ol n j in Q. T i :

$$w_{i,j} = \sum_{\ell=1}^{n} C_{\ell,i} \cdot Q_{\alpha(\ell),j} \ .$$

T A i n n P o l l in ion ini i o o w i . T l in ion ini i o o o - loy n .

iv l n ly, i ion β i iv n, l no in i i n no n i no n o ; w i o n n o o no in iv n o . In i :

$$w_{i,j} = \sum_{\ell=1}^{n} C_{i,\ell} \cdot Q_{j,\beta(\ell)} \ .$$

A in, A i n n P o l ini i w i , l in in - ion ini i o o o loy n . □

4 Hardness of the Quorum Deployment Problem

W il P i l D loy n P o l i ily olv l , n l Q o D loy n P o l i i . In i ion, w ow in S ion i i NP- o oxi n l Q o D loy n P o l wi in . , on n o . In , o ny $\delta > 0$, i i o oxi wi in o o n^{δ}, w n i n o no in n wo . W n ow no v i n , M i o D loy n o l , i NP- , n l x v ion (w o no i o in) i l o NP- o oxi .

Hardness of Approximation

in n l i iv o - in ion o B ln o l Bi i S (B BS) P o l ([ ] o n o o l , n [5] o n l). In i o l , w iv n ii , $G = (V, E)$, on i in o l no , L, n i no , R. W l o iv n on n , k. T o l i o n l n o l i i o i k, wi k l no n k i no .

T o o i ion, w i i in Fi n x l . No i i l n , o l o i wo, on i in o no n 3 on l (in L) n no 5 n on i (in R). How v , i no o i .

In o ion, w o n in n o Q o D loy n P o l n i n loy n i n only i G on in l n o l i i o i k.

Fi , w n ion, . . . $(G, k) = C$ n . . . . . $(G, k) = Q$, n o n in n o B BS o l in o n in n o Q o D loy n P o l . W oo $n = |V| + 1$. T $n - 1$ ol n n o o i in l B BS o l ; l ol n n ll o in .

T o ix, C, i l o “ o l n ” o in i n ix o , G: in ix G l in lin in ix C, w il wo i onn no in G onn y n x n iv lin in

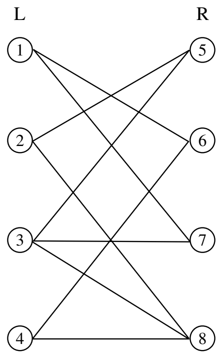


Fig. 2. Example instance of the Balanced Complete Bipartite Subgraph problem, where $k = 2$

$$\left(\begin{array}{cccc|cccc|c}
n^x & n^x & n^x & n^x & n^x & 1 & 1 & n^x & 1 \\
n^x & n^x & n^x & n^x & 1 & n^x & n^x & 1 & 1 \\
n^x & n^x & n^x & n^x & 1 & n^x & 1 & 1 & 1 \\
n^x & n^x & n^x & n^x & n^x & 1 & n^x & 1 & 1 \\
\hline
n^x & 1 & 1 & n^x & n^x & n^x & n^x & n^x & 1 \\
1 & n^x & n^x & 1 & n^x & n^x & n^x & n^x & 1 \\
1 & n^x & 1 & n^x & n^x & n^x & n^x & n^x & 1 \\
n^x & 1 & 1 & 1 & n^x & n^x & n^x & n^x & 1 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right)$$

(a) Cost Matrix, $Cost(G, k)$

$$\left(\begin{array}{cc|cccccc|c}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right)$$

(b) Quorum Matrix, $Quorums(G, k)$

Fig. 3. An example of a reduction from the Balanced Complete Bipartite Subgraph problem in Figure 2 to the Quorum Deployment Problem

ix C. Fo o o ion, w x x o n^x i i n ly l . T i o x n on i v l o δ. (T i , $x = O(\delta)$.) Fo lly:

$$(G,k)_{i,j} = \begin{cases} 1 & \text{i } (i,j) \in E \text{ n } i,j < n \\ n^x & \text{i } (i,j) \notin E \text{ n } i,j < n \\ 1 & \text{i } i = n \text{ o } j = n \end{cases}$$

on i x l in Fi 3( ). T ix li i y o ow n o ol n n w n no in L. No i no w n no in L, ll n i n^x. T ix li i y ow v o i n ol n v o i n w n no in R, n o on i only o n i n^x. T l ow n l ol n on in v l 1. T inin n i n w n no in L n no in R. Fo x l , n y (3,5) n w n no 3 n no 5. v o ix i y i .

No i xi n o i - o lin i i o n in ion. I io o xi o ni ion o o ini o i o n y on n , n ny loy n oxi o i l o wi in on n o . T o , ny in oxi ili y l llow io o ow n ow .

T o ix, Q, on i o k o on inin k no , n x no , n. I l o on in in l o on in ll no . T o o on in only no n. Fo lly:

$$(G,k)_{i,j} = \begin{cases} 1 \text{ i } i,j \le k \\ 1 \text{ i } i=n \\ 1 \text{ i } j=n \\ 0 \text{ o wi} \end{cases}$$

on i in x l in Fi 3( ). T wo ow n wo ol n on in v l 1, n in o l i i o i wo. T l ow n l ol n on in v l 1, w ll.

W ow i o i in l i i on in l n , o l i i o i k, n iv Q o D loy n P o l ll o . Al n iv ly, i o i in l i i o no on in , n iv Q o D loy n P o l l in i o loy n . A ll oo i on in in ll v ion [6].

Lemma 1. $x > 1$ $G = (V,E)$ $1 \le k \le |V|$ $C =$ (G,k) $Q =$ (Q,k)

$$(G,k) \in BCBS \;\Rightarrow\; \exists\alpha, \exists\beta,\; D(C,Q,\alpha,\beta) \le n^2$$
$$(G,k) \notin BCBS \;\Rightarrow\; \forall\alpha, \forall\beta,\; D(C,Q,\alpha,\beta) > n^x$$

k G n^2 no k G n^x

( ) T oo on i o wo . In , w $(G,k) \in BCBS$. In on , w $(G,k) \notin BCBS$.

$(G,k) \in BCBS$ Fi , o i l n o l i i on k no in G. W in loy n , (α,β) ll o . L $L' \subseteq L$ l i ion o n $R' \subseteq R$ i i ion o . oo α o no in L' o k ow , n oo β o no in R' o k ol n . No n i o ow n n ol n n. T n o o n i in k ow n k ol n i o on o in o l i i

, n l, o 1. o o n i in ow n n ol n n i o n n y in o ix o o 1. T o, o l o o loy n i $k^2 + n - 1 \le n^2$, i .

. , $(G,k) \notin BCBS$, n o n, o i no o l i i on k no in G. W ll ny loy n o l n n^x. In i l, v y loy n in l l on x n iv . I i l no n n, wi o lo o n li y, o ow n n ol n n: iv n no i l i n n w i i no , i i o i l o i n n o i i , wi o in in o . T n no i i i loy n o no in l ny n y o n^x, n i i li xi o l i i o i k, w i w w no . □

W on l Q o D loy n P o l i o oxi :

Theorem 2. $\delta > 0$ n^δ

Hardness of Metric Cost Quorum Deployment

In M i o Q o D loy n P o l , o ix i i o y i n i y i n l in li y. In i , o o i n in o j i o o j n in o i, n o o n in o i o j i no l n o o n in o i o k n o k o j. I i l o ion in L 1 i v ion o o l i NP- :

Theorem 3.

W ion in L 1, x in o on in ix (G,k) y in non- o o n^x, w non- o o . T ix i i ly i i n o i . T o n ollow y n in L 1, w i $(G,k) \in BCBS$ n o i l o o loy n i $k^2 + n - 1$; o wi , i $(G,k) \notin BCBS$, n o o ny loy n i l $k^2 + n$. □

Hardness of Relaxed Metric Quorum Deployment

I w o no i “ o ” in , n w n ow l x loy n o l i in oxi l v n w n o ix i y i n i i n l in li y. T oo i in i y - ion o on ly NP- o l 3-P i ion P o l ([ ], SP15) o Q i A i n n P o l (QAP) ([ ], ND 3) iv n y Q y nn [ ]. ion x n l o Q y nn . Sin loy n l o i llow wo o o , α n β, o o QAP only on o o ($\alpha = \beta$ in QAP), w on n in n o loy n o l i fl xi ili y, n in w n i no 3- i ion o o loy n i i . T oo i n in ll v ion [6].

Theorem 4. [illegible]

W no oo o i o i li w n o ix i lo i on l ix (on in i lo n o v yw l) n n o lo n l olyno i l ion o n, n loy n o l i in oxi l o wi in ny on n o . W l o no i o ix on in j on lo , n i i NP- o o i lly olv o l . T i ollow o oo o T o 3, w o o ow n i ol n i o i .

5 Approximation Algorithms for Metric Costs and Restricted Quorums

W v n i w llow i y l x o ix, n i no on n o oxi ion l o i o loy n o l v n i w o ix i y i n i i n l in li y. I in i y o o ix l y n i o n ol in ili y o oxi o l . T o , in i ion, w li ily o o w on iv o i i on n o oxi ion o i o n wo . Solvin loy n o i lly o i ily, ow v , i ill NP- .

W iv on n o oxi ion l o i o Q o D loy- n P o l wi [illegible] n y i o ix i i n l in li y. T o ix i o- o o on n n p o [illegible] l on i on l. y- ol i i on in in i on n n k_i o n (Fi , n o l ni ion in ll v ion [6]). T oxi ion o i $c =$ · x$_{1\leq r\leq p}\, k_r$. T l o i n in $O(n^{k_1+\ldots+k_p+3p})$ i .

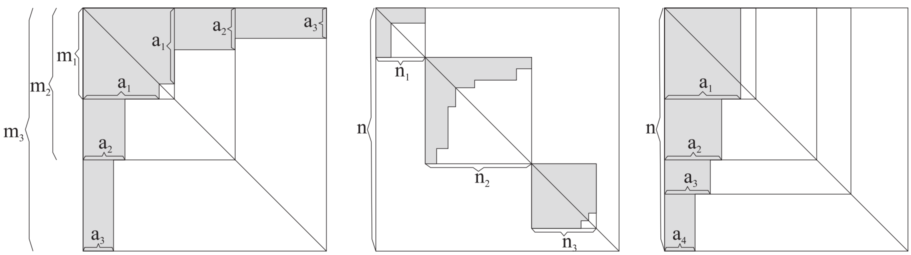


Fig. 4. Left: example of a hyperbola contained in $k = 3$ nested squares. Middle: example of a block diagonal hyperbolic quorum matrix with $p = 3$ hyperbolas with $k_1 = 1$, $k_2 = 3$ and $k_3 = 2$ nested squares respectively. Right: a quorum matrix composed of a part, called vertical telescope, of a single hyperbola. Note that any two quorums intersect in this matrix

Theorem 5. c $c =$ · x$_{1\leq r\leq p}\, k_r$ $O(n^{k_1+\ldots+k_p+3p})$

T oo ollow n n ov vi w o oxi ion l-o i n o y o v ion . A il oo o o i iv n in ll v ion [6]. Fo onv ni n o n ion, w i y -ion α n β o n ow n ol n o ix n ix. T i o o yi l n iv l n o i i ion o l .

( ) S o o o n o ix on in i ix $U \times U$, n o v yw l . L $m = |U|$. An o i l loy-n will l o ow $\tilde{U'}$ n o ol n $\tilde{V'}$ o o ix in i $U \times U$. W n w i ow i n m ol n V, ini i o o in ion o ow n ol n , n y i n l in li y n y i i y o o ix, w n on l o o in i ix $V \times V$ i o wi o o o i l loy n . W no i on l ion i v n o o i l loy n y v $\tilde{U'} \neq \tilde{V'}$, i. ., y in v n o wo o o o low o . T i o v ion x n ni o K l. [ ] v lo o Q i A i n n P o l w w wo l v $\tilde{U'} = \tilde{V'}$ (in QAP ow n ol n in w y).

Now o o ix i o in l y ol . T n n o i l loy n x ili y o voi i o o “ ol ” in o ix, o o $U \times U$ j i . W n ow, ow v , ow o ff iv ly l wi ol y o i ly n in ow n ol n o “ ” low o o o i y y ol , n l v i o in . T y ol i on-in in k n . T h i m_h y m_h n y ol i n a_h o (. Fi). Fo h, w n n ow i_h n m_h ol n V_h ini i o o in -ion o i ow n ol n . Sin w v l ow n ol n ini i , l ly, o o ny o i l loy n i l $1/k \sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j}$. T i i li i o n l v oo i o in oi o V_h, n o i $V_h \times V_h$ wo l no l o oxi ion i wo l no v o n . -ll k n in o i l loy n , o w n ill o n o low o o loy n i w in o on in $V_1 \subset V_2 \subset \ldots \subset V_k$. Wi i on in o , n n i -w n V_h’ . H n w nno o ini i ion o $\sum_{j\in V_h} C_{i_h,j}$ o oi o i_h n V_h in n n o ini i ion o o-on in o o ow n o o ol n w o l in lo l ini . W w n o o in , i o in-i i v l o n i o n o ll o i l oi n on-in . W n n n V_h n ow i_h ini i o n

$\sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j}$ in n o i ly j olyno i l i l oi o To y n N no [9], in ion lin o y nn-B n H in [30]. A V_h' n i_h' v n o n , w - n ow n ol n . U in i n l in li y n y i i y o o ix, w on l o in i ix $V_h \times V_h$ n o n o ov y m_h i o in ion o ow i_h n ol n V_h. I w ov a_h low o ow o o o ix, n o o l i o o ion lly , n o i i o a_h/m_h ion o o o in i n i ix. W n ow o ix $V_1 \times V_1$, n ow o $V_2 \times V_2$ n o on, n n ol n . W n n n on lly, w n n on n n o no oy o n on o y io n n . A n n , o o in i ol will o $\sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j}$. T i o l oxi ion n o in l ol .

Fin lly, o ix i lo i on l y oli oix o o o p y ol . W o i y l o i o n in n o ol n , o now l o i ini i o p oll - ion o n o ol n . A w v o n oll ion , w ly, o o p oll ion o n i , l o i o n in ow n ol n . T i yi l i oxi ion l n o l oo . □

W on o oxi ion l wi in oxi ili y l o vio ion. W n n o y ol n i olyno i l ion o n, n loy n o l i in oxi l o wi in ny on n o , v n w n y ol i i j in l o l ly ll in wi on . How v , w n oxi o l o wi in on n o w n n o y ol i on n , n v n i y ol i on in in o n on .

6 Conclusions and Future Work

In i , w v in o Q o D loy n o l , n l o l i w n in o i n ly li . W v xin o l xi y o n o v i n o o l , owin P i l D loy n P o l n olv in olyno i l i , w il n l Q o D loy n P o l n l x M i D loy n o l in oxi l . Fin lly, w n o i l NP- in w i o l n oxi n o i o i l olyno i l i ol ion.

W il ny o l n in i n iv , w li v i i i o n o on in x inin o w i o y i n ly loy , o l i ni n i l i o . Mo vio o on v lo in o y v oo o n o v - io il o ; o l l o in o o n i l y o loyin o . W il w onj o n ly v lo o y (i o y) nno loy i n ly, w wo l li o v lo ili o o y o o n n loy i n ly.

References

1. Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the seventh symposium on operating systems principles. (1979) 150–162
2. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. Transactions on Database Systems **4** (1979) 180–209
3. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. Journal of the ACM **32** (1985) 841–860
4. Herlihy, M.: A quorum-consensus replication method for abstract data types. ACM Transactions on Computer Systems **4** (1986) 32–53
5. Agrawal, D., Abbadi, A.E.: Resilient logical structures for efficient management of replicated data. Technical report, University of California Santa Barbara (1992)
6. Bearden, M., Jr., R.P.B.: A fault-tolerant algorithm for decentralized on-line quorum adaptation. In: Proceedings of the 28th International Symposium on Fault-Tolerant Computing Systems, Munich, Germany (1998)
7. El Abbadi, A., Toueg, S.: Maintaining availability in partitioned replicated databases. Transactions on Database Systems **14** (1989) 264–290
8. El Abbadi, A., Skeen, D., Cristian, F.: An efficient fault-tolerant protocol for replicated data management. In: Proc. of the 4th Symp. on Principles of Databases, ACM Press (1985) 215–228
9. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. Journal of the ACM **42** (1995) 124–142
10. Lynch, N., Shvartsman., A.: RAMBO: A reconfigurable atomic memory service for dynamic networks. In: Proc. of the 16th Intl. Symp. on Distributed Computing. (2002) 173–190
11. Gilbert, S., Lynch, N., Shvartsman, A.: RAMBO II:: Rapidly reconfigurable atomic memory for dynamic networks. In: Proc. of the Intl. Conference on Dependable Systems and Networks. (2003) 259–269
12. Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. ACM Tranactions on Computer Systems **3** (1985) 145–159
13. Naor, M., Wieder, U.: Access control and signatures via quorum secret sharing. IEEE Transactions on Parallel and Distributed Systems **9** (1998) 909–922
14. Upfal, E., Wigderson, A.: How to share memory in a distributed system. Journal of the ACM **34** (1987) 116–127
15. Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: Proceedings 27th Annual IEEE Symposium on Foundations of Computer Science, New York, IEEE (1986) 233–243
16. Cheung, S.Y., Ammar, M.H., Ahamad, M.: The grid protocol: A high performance scheme for maintaining replicated data. Knowledge and Data Engineering **4** (1992) 582–592
17. Peleg, D., Wool, A.: Crumbling walls: a class of high availability quorum systems. In: Proceedings of the 14th ACM Symposium on Principles of Distributed Computing. (1995) 120–129

18. Malkhi, D., Reiter, M.: Byzantine quorum systems. In: Proceedings of the 29th Symposium on Theory of Computing. (1997) 569–578
19. Naor, M., Wool, A.: The load, capacity, and availability of quorums systems. SIAM Journal on Computing **27** (1998) 423–447
20. Naor, M., Wieder, U.: Scalable and dynamic quorum systems. In: Twenty-Second ACM Symposium on Principles of Distributed Computing. (2003)
21. Tsuchiya, T., Yamaguchi, M., Kikun, T.: Minimizing the maximum delay for reaching consensus in quorum-based mutual exclusion schemes. IEEE Transactions on Parallel and Distributed Systems **10** (1999) 337–345
22. Fu, A.W.: Delay-optimal quorum consensus for distributed systems. IEEE Transactions on Parallel and Distributed Systems **8** (1997) 59–69
23. Schrijver, A.: 17. In: Combinatorial Optimization. Volume A. Springer (2003)
24. Gary, M.R., Johnson, D.S.: Computers and Intractability. Freeman (1979)
25. Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Applied Mathematics **131** (2003) 651–654
26. Gilbert, S., Malewicz, G.: The quorum deployment problem. Technical Report CSAIL-TR-972, MIT (2004)
27. Queyranne, M.: Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. Operations Research Letters **4** (1986) 231–234
28. Krumke, S.O., Marathe, M.V., Noltemeier, H., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J.: Compact location problems. Theoretical Computer Science **181** (1997) 379–404
29. Tokuyama, T., Nakano, J.: Geometric algorithms for the minimum cost assignment problem. Random Structures and Algorithms **6** (1995) 393–406
30. Guttmann-Beck, N., Hassin, R.: Approximation algorithms for min-sum p-clustering. Discrete Applied Mathematics **89** (1998) 125–142

A Constraint-Based Formalism for Consistency in Replicated Systems

M S i o[1], K i y n B v n[1], n Ni i K i n [2]

[1] Microsoft Research, Cambridge, United Kingdom
[2] Compter Science Department, Courant Institute,
New York University, USA

Abstract. We present a formalism for modeling replication in a distributed system with concurrent users sharing information. It is based on actions, which represent operations requested by independent users, and constraints, representing scheduling relations between actions. The formalism encompasses semantics of shared data, such as commutativity or conflict between actions, and user intents such as causal dependence or atomicity. It enables us to reason about the consistency properties of a replication protocol or of classes of protocols. It supports weak consistency (optimistic protocols) as well as the stronger pessimistic protocols. Our approach clarifies the requirements and assumptions common to all replication systems. We are able to prove a number of common properties. For instance consistency properties that appear different operationally are proved equivalent under suitable liveness assumptions. The formalism enables us to design a new, generalised peer-to-peer consistency protocol.

1 Introduction

li in in i i y i ov v il ili y o o
in inin on i n y, in i ’ vi w y i l o l . I i w ll
li ion n o i n y in n i in o
o n , i i i l o on o o n o w o-
o ol . P i l li ion on i n o li ion. D i l
o y o vio wo [1], w l o l wo o n n in , -
onin o , n o in li ion o o ol . T i n
wo .

W o l i i y li n li ion
o o ol. U in n ly i i n o , -
n , li ion n y (o
li n in i wo) l o i lin o x i -
o n in n o n i o i .

i lo l vi w (ll l ilo) o nown ion n on in .
T i x l , w i o l ly in o
li . Si onv i y x ion in o , w i
li ion o o ol n , i n y, y in o on in .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 331–345, 2005.

on i ion ollowin . W o o nov l wo o onin o li y . I i ni : n i o ivi y n onfli , li ion n i l n n , in n o i i y, n o o ol i ion o w i o ion o x n in w i o . wo i i l , n iv n i o li ion in o on in .

l l i y i n n ion o li ion y . A n x l , w will o l iff n y in o wo , . ., B yo n SDS, n ov on i n .

W l o ov in in o i o l o li ion o o ol . Fo in n , w i n i y o iff n no ion o on i n y, w i o iff in i o ion l i n . W l o ov , n i n ly on liv n ion , y iv l n .

T wo n i i n o n w li ion o o ol . W oo n w i i li ion l o i n li in B yo , w o i n i i ly i y wo . I ow on in n n i l n ion ni w ll i ion wo .

T o ollow . S ion ov vi w i o li . S ion 3 n n o on i n y o i . W x in o i ion l o i o li n l o lo l i ion in S ion . W iv nov l n li li ion l o i in S ion 5. S ion 6 o wi l wo , n w on l in S ion wi y o on i ion n wo .

A ni l o [ ] ovi o l o l n . H w o on n in in i ion n o li o ini .

2 Formal Framework

i in li y in in lo l vi w ll .[1] T n l o x in (i. ., v li) l o o l ilo . v i l ilo ow (n on lly n v in) y i ion o ion n on in , i y lo l li n o iv o o i . T o o n l ow wi n o ion n in n o on in in .

2.1 Actions and Schedules

Sli ly o o lly, A i o ni ion INIT, $\alpha, \beta, \ldots$. A ion ini i [2] o wi nin . T $\overline{\alpha}$ i l ol wi no ff (non- ion will l w n i in liv n). A ion INIT n ini i l n no ff .

[1] We call it a multilog and not a log, because it contains actions submitted at several sites and the actions are not ordered.

[2] Executing the same action from two equivalent input states yields equivalent output states.

A i non- y n o ion n non- ion , o in n $S = \textsc{init}.\alpha.\overline{\beta}.\gamma$. In i x l , α i (no $\alpha \in S$), n β i non- x (no $\overline{\beta} \in S$); ll o ion i (no (α, S)). A iv n ion y only on in l , i x o non- x . T o in i no $<_S$. v y l wi INIT. In i iv ly, non- ion in l in i l i w o ion o no x i , . ., o on in .

A ion o nl i o wi y no ion $\alpha \not\lhd \beta$ ("non- o in "). A non- ion o wi v y ion n non- ion. Two l $(S_1 \equiv S_2)$ i y x ion , n non- o in i o ion x in o .

o ivi y llow o o l n o l-wo l o l iv l n :

- l i lly, ion o i o , o i y in n n v i l .
- v w i in : in o y n o -o -o w i no ff ; n w i ff iv ly o . Fo in n in i li ion (L W i Win) [1], w i in l w w i i i n l ' ; i o w i ff , o wi i i no-o [ ].
- on ili ion: An x l o on ili ion l o i i ion l T n - o ion [3]. Two ion i on n ly x in i y o . T on on o x i n o o i no ff o , in ff n in o iv .
- F il o o : An ion il o o o , i. ., non- ion in ll l , w i o wi ll ion .

2.2 Multilogs and Sound Schedules

M l ilo $M = (K, \rightarrow, \rhd)$ n i ' vi w. K i o nown ion ($K \subseteq A$); $\rightarrow$ n $\rhd$ o nown on in . T l ion $\rightarrow \subseteq A \times A$ (ono n B o) i no n ily y li , no fl xiv , no n i iv . l ion $\rhd \subseteq A \times A$ (ono n M H v) i n i iv n fl xiv . By onv n ion, o ny $\alpha \in A$, INIT $\rightarrow \alpha$ n $\alpha \rhd$ INIT; i i l i li i in o .

Fi l iv o x l o on in n o o on o in ion . In i iv ly, $\alpha \rightarrow \beta$ in i l in in n o in w n wo ion : no l y x β o α. A l x n i α no β, o only α, o only β, o o α n β in o (no n ily j n) i o wi o i on in . l ion $\alpha \rhd \beta$ i n i li ion: i α x in l , n β l o x o - w in l , l o no n ily in o . A l x only β, o x n i α no β, i o wi o i on in . onv ly, i l non- x β, n α y no x .

T o o n l o M i no $\Sigma(M)$; M i i o n i $\Sigma(M) \neq \emptyset$. S l $S \in \Sigma(M)$ iff:

Fig. 1. Example constraints. α, β and γ form a *parcel*, an atomic (i.e., all-or-nothing) execution. γ executes only if δ also executes. δ is *causally dependent* on ϵ. ϵ and ζ *conflict* with (i.e., mutually exclude) each other. Only two actions out of the three γ, θ and κ can execute. If both χ and κ execute, χ comes first

l n i y n o o n l : $M_1 \equiv M_2$ iff $\Sigma(M_1) = \Sigma(M_2)$. No $\Sigma(M)$ i lo wi o l iv l n . H - , w i n i y l ilo wi i iv l n l .

T i li i on in l n i i in ly x iv . W v i o x n i o li- ion iv l n- , v l v ion y n li l y [,5]. Fo in- n i α i o y n β l in i o y, l y i $\beta \rhd \alpha \wedge \alpha \rightarrow \beta$ (l n n) lon wi β.

A o ion c i i o i ion in c o → y l . In i iv ly, i n no o n l n x ll ion in c. Fo x l , i $\alpha \rightarrow \beta$ n $\beta \rightarrow \alpha$, n α n β onfli , i. ., n no o n l x o o .

2.3 Significant Subsets and Events of a Replication Protocol

x ion i v y wi ly w n li ion o o ol : in o , ion x i i ly, in o y ; x ion o y - li o o ; ion i oll . How v o o ol wo l l i i i no o n l i ion o v y ion. W n i ion on in ; ollowin o i l o i vo l i ion:

- **Guaranteed** ion x in v y l . (M) i ll i yin : (1) INIT $\in$ (M). ( ) $\forall\beta \in A$: I $\alpha \in$ (M) n $\alpha \rhd \beta$ n $\beta \in$ (M).
- **Dead** ion non- x in v y l . (M) i ll i yin : (1) $\forall\alpha \in A$: I $\beta_1, \ldots, \beta_m \in$ (M), w m i ny n l in , n $\alpha \rightarrow \beta_1 \rightarrow \ldots \rightarrow \beta_m \rightarrow \alpha$, n $\alpha \in$ (M). ( ) $\forall\alpha \in A$: I $\beta \in$ (M) n $\alpha \rhd \beta$, n $\alpha \in$ (M).
- A **serialised** ion i on i o wi o ll non- o in ion x $(M) \overset{\text{def}}{=} \{\alpha \in A | \forall\beta \in A, \alpha \not\bowtie \beta \Rightarrow \alpha \rightarrow \beta \vee \beta \rightarrow \alpha \vee \beta \in$ $(M)\}$
- An ion i **decided** on i i i , o o n n i- li .
  $(M) \overset{\text{def}}{=}$ $(M) \cup ($ $(M) \cap$ $(M))$
- An ion i **stable** w n i ff nno n , i. ., i i i , o i i n n i li n ll in ion lv l . (In i , l ion n n o l ilo .) (M) i ll i yin : (1) INIT $\in$ (M), ( ) $(M) \subseteq$ (M), (3) I $(\alpha \in$ $(M) \cap$ $(M)) \wedge (\forall\beta \in A : \beta \rightarrow \alpha \Rightarrow \beta \in$ $(M))$ n $\alpha \in$ (M).

No i M i o n , v y n ion nown: $(M) \subseteq K$. Al o no $\alpha \rightarrow \alpha \Rightarrow \alpha \in$ (M) n INIT $\rhd \alpha \Rightarrow \alpha \in$ (M). M i o n iff n n i join .

3 Replication and Consistency

In i ion, w i liv n n y o i w i o li ion y , in o o ion- on in wo .

3.1 Site Schedules and Transition Rules

Diff n li ion y (SDS n B yo) iff y ion n on in y , n y i ion y . W i li ion o o ol y l i in ow y n o i t o $t+1$.

T n o i i i l o nnin $S_i(t) \in \Sigma(M_i(t))$. In o wo , i $|\Sigma(M_i(t))| > 1$, n oi w n o n l i i l v n o on i n y, l o in ivi l li ion y y lly i l o o i li y.

i i i own vi w $M_i(t) = (K_i, \rightarrow_i, \rhd_i)(t)$, volvin ov i t, ll i .[3] M l ilo ono oni lly non- in in , w i i li i ni n o S ion .3 non- in in , n n n o n l ilo in n o n o v .

[3] For simplicity we assume discrete time and use a global time notation. The theory does not assume that a site can observe the global time.

All o o ol o y Univ l T n i ion l , w i y i ly i y iv ion n on in o lo l li n o o o l ilo . A i o o ol y v i ion l n i ion l . A n x l , l n o lin i l o o ol, i. ., on in w i n ion ff o in n in i , n ion x in in - ff o . W n l i o ollowin n i ion l : “ nly on ion y i ni o i ; i α i i i t, n o ny ion $\beta \neq \alpha$: i $\beta \in \bigcup_j K_j(t-1)$ n $\beta \rightarrow \alpha$, o wi $\alpha \rightarrow \beta$.”

A li y on i i i on n y on ol, o , n i ion l n v y i n v y i $S_i(t)$ i x o $S_i(t+1)$. wi y i i .

3.2 Liveness Conditions

W il iff n li ion l o i in in iff n on i n y inv i n , ll o i y o liv n on i ion o onv n . W i n i y wo liv n on i ion , on o o ion o o ol i i ion n on in , o o i ion l o i ili ion n l ilo .

T o ion o o ol n ll ion n on in - i o y v n lly ll no .

Property 1 (Eventual Propagation). ( ),

- $\alpha \in K_i(t) \Rightarrow \forall j : \exists t' : \alpha \in K_j(t')$
- $\alpha \rhd_{i,(t)} \beta \Rightarrow \forall j : \exists t' : \alpha \rhd_{j,(t')} \beta$
- $\alpha \rightarrow_{i,(t)} \beta \Rightarrow \forall j : \exists t' : \alpha \rightarrow_{j,(t')} \beta$

T i ion l o i n ll lo lly nown ion v n- lly i :

Property 2 (Eventual Decision). ( ), $\alpha \in K_i(t) \Rightarrow \exists t' : \alpha \in$ $(M_i(t'))$

D i li v y ion v n lly o l [ ]. D o no l ivi l i l n ion v y ion ; o wo o no l i o , in i i v li y i ion il.

3.3 Mergeability and Uniform Local Soundness

W now i iff n ni ion o on i n y in o wo . T on , M ili y, in i ion i no onfli in i ion : y o i l o ni i n o v wo l no ny in w on . M ili y n li l i l o y.

Property 3. $i, i', i'' \ldots$ $t, t', t'' \ldots$: $M_i(t) \cup M_{i'}(t') \cup M_{i''}(t'') \ldots$

M ili y i no y o n in i i in . Fo in n , on i Si 1 l ilo $(\{\alpha\}, \emptyset, \{\textsc{init} \rhd \alpha\})$ n Si l ilo $(\{\alpha\}, \{\alpha \rightarrow \alpha\}, \emptyset)$. T y o o n no l , i nion $(\{\alpha\}, \{\alpha \rightarrow \alpha\}, \{\textsc{init} \rhd \alpha\})$ i no o n .

M ili y o y, i i no i ll i on ini i i ion y. Fo in n , i l i - o o ol n n ili y y n in ll i o ion ni o ly, in lo l i .

Un P liv n ion, v y i ion n on in i v n lly iv v yw , o in ff v y i o n o ni i n o v . T n M ili y o i l Uni o Lo l So n n (ULS) inv i n i l ilo o n ll i : $\forall i, t : \Sigma(M_i(t)) \neq \emptyset$.

3.4 Eventual Consistency

A l i l on i n y o y o o i i i li ion y i v n l on i n y. I n o in o lly o o n o vin [6] o B yo [ ].

Property 4. [illegible]

$$\begin{aligned}
&\exists T : \forall i, t > T \Rightarrow \quad [illegible] \quad i\\
&\implies\\
&\exists T', \forall t', t'', i, j : t' > T' \wedge t'' > T'\\
&\wedge S_i(t') \in \Sigma(M_i(t')) \wedge S_j(t'') \in \Sigma(M_j(t''))\\
&\qquad \Rightarrow S_i(t') \equiv S_j(t'')
\end{aligned}$$

Al o v n l on i n y i ly no ion o li onv n , i y li l o y inv i n i y l o i o y ili ; o i y ili y.

3.5 Common Monotonic Strong Prefix (CMSP)

L o ' li in o [ ] n ll i x x ly l . l ly y i on i n , i o no wo o o i i i o o ol w $S_i(t)$ i no n ily x o $S_i(t+1)$. How v , v n in n o i i i y , ov i o ion will ili n o x o ll l . S y i on i n i l x o iff n i iv l n . T y o i x ow .

Fo lly, l P i x o l S, w i n $P \ll S$, i $S \equiv S'$ w S' i l o o $P.Q$ o o n o ion Q.

Property 5. [illegible] $M_i(t)$ *(i* [illegible] *t* [illegible] *)* [illegible] *(* [illegible] *)* [illegible] $\pi(i, t)$ [illegible]

$\pi(i,t)$ ll $\forall S \in \Sigma(M_i(t)) \Rightarrow \pi(i,t) \ll S$
$\pi(i,t) \equiv \pi(j,t)$
$t < t' \implies \pi(i,t) \ll \pi(i,t')$
$\forall \alpha \in K_i(t) \implies \exists t'$:
$(\alpha, \pi(i,t'))$

W ow ion in MSP l , n o l ion o MSP [ ].

3.6 Summary

W v n o ni ion o on i n y, lon wi wo liv n on i- ion . An in in l i n ni o ion , ni ion o on i n y iv l n . T i y o i , in o - ion l ni ion o iff n . In i l , n v n l o - ion n v n l i ion liv n on i ion , ni o lo l o n n (n n ili y) n v n l on i n y n o on ono oni on x o y. W ovi o l oo in o ni l o [ ].

4 Replication Systems and Decision Strategies

on i n y i n n w n ll i , w i in n l n il on n . Fo in n , ili y o i iff n i o - in onfli in i ion , i in on n w n i in i . Y o i l o o ol n wi o i o l xi y, i ily y in ion o i i ion o on in o ion . H , w v y w li ion l o i n i i ion i .

In i li ion, ll ion o v ni i . T i i in o l → o on ll ion v n o y i o y . No i ion n o in ll ion n n o , n l , w n i . H n ili y i n y l . T li ion l o i o i l o ion o o ol i y P o y.

A v i ion on i li ion l o i i on "l -w i win " i ion y. I ion o i in l v i l n w n wo ion o i y v i l , ion wi l i o l ff iv ly x l . So, w n wo ion iv o o o , i y o , o on iv l i onv o no- o (i n i y ion). A w in S ion .1, i y ll ion ff iv ly o , n ivi lly n ili y, w il llowin i o x ion wi o l y.

T SDS o o ol [9] ion only v y li l on in w n , o o $\alpha \rhd \beta, \beta \rightarrow \alpha$. All ion n i ion, only i n i i l o x . SDS in o o ion l o i i y P o y, w il in inin inv i n w n v α i o o i , ll β $\alpha \rhd \beta$ v l y n o . T i n l i o ily o ion i n ly x . Sin ion o no o , SDS i i i n o i li ion. All i i i in o in o l o o ion i on i n wi l o .

M ny y n li on n i y i . B yo [ ] n i ion in o in n n , wi i own i y i . A ion on iff n i ion o n o v no on in w n o . P i i i ion o i own ion n o . H n , li ion y on i o o ion o o ol i yin P n ll ion i i i , i y i ion y n D, n o ion o o ol i i i y i ion o ll i . By n li in i ion- in o i ion, ili y o ion n on in on in l i ion n , n y i llowin on in w n i ion , ll i l ilo l .

I on in o w ll- v o i , o i ion n ly n li ; in S ion 5 w will iv n i n i ion o o ol o ollowin o v ion . on i o in n n ion α i involv in in l on in $\alpha \rhd \beta$: n i i lw y o α , l o i ion o β. onv ly, i α i only involv in $\gamma \rhd \alpha$, i i lw y o α n , l o γ. T i n n li o ny y li $\rhd$. T in x l o in $\alpha_1 \rhd \ldots \rhd \alpha_n$ i i o i : α_1 , n ov on o α_2, l o i ; o α_n n , n ov on o α_{n-1}, i o l . Sin on' li o i ion o , n in in i -l i ion i l .

T i ion in α_i on i → on in . I α_i i no o → y l , i ion y i n o (l o n in i l). I i i o → y l , n ll o ion in y l n , only o n i ion i o α_i ; o wi i i ion i llow .

S lo l i ion y -o i l. To n o i li y, vi ., ll o i l n o ion i , i i n y o on i w ol in I [ ].

5 A Decentralised Replication Algorithm

on i v l oo in y , w i lin n o l n i own i i , w n i o l n fli oo in o n o- i lly (ll-o -no in). P vio y o no o i n io: o in- n B yo i o ll ion in n ion v i y.

W n n w l o i , iv o i ion on i ion o S - ion , wo o i y on in , n o no ff i i ion.

5.1 Input Assumptions

W inv i n w n α i in K_i, ll on in $\alpha \rhd \beta$ n $\beta \to \alpha$ (o ny β) nown *i.* ion in A i v n lly i o i . In i ion, ion α i i n ni i y i , $P(\alpha)$. W wo ion o i n only i y v iff n i i . onfli in (lly- x l in) ion n y → y l . W xi n o n ion (*c*) ini i lly oo on ion o o ion C.

No B yo li on in n n o i i o n l i i i ion in . i y w i o ion n n o wi o oo in in wi o i i . In on , o l o i on i ▷ n → on in w n ion on iff n i i .

5.2 Propagation Module

W - n B yo n i- n o y l o i o o in ion n on in o ll i . T l o i i v n l o ion o y: v y ion n on in i o i v n lly ll o i . In i ion, i in in inv i n o vio ion.

5.3 The Decision Algorithm

v y i y now o v y ion w i i n o , n i x ion o wi o o non- o in ion . To n i ion , i in in G_i o n ion , D_i o ion , n l ion $O_i \subseteq G_i \times G_i$ o lly o ll ion lon - in o i y. T no l o ion o l li ly i i i ion on ll i .

iv n , l x i i ny l on in ll n ion , no ion , n o y M H v on in in $\rhd_i$ n o in on in in $\to_i$ n O_i. Fo ni o lo l o n n , l lw y xi . Fo v n l i ion o ol , ll ion in K_i v n lly in l in G_i o D_i.

T i ion l o i n on n ly wi o ion o l . An ion i i o y , n i o y o i ion, i y o n l , n n lly i o n o . W n i ion l o i in o .

An ion α i i o i i y $P(\alpha)$ i

- All β $\alpha \rhd \beta$ nown $P(\alpha)$
- All β $\beta \to \ldots \to \alpha$ nown $P(\alpha)$.

o on i ion i o w i o ny i ion on α n n. A i y o y ion o w i i oo n x ion o i ion on.

n ll on in on n ion α oll , i y in o o i ov in w α n n . In i l , in i now lo o B o n M H v l ion , i n ll i ion y l w n ion . Fo n ion o n l , ll ion i M H v o l n l n l on o → y l i lon o o l . T i o i ollowin :

- o M o ll ion in ▷ y l wi α. L o inin ion i M H v i n M'.
- o C o ion n in y l o → involvin α.
- W i o ll ion in M' o o n . I ny o ion o , α i now nown o ; xi .
- Fo y l c in C, i n (c) o . I i ion i α, xi .
- D i n α n l .[4]
- S n o ll i i wi ion in M yin α i n- l .

W in ly on o ion o l o i i n l ion o l i i .

In o ▷ y l , ll o y l o i n o . T n l o n in ollow :

- W i n il i o ion in M i , o ll ion in M n- l . I o , α i now nown o ; xi .
- W i n il ll β $\beta \to \ldots \to \alpha$ n $P(\beta) = P(\alpha)$ v n i .
- n α n o i ll n β wi $P(\beta) = P(\alpha)$

In o o o in n l n n - ion , w i n i y wo on i ion in w i n ion o : i w n on o ion i M H v i (i own ▷ in, o in ▷ y l), o w n i i i n vi i in → y l .

T oi o ion o in → y l n i y. In n l i i o on o o ion in y l , lon i i o ny ▷ in. How v , in oo ny ion , o oo in w on ion o , n v n iv i on o n .

W now i o i in - ion α. A α w i i j.

[4] Some systems may elect to make α dead at this point according to their own strategies. For instance, Bayou checks a predicate, called the "dependency check," attached to each action.

1. T o i i (o o) o ni ion, α i v n lly nown i i y i i, $P(\alpha) = i$.
. T o ion o l i i o ni wi o i , i ov in ll β : $\alpha \rhd \ldots \rhd \beta \vee \beta \to \ldots \to \alpha$. T ion o y.
3. Fo y l c o → involvin α, i $(c) = \alpha$, n i α i (. ., on in $\alpha \to \alpha$) n xi .
. P i ion ll β $\alpha \rhd \beta$, in o M n M', o in o ollowin o y: ion in M $\beta \rhd \alpha$, o in M' no .
5. W i n il: i o ion in M' i nown o ; o ll ion in M' nown o n . In o , α i now nown o ; xi . In l , α i now n l .
6. To ll ion in M, n yin i α i n l .
. W i o i o ion in M o nown o , o o ll ion in M o n l . In o , α i now nown o ; xi . In l , i α i n (. ., INIT $\rhd \alpha$).
. T n l x ion o o α i iv n y i → l ion . W i o ll β $\beta \to \ldots \to \alpha \wedge P(\alpha) = P(\beta)$. x α ll ion n .

5.4 Correctness

To ov on i n y n onv n o l o i , w ly on v n l o ion, on v n l i ion, n on ni o lo l o n n .

T o ion o l i ion on n n i- n o y o o ol n li ly liv ll ion , on in , n i ion . To ov i ion l o i v n lly i v y ion, w ow ll w i on i ion in l o i v n lly i , i. ., now i - o y l . Fo ni o lo l o n n , w v y i ion x n o on in in o n nn y o in - y- n ly i on l o i .

5.5 Extensions for Partial Replication

U o now w ll i li v y i . L now on i i l li ion: i i ion in o n i join $D^1, \ldots, D^n$, n w llow i o li n i y o (lon v y i n on l on i). A ion o- on in ly i ion in o $A^1, \ldots, A^n$. A i li in D^i o l iv i ion in A^i, n on in involv ion . I o no n o iv ion o on in o i o no li .

Bo o o n on i ion n o i i l o i x n n - lly o i l li ion wi on in o i ion . T n ly i n n ion o i o i l o i i l o ; w o il .

T $\rhd$ on in i no o i l li ion, i $\alpha \rhd \beta$, n i x α l o now β. T o w n v ion i “ o l ” o i ion , S li M H v , no $\rhd\!\!\rhd \subseteq A \times A$. T ni ion o ili y n v n l on i n y n n x n in o i n w ▷ o o .

T i i l o i ov ll li ion only in o in lo (n y l) o ▷ n →. Un i l li ion, i o ion on in i i nn . W o M niv nn n n Sin l' i i no ion l o i [10] o i o .

6 Related Work

I i n l- o y o in o i i i li ion n o-o iv wo [ ], on ion n on in . x i n wi I ow l iv ly o l x li ion n ily n o in i -wo . I i ion l o i i n li n o n o i l l iv n n i y o ion n on in . Al o o l i NP- , I i n i i n n o x in l o lin i in o on .

v y o o i i i li ion [1] o iv o n n o-on li i n iff n w n o o ol .

on n H i [11] o o n li i ion l o i on on in i ion in i l , w i in i o l o i in S ion 5.

T l ion w n on i n y n o in v n w ll i in on x l n n l ion [1]. i l n o l i i iv l i y n n li i n ly i . T i i iv o on o ll o o ol , i ni n v n o ion o in n , , i li , i n l .

L o ' - in li ion [ ] o ion o ll i n n on i n y i x x ly l . MSP o y n li i ni ion. So l. [13] n li L o ' - in o o o i n o i lly li .

M o l wo on on i n y o on i li ili y. M ili y on-i n li ion o i li ili y.

T X-A ili y o y [1] llow n ion o v l i in l i i i i o n ; o in n , yin il ion i llow . S l o iv l n l in o li . I wo l in in o n o i o in o o li , n n ly i ion , w i i on . T i i l o wo .

o ny i il i i wi A wo [15]. A o-vi o lo i l i i iv ov x ion i o i , in l in n o n v n , i li ion, n l n n n o in w n v n . A ion i o , xi n o n ion o i n o i i iv . T A i ion l n i o ow l n i o n ly o o ol n n l i y. n o n , ion- on in l n i i l ; i i i o w o n l o o A n n i in o o l n . A i li ili y -ni ion o on i n y, n o no l wi i l li ion.

7 Conclusions and Future Work

W n o li o i in li ion o o ol n on i n y. i ni n o on o ny li ion o o ol n i in o l n . W n li n o l i l o l ion o on i n y o y n ov iv l n . T i n o o on li i w n o o ol i iff n on . Al- o on i n y n il lo l on n in n l , w x i i i n on i ion o in lo l i ion . W iv n w i i i ion l o i , w i o l i l i i , on in o i- i , n n x n o n l i l li ion. l ly o o n o o o ol , o i i i n o i i i .

T i only n in i ion ; in will n lly o l n in o ni l o [ ]. T o l o on in il i ion o v i y o iv l i l li ion o o ol , in l in on- i n y oo .

T o li on only wo in y on in . T i i y o ov o i , n i ow l no o in o o ll l i l li - ion o o ol . How v n i o o li ion (. ., n o n) n o ow l i i iv . A o i l i ion i o n li on in o n- y n o i ni n o n . T n i l y o y wo l n n i join .

Acknowledgments

W n F i l F n o i i i ion o ly o i wo , Y on n Yo H i o i on i ion on n li i- ion l o i , n Tony Ho , Mi l o n P i V l i o i n o n n ion .

References

1. Saito, Y., Shapiro, M.: Optimistic replication. Computing Surveys (2005).
2. Shapiro, M., Bhargavan, K.: The Actions-Constraints approach to replication: Definitions and proofs. Technical Report MSR-TR-2004-14, Microsoft Research (2004).
3. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. Trans. on Comp.-Human Interaction **5** (1998) 63–108.
4. Preguiça, N., Shapiro, M., Matheson, C.: Semantics-based reconciliation for collaborative and mobile environments. In: Proc. Tenth Int. Conf. on Coop. Info. Sys. (CoopIS), Catania, Sicily, Italy (2003)
5. Shapiro, M., Preguiça, N., O'Brien, J.: Rufis: mobile data sharing using a generic constraint-oriented reconciler. In: Conf. on Mobile Data Management, Berkeley, CA, USA (2004).
6. Birell, A.D., Levin, R., Needham, R.M., Schroeder, M.D.: Grapevine: An exercise in distributed computing. Communications of the ACM **25** (1982) 260–274

7. Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.H.: Managing update conflicts in Bayou, a weakly connected replicated storage system. In: Proc. 15th ACM Symposium on Operating Systems Principles, Copper Mountain CO (USA), ACM SIGOPS (1995).
8. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21** (1978) 558–565
9. Fekete, A., Gupta, D., Luchangco, V., Lynch, N., Shvartsman, A.: Eventually-serializable data services. Theoretical Computer Science **220** (1999) 113–156
10. Manivannan, D., Singhal, M.: An efficient distributed algorithm for detection of knots and cycles in a distributed graph. IEEE Transactions on Parallel and Distributed Systems **14** (2003) 961–972
11. Chong, Y., Hamadi, Y.: Distributed IceCube. Private communication (2004)
12. Ramamritham, K., Chrysanthis, P.K.: A taxonomy of correctness criteria in database applications. VLDB Journal **5** (1996) 85–97
13. Sousa, A., Oliveira, R., Moura, F., Pedone, F.: Partial replication in the database state machine. In: Int. Symp. on Network Comp. and App. (NCA'01), Cambridge MA, USA, IEEE (2001) 298–309
14. Frølund, S., Guerraoui, R.: X-Ability: A theory of replication. In: Symp. on Principles of Dist. Comp. (PODC 2000), Portland, Oregon, USA, ACM SIGACT-SIGOPS (2000)
15. Chrysanthis, P.K., Ramamritham, K.: ACTA: The SAGA continues. In Elmagarmid, A.K., ed.: Database Transaction Models for Advanced Applications. Morgan Kaufmann (1992) 349–397

Analyzing Convergence in Consistency Models for Distributed Objects

F n i o J. To - oj[1] n n M n[2]

[1] Costa Rica Institute of Technology (I.T.C.R.) and
University of Costa Rica (U.C.R.)
torres@ic-itcr.ac.cr

[2] Costa Rica Institute of Technology and PrediSoft,
Costa Rica
emeneses@ic-itcr.ac.cr

Abstract. At instant t, two or more sites could perceive different values for the same distributed object `X`. However, depending on the consistency protocol used, it might be expected that, after a while, every site in the system should see the same value for this object. In this paper, we present a formalization of the concept of *convergence* and analyze its relationships with several consistency models. Among other things, we claim that, by itself, *sequential consistency* is not a convergent protocol.

1 Introduction

In o o l wi v l, o i ly iff n , o i o li o o j in iv i o i i y , i i n y o n on- i n y o o ol. n wo l x on i n y o o ol o l off o in o n o onv n , n o y, o o j . T , onv n i l o n oo i n o o n o i i o ion. How v , on i n y o l [15], w i i lly “ on on i n y”, o **not** on in no i ly, y i l , onv n o o j . I i o lin ny x ll n i l n ion o i on i n y o l lly i in o li in onv n o in o ion, n no o j i yin i (n ini l) i l i n o n- i l on i n y, w i i v l o lin w n onv n n on i n y. In i , w li i ni ion o w w n n y onv n n n ly v l w ll- nown on i n y o l o li o ni ion .

i n i o n in l i l o i n . I i o n l o o in o ili y. Al o o ion in o l v in i o ni ion in n n o n ly y , i o- o in nowin o o j o y i v l vio iv n in n . M i lly, ni ion o onv n i lly o i o o . Fo x l , in l n , n $\{x_n\}$ onv

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 346–356, 2005.

o i, o v y $\epsilon > 0$ xi o n l n o $\geq$ i n $|x_n - x| < \epsilon$. A i n o v , n i o li i w i , v y in o “ l ”. In i , ili y o n i i y in i n ly lo o n .

In on x o o w y , o o [9,1] n onv n y loo in n l l o wo ion. ion y o l iv iff n i o o i , x in o i ly in iff n o . How v , i i i n l l x ly o v y . I i l o i o n off in onv n in o il o in li ion [10], i lly w n i onn ion io on i . In i , o ion (o i ly onfli in) ov iff n li o o j , i i i ll li onv o ll o v n onn o i n ly lon . T i i on o y i w x lo in i : in v l o o j in i i y o . Mo ov , w on i o o j n l o n o n i n .

o o n n in onv n in i i y i in o- in S ion . W y onv n o i o o on i n y o l in S ion 3. Fin lly, on l ion o i n in S ion .

2 Convergence Model

on i i i i o y own in Fi 1. A X i y on o i , n w v l i o ni , wi o l y, o o i . Si X i t_1 ivin i v l 3 (w i w 0 ini i lly). I i no n il i t_2 Si i ov X n . How v , i t_3 Si n w n o X, ivin i v l o 7. L ’ y n w o i n iv ool o Si , n y i t_4, Si in X o v l 4. Si il ly, Si o no iv i l n n X i t_5 o v l 6. A i t_6 Si li X n w v l n o on, o i on v l o X. T , n lly, onv n n .

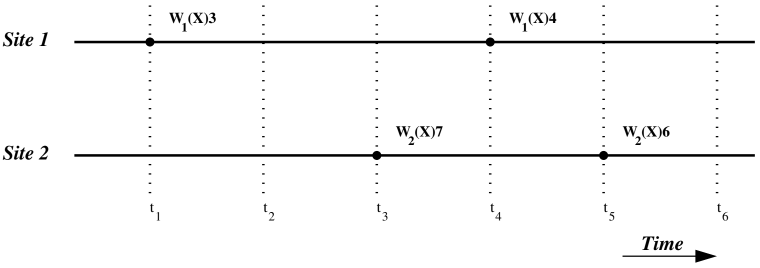


Fig. 1. Two sites in a convergent execution

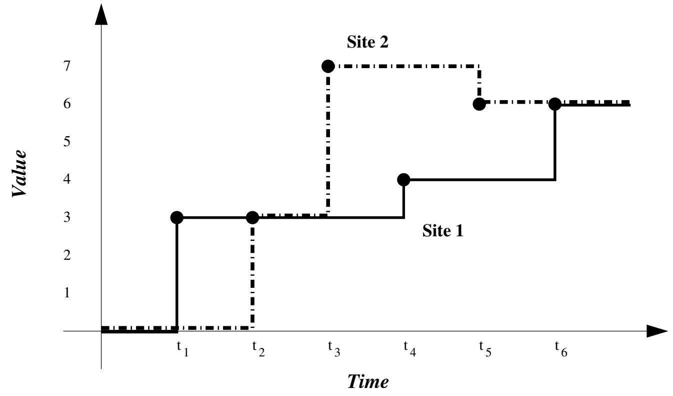


Fig. 2. Convergence of two sites

Fi lo v l X v y in n , iv y i , n i ow i onv n i t_6. N v l , i o l l i i in v l [0,t_1],[t_2,t_3] n [t_6,+∞] o o n w onv n w i v , w o i n (S ion .3). o l ni ion o onv n will wo in o ili y n : on o in o o i i o j , n i w wo o o i on v l o i l o j .

2.1 Trivial Convergence

Definition 1. **trivially convergent over object** X X *t* **trivially convergent**

T i i l in in o onv n , in i ion li i ly n n l , n v n o y i in onv n in v l o X, i o no i ly o n in x ion.

2.2 Absolute Convergence

Fi 3 ow i l i i o ion, wi i n on o j X. T i o **writes** n **reads** x y o i , n , v l i in x ion, i iff n v l o o j X. B , o i v n **w** (X)4, w i n o l li ion o X in w ol x ion, ll **reads** o o j X x y ny i n v l 4. T , i *t*, i y onv in v l o o j X. T in i ion in i , n o y, **writes** o , v y i involv in i i o ion will on v l o o j . No i i i il o on o v n l on i n y, i i o o o i in i w n onv n n on i n y.

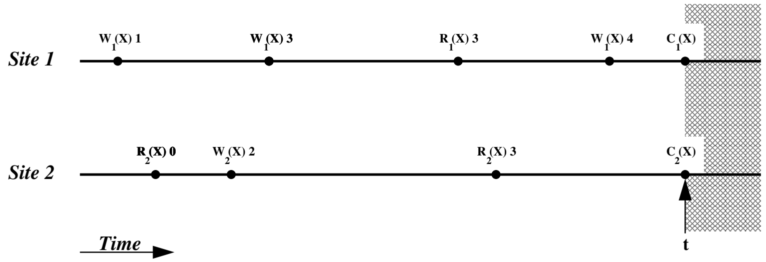


Fig. 3. Convergent cut

Followin lin o [16], w n i w y:

Definition 2. **convergent cut over object** X $\mathcal{C}$ $\{C_1, C_2, \ C_N\}$ C_i $\mathcal{H}_i$ C_i **read** X **write** X

Definition 3. **absolutely convergent over object** X **write** X onv n ov o j X **absolutely convergent**

I n x ion i ol ly onv n ov o j X i , i. ., w n in onv n $\mathcal{C}$ i , i $\mathcal{C}$ n in , wi i n i l l , ny i $u > t$. Now, i o j X_j, $1 \leq j \leq M$, n x ion i ol ly onv n o ll o j , n o v y o j w o i ini i $_j$ w i o on in onv n n in . T o , i i y i ol ly onv n ny i $t = max(t_1, t_2, ..., t_M)$.

2.3 δ-Convergence

I i y i lly i l l o n i o ni o v y o y l in i i y o o i l . How v , in v y iv y wi n **writes** o o j , i i no l v l o o j iv in x ion. v n n i n , w o l x "l " **write**, n ion in vio ion, y . B i , i y , on i in o ov , o ni ion l y , n on i n y o o ol , n n n i nown o o l y (i y in o y inv li ion) in o δ ni o i , x ion i ni in v l w y i vi n ly onv n in l ion o o o j .

Now, w l i i l w n l i l on iv **writes** o o j o n δ, w no no i o o in v l y ll **writes**, y n ill l i onv n . onv ly, i wo on iv **writes** o o j X o o n δ ni n w **not** l o in onv n o i o j l δ ni o i **write**, y i no onv n . T i i in i ion o w w ll δ- onv n :

Definition 4. **δ-convergence** **writes** X δ X δ ***write***

T , in δ- onv n x ion, i X i i n n x o i o j , nyw in y , o i , wi $t+\delta<u$, i n in v l $[t+\delta,u]$ w ll i in y wo l iv , i y i , v y v l o o j X. W ll i in v l o o j X. n o n , i $t+\delta>u$, w i no n onv n , w ill l i y i δ- onv n . In o wo , y i llow o " n l " o o δ ni o i **write**, wi o in on i non- onv n .

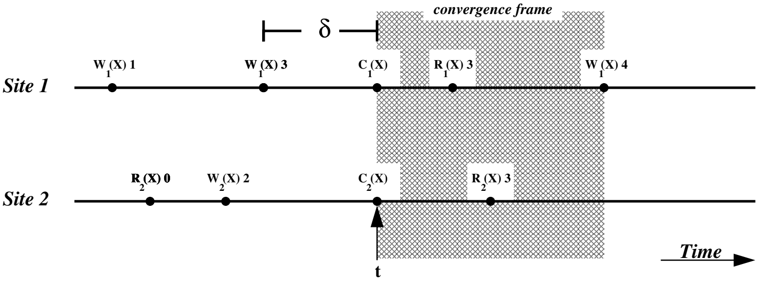


Fig. 4. Convergence frame

Fi ow o o vio on . I δ ni o i o ion **w** (X)3 o , w l o in onv n o i o o j X (w i n i v y i in y wo l X ll y wo l n v l), i li onv n o o j X. o , no o o j X n , n il n w y onv on v l o X. x n in i on o v l o j i i o w .

3 Convergence and Consistency Models

on i i i y wi i in o j . T $\mathcal{H}$ o i y i i lly o o ll o ion o in ll i . $\mathcal{H}$ i o l o o n o o ion x on i

. In o o i li y, w ll o ion i **read** o **write**, v l wi n i ni , n ll o j v n ini i l v l o o.

I $\mathcal{D}$ i o o ion , n o $\mathcal{D}$ i lin n on inin x ly ll o ion o $\mathcal{D}$ **read** o ion o i l o j n v l wi n y o n (in o o) **write** o ion o o j . I $\prec$ i n i y i l o l ion ov $\mathcal{D}$, w y i li ion $\prec$ i $\forall$ **a**, **b** $\in \mathcal{D}$ **a** $\prec$ **b** n **a** **b** in .

3.1 Convergence and Linearizability

T **read** n **write** o ion $\in \mathcal{H}$ ni , non- o i o x , o i i l o in n w n **read** o **write** " " o o n w n o ion " ni ". N v l , o o o i , w o i o o ion n in n (o oin w n n n), ll o o ion. I **a** n ff iv i vio o ff iv i o **b** w no i **a** $<_{\mathtt{E-T}}$ **b**.

Definition 5. $\mathcal{H}$ Lin i ili y (**LIN**) $\mathcal{H}$ $<_{\mathtt{E-T}}$ [11]

I i li ion o $<_{\mathtt{E-T}}$, i n ny **read** o o j $\mathtt{X}$ n v l o on in o o n **write** on $\mathtt{X}$. T o , **LIN** i n in n ly onv n o o ol. I i y o , n **LIN**, onv n o i o o j n lw y in i l **write** in o o j , w i ov x ion i (D ni ion 3). Si il ly, no ow lo wo **writes** o o j , onv n n lw y in i i ly v y **write**, w i ov x ion i δ- onv n o ny v l o δ.

3.2 Convergence and Sequential Consistency

I **a** o o **b** in $\mathcal{H}$ w y **a** **b** in , n no i **a** $<_{\mathtt{PROG}}$ **b**.

Definition 6. $\mathcal{H}$ S n i l on i n y (**SC**) $\mathcal{H}$ $<_{\mathtt{PROG}}$ [15]

T , **SC** o no i **read** o ion n o n v l wi o l- i , j l o ny x ion i i o ion o ll i n n in i li ion i l o $<_{\mathtt{PROG}}$.

W l i **SC** o no i ly onv n . iv n non xi n o l- i i n in i on i n y o l, i no lly o o o inv li i lo l o j nl i i i o il o ly. How v , i v y o on i on ion o in

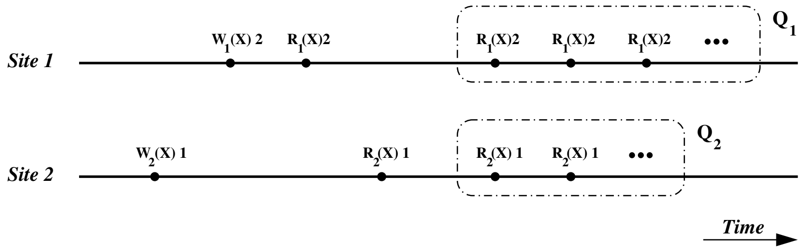


Fig. 5. A sequentially consistent history that does not converge

SC n onv n . No lly, w “ n i l on i n y” n “ on on i n y” in n ly, i o ly i ny o o o ol o in **SC** on x ion lly i o in o i n (wi i ion l ov) n w lly n o i y ni ion o **SC** in [15].

Fi 5 n i l x l o i i x ion i- **SC**, w o o j n v onv . Si w i v l 1 in o o j X, n , o i l , Si w i v l 2 in o o j X. T ollowin **read** on Si n 2, w il n x **read** o ion on Si n ol v l 1. No i , on i in only o v n , i o y, o , i **SC**: j = **w** (X)2, **r** (X)2, **w** (X)1, **r** (X)1. T i i li - ion o no l- i , l ll i n o **SC** [15]. Now, no in o Si n Si o , ny oin in , on v l o X. on i o ion $\mathcal{Q}_1$ n $\mathcal{Q}_2$, o on inin j **reads** ov o j X, on x in on Si n o x in on Si , - iv ly. T v l i v y o ion in $\mathcal{Q}_1$ i 2, w il v l i v y o ion in $\mathcal{Q}_2$ i 1. I n ov y in ion ov $|\mathcal{Q}_1|$ n $|\mathcal{Q}_2|$ i i i i o y i n i lly on i n . A o i l i li ion i = **w** (X)2, **r** (X)2, $\{\mathcal{Q}_1\}$, **w** (X)1, **r** (X)1, $\{\mathcal{Q}_2\}$. T , Si n n x - n in ni n o **reads** ov o j X, i y **SC**, n n v onv o v l . In i x l , i no n y **SC**, n , i w oo **w** (X)2 n **w** (X)1 o in o n δ ni o i , n i δ- onv n i i . I i y o il n x l o o - l x n Fi 5, involvin l i l i , o j n v l w i n, w **SC** i , n w ol onv n n δ- onv n n v .

3.3 Convergence and Causal Consistency

L $\mathcal{H}$ + o ll o ion in $\mathcal{H}$ l ll **write** o ion in $\mathcal{H}$. T i lly o l ion i “→” o in y n in [1] n o i o o o ion o $\mathcal{H}$. L **a**,**b** n **c** $\in \mathcal{H}$, w y **a** → **b**, i. ., **a** n - o (o) **b**, i on o ollowin ol :

1. **a** n **b** x on i n **a** i x o **b**.
. **b** n o j v l w i n y **a**.
3. **a** → **c** n **c** → **b**.

Definition 7. $\mathcal{H}$ l on i n y (**CC**) i $\mathcal{H}_{i+w}$ → [3]

CC i on i n y o l w n **SC**, i. ., v y n i lly on i- n x ion i l o lly on i n , v i no . I n i l n i n ly [3, 19]. **CC** i ll lly l o ion n in o y ll i , w il iff n i o l iv on- n o ion in iff n o [3]. **CC** n own o i n o li ion o yn ono in on i i . I n x lo o in in y [ ] n in o y n o j y [, , 5, 1 , 13, 19]. l ion w n **SC** n **CC** v n i in [3, 1]. iv n xi n o on n w i in n x ion, **CC** n no n ol onv n no δ- onv n .

3.4 Convergence and Timed Consistency

(**TC**), o o in [0], i i ff iv i o **write** i ., v l w i n y i o ion vi i l o ll i in i i y y i . + Δ, w Δ i o x ion.

Definition 8. **a**, **b** $\in \mathcal{D} \subseteq \mathcal{H}$ t_1 t_2 X **a** $<_\Delta$ **b**

a **b** **write** $t_1 < t_2$
a **write** **b** **read** $t_1 < (t_2 - \Delta)$

Definition 9. $\mathcal{H}$ Ti on i n y (**TC**) $\mathcal{H}$ $<_\Delta$ [0]

Un **TC**, **read** o no n l v l i o n v l v n v il l o o n Δ ni o i . I n n w n Δ = 0, **TC** o **LIN**. So, **TC** n on i n li ion o w nin o **LIN**. in n i wo iff n o on i n y. n voi onfli w n o ion , o ow i ly ff o n o ion iv y o y [6, 0].

T x ion ow in Fi 6 i **SC** n **CC**. U o on o ion o Si , x ion i **TC** o v l o Δ n in i , , y in n , **LIN** i no lon i . A i oin , x ion n i i **TC** **read** o ion in Si o n Δ ni o l- i Si w i v l **7** in o o j X n **read** o ion o no n i v l .

Definition 10. $\mathcal{H}$ Ti S n i l on i n y (**TSC**) $\mathcal{H}$ $<_{PROG}$ $<_\Delta$ [0]

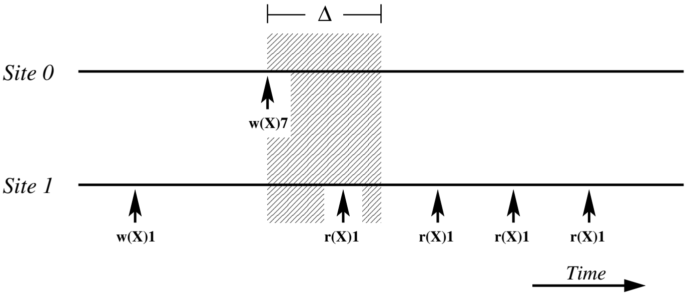


Fig. 6. Distributed History does not satisfy **TC**

Definition 11. $\mathcal{H}$ Ti l on i n y (**TCC**) i $\mathcal{H}_{i+w}$ → $<_{\Delta}$ [0]

W l i **TC**, **TSC** n **TCC** o n onv n . Fi ow x ion n vio ly in Fi 5, in l in i n o **TC**. A i n i , o ion **w** (X)2 i n in v l o Δ ni o i ni o Si , n ily, w o i n . How v , i oin o i now l v l o X, n , o , ll **reads** in $\mathcal{Q}_1$ n $\mathcal{Q}_2$ will o v l **2**.

Theorem 1. ***TC*** ol onv n δ- onv n

A o Δ ni o i l **write** o v y o j , **TC** n v l i nown o v y i in i i y , o , w n in onv n ov o j Δ ni o i o on in l **write** o ion, w i o in o D ni ion 3 ov x ion i . Now, i o l y o δ- onv n i n o v l $\delta = \Delta$.

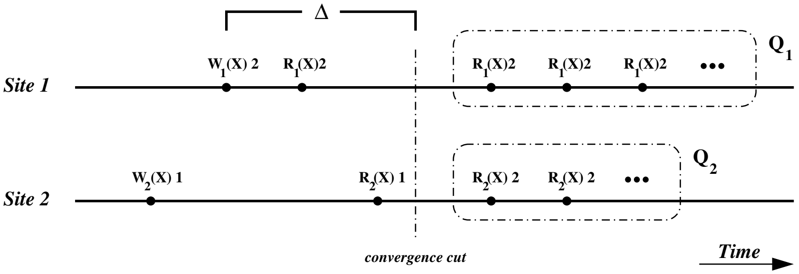


Fig. 7. A timed consistent history

4 Conclusions and Future Work

onv n i v y l n i o n no ion in i i y . In
i , w n o li ion o on n o ni ion
o n ly vio o v l w ll- nown on i n y o o ol . W o n
n v i n o , o l- i
on i ion in l in ni ion o o l , n
n δ- n in i . n o n ,
n n no l i .

iv n o l i y o **SC**, l ion n i in . T i n
n l in i i i w y (i. ., wo yin o o ol in
SC, no n ily i li onv n nl i x li i ly in l
on i ion), o in n o i i i w y (i. ., in v n o l o
n o y onv n o **SC** o o in o i n on i n y o o ol
n o n o li ion only i ini l
v ion o **SC**).

Fo wo w w n o x lo i n i l n ion o **TC**. W
l o y i li ion o o no ion o onv n in v l li ion
i i o i , oll o iv y , o il o in
n i i . B i , w ill n o lly n n vio
o o o ol **TCC**, , o vio ly i o n i
w n w ov o **CC** o **TCC**.

References

1. Adve, S. and Gharachorloo, K., *Shared Memory Consistency Models: A Tutorial.* Western Research Laboratory, Research Report 95/7, 1995.
2. Ahamad, M., Torres-Rojas, F., Kordale, R., Singh, J. and Smith, S.,*Detecting Mutual Consistency of Shared Objects*, Proc. of International Workshop on Mobile Systems and Applications, 1994.
3. Ahamad, M., Neiger, G., Burns, J., Kohli., P. and Hutto, P. *Causal memory: definitions, implementation and programming.* Distributed Computing. September, 1995.
4. Ahamad, M., Bhola, S., Kordale, R. and Torres-Rojas, F., *Scalable Information Sharing in Large Scale Distributed Systems*, Proceedings of the Seventh SIGOPS Workshop, August 1996.
5. Ahamad, M., Raynal,M. and Thiakime, G., *An adaptive architecture for causally consistent services*, Proc. ICDCS98, Amsterdam. 1998.
6. Ahamad, M. and Raynal, M., *Ordering and Timeliness: Two Facets of Consistency?*, Future Directions in Distributed Computing, 2003.
7. Attiya, H and J. Welch, J., *Sequential Consistency vs. Linearizability*, ACM Transactions on Computer Systems. Vol 12, Number 12. May 1994.
8. Birman,K., Schiper, A. and Stephenson, P.,*Lightweight Causal and Atomic Group Multicast*, ACM Transactions on Computer Systems, Vol 9, No. 3, pp. 272-314, Aug. 1991.
9. Ellis, C.A. and Gibbs, S.J. *Concurrency Control in Groupware Systems.* In ACM SIGMOD'89 proceedings,pages 399-407, 1989.

10. Guerraoui, R. and Hari, C. *On the Consistency Problem in Mobile Distributed Computing*. ACM POMC, 2002.
11. Herlihy, M. and Wing, J. *Linearizability: A Correctness Condition for Concurrent Objects*. ACM Transactions on Programming Languages and Systems. Vol 12(3), July 1990.
12. Kordale, R. and Ahamad, M. *A Scalable Technique for Implementing Multiple Consistency Levels for Distributed Objects*, Proceedings of the 16th. International Conference in Distributed Computing Systems. May 1996.
13. Kordale, R. *System Support for Scalable Services*, Ph.D. dissertation, College of Computing, Georgia Institute of Technology. January 1997.
14. Lamport, L. *Time, Clocks and the Ordering of Events is a Distributed System.* Communications of the
15. Lamport, L. *How to make a Multiprocessor Computer that correctly executes Multiprocess Programs.* IEEE Transactions on Computer Systems, C-28(9), 1979.
16. Mattern, F. *Virtual Time and Global States of Distributed Systems.* Proceedings of the International Workshop on Parallel and Distributed Algorithms, 215-226, 1989.
17. Raynal, M. and Schiper, A., *From Causal Consistency to Sequential Consistency in Shared Memory Systems*, Proceedings 15th Int. Conference FST & TCS (Foundations of Software Technology and Theoretical Computer Science), Springer-Verlag LNCS 1026, pp. 180-194. Bangalore, India, Dec. 1995.
18. Sun, C. et al. *Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems.* ACM Transactions in Computer-Human Interaction, 5(1):63-108, 1998.
19. Torres-Rojas, F. J., Ahamad, M. and Raynal, M. *Lifetime Based Consistency Protocols for Distributed Objects.* Proc. 12th International Symposium on Distributed Computing, DISC'98, Andros, Greece, September 1998.
20. Torres-Rojas, F. J., Ahamad, M. and Raynal, M. *Timed Consistency for Shared Distributed Objects.* Annual ACM Symposium on Principles of Distributed Computing PODC'99, Atlanta, Georgia, 1999.

Directional Versus Omnidirectional Antennas for Energy Consumption and k-Connectivity of Networks of Sensors*

v n lo K n i [1], D nny K i n [2], n i Willi [2]

[1] School of Computer Science, Carleton University,
Ottawa, Ontario, K1S 5B6, Canada
[2] Department of Mathematics and Computer Science,
Wesleyan University, Middletown CT 06459, USA

Abstract. A network is k-connected if it remains connected after the removal of any $k-1$ of its nodes. Assume that n sensors, modeled here as (omni)directional antennas, are dropped randomly and independently with the uniform distribution on the interior of a unit length segment or a unit square. We derive sufficient conditions on the beam width of directional antennas so that the energy consumption required to maintain k-connectivity of the resulting network of sensors is lower when using directional than when using omnidirectional antennas. Our theoretical bounds are shown by experiment to be accurate under most circumstances. For the case of directional antennae, we provide simple algorithms for setting up a k-connected network requiring low energy.

1 Introduction

o ni ion n wo li in in i o i n n i y ovi in i o in o ion. N w n o y n ly n- v lo n o i i y ovi in ili y o n - ion, ono o ly, in n lly x n o l x nvi on n . T y l o v n o li ion in l - i in , n o ion, in n n- i , in oxi n , w ll oni o in i y o ivil n n in in in .

S n o low ow o ni ion n n in vi n - in y i l wo l (K n l [ ], So i l [1], in l [5]). L l n o n wo o y n o n o i lly on in o ov iv n ion. I i x o o vi will o i ni n ly in n ([1], A l [1], n W n l [ ]). S n o no n l ono y, l - on ili y, n

* Research of the first author supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 357–368, 2005.

l-w n , in n y n l lv o i lly,
yn i lly o il , n ov n , n o n in n -
wo i n . How v , l n ionin o in ivi l n o y w ll l
o o ion l il l in i in i onn n wo o ilin o
oni o in ion.

o l o o in n y on ion -
w n n wo o o ni i ion l n i ion l n o n ion
o in inin n wo onn ivi y. A n n o o n o ly
n in n n ly wi ni o i i ion ov ion (w i w
on i i ni l n n o ni). A n wo i k- onn
i i in onn ov l o ny $k-1$ o i no . W inv i
i o i o ili y i o n o (iv n n ion
o o l n n o n o n n k o l) on k- onn ivi y
o n o y . W i n ly i o o n y on ion
i o k- onn ivi y o l in n o n wo w n in o ni-
i ion l v i ion l n o . l ow i ni n vin
o i l w n i ion l n nn ov o ni- i ion l n nn ,
in wi o i ion l n nn i i n ly ll. W
o o i l o n on xi wi llow l in o
o v n y n o o o i l l o x i n lly iv
o n . A o iv ion o o o n o i ion l n nn ,
w n i l l o i o i vin k- onn ivi y in n o n wo .

1.1 Model of Sensors

W on i wo y o n nn : o ni i ion l n i ion l. T o
n i i i n l ov 360 n l n , o o o i ,
ny n o wi in ili y i o i n o will iv i n l.
T l i ion l n nn n i n v iv n
wi α. T y n o o i in on “ wiv l” n
o i n ow o iv l n ly n o l i l
n nn o yin o wi wi α o o ov 360
n l (in $\lceil \pi/\alpha \rceil$ o n nn wo l). How v , n o o
no n ily v o iv ll n nn i . In , i
will i n i o in “ ion o no ” y iv in o i
n nn o o ov ion in iv n i ion.

1.2 Energy Consumption

In ny wi l n wo i n l n i n iv wi i n
n in o o o ly n in . Fo ny in o n-
i , wi l i i n l i n ll off wi i n ov
n i ion i . Al o n ion i in n l o l x n ion o
i n n o o , i ni n o i n l
ion i i ly w i i o i n l
ov n v l . Fo n i l i o o i n nn lo i
io o n i o iv ow n i l o $\frac{(4\pi d)^2}{\lambda^2}$,

w λ i i w v l n , n d i o ion i n w n nnn . In i l , n y i y n n nn o ll o wi in i i i o o ion l o ov . T , wi ili y i r n o ni i ion l n nn will on ow o o ion l o πr^2 (o i l wi i r) w il i ion l n nn wi wi α i n will on ow o o ion l o $\frac{\alpha}{2} r^2$, w y w i n l i n i ov i y lo n ow on y inin lo i n li i l . Fo i ion l in o ion on n nn o n n n [1] n on n nn o y B l ni [ ].

1.3 Results of the Paper

T o o i ivi in o wo ion . Fi , in S ion w on i w n o o on ni l n lin n . In S ion 3 w on i w n o o on ni . In o w ovi n o o i n ion l o i n n ly w y n y on ion o l in n o n wo . W l o iv i n on i ion on wi o n nn o i ion l n o on l n y o i v onn ivi y o l in n o n wo . T l 1 i o o i l l . In S ion w ovi x i n l n ly i .

Table 1. For the threshold value of the beam width indicated in the right column the energy consumption of a sensor network of n directional sensors is below the energy consumption of a sensor network of n omnidirectional sensors so as to achieve $(k+1)$-connectivity with probability at least $e^{-e^{-c}} - e^{-e^{c}}$

	Threshold beam width
Unit Segment	$\frac{\pi}{2} \cdot \left(\frac{\ln n + k \ln\ln n + \ln(k!) - c}{\ln n + (2k+1)\ln\ln n + c} \right)^2$
Unit Square	$\frac{2}{5(k+1)} \cdot \left(\frac{\ln n + k \ln\ln n + \ln(k!) - c}{\ln n + k \ln\ln n + c} \right)$

1.4 Related Work and Preliminaries

Di ion l n nn v no n x lo wi ly in on x o - o n wo . So n x lo in l i l n nn in o o ino , n l y n o in ov in l [9, 13, 1 ,]. To , ow v , w no w o ny wo on i o - i on o n y i n y o o ni i ion l v i ion l n nn wi o onn ivi y o i o n wo . l o o wo i o S o i l [0] w i ov n onn ivi y o i o l o o ni i ion l n o o yin v i o ni i n o o K n i l [10] w i inv i o n l o l o i ion l n o wi iv n wi o yin i y o i ion (o o o i oin) in in io o ni .

U l o o n ly i i o on oll o ' o l n i x n ion . In i l , n x n ion o o on oll o o l i o inin ol o n (no y $X^{(k)}$) o l ion (o on) i in o o oll l $k+1$ o i o o on y . I i w ll- nown (Mo w ni l. [1][x i 3.11]) ol i n $n(\ln n + k \ln\ln n)$, i. ., o ny in $k \geq 0$ n on n c,

$$\underset{n\to\infty}{\mathrm{li}}\ \mathrm{P}\ [X^{(k)} > n(\ln n + k\ln\ln n + c)] = e^{-e^{-c}}. \qquad (1)$$

I i l o no o $c > 0$ l no $e^{-e^{-c}}$ in i n i o ion 1 i i ily lo o 1 n o $c < 0$ l no i i i ily lo o 0.

V l l o o o i l n ly i l o i on ol o onn ivi y n ini no , w ll n l ol o ono on o i in o i i n o n in wo o P n o [15, 16, 1]. l o n n o n in [6, , 19, 3]

2 Sensors on a Unit Length Line Segment

In i ion w li i o ion o ni l n n . W on i $(k+1)$- onn ivi y n on n y on ion o o ni i ion l v i ion l n nn . Fo l i y o x o i ion, w onn ivi y n ly i o o ni i ion l n i ion l n nn .

2.1 Omnidirectional Sensors

A n o ni i ion l n o o n o ly n in n n ly wi ni o i i ion on in io o ni n . Fo ny in $k \geq 1$ n l n on n c l n o v i n i l i r, iv n y o l

$$r = \frac{\ln n + k\ln\ln n + \ln(k!) - c}{n}. \qquad (\ )$$

T in l o P n o [15][T o 1.1 n 1.] o on ni n n i iv n y I ni y

$$\mathrm{li}\ _{n\to\infty} \mathrm{P}\ [\text{n wo i } (k+1)\text{- onn }] = e^{-e^{c}}. \qquad (3)$$

T o oi l i n i iff o l i n i on ni n only in w o n o n y ff . T o Fo l 3 iv n o n on o ili y o i vin $(k+1)$- onn ivi y on ni n wi l i n i . T o w v ollowin o .

Theorem 1. r $k \geq 0$ $c > 0$ n

$$\mathrm{li}\ _{n\to\infty} \mathrm{P}\ [\ (k+1)\ ] \leq e^{-e^{c}}. \qquad (\ )$$

T , o i o n y Fo l n wo i $(k+1)$- onn wi o ili y in i y ion .

2.2 Directional Sensors

on i o i ion l n o wi in l n nn o wi α y o i n in ny i ion. (W no in o l i l n nn , n n y vin i ivi lly o i l o ny wi y in wo o o in n nn ov n in i i y o ni i ion l .) I i i ly y o ow y i in l n ly $k+1$ n nn o i lon n ollow y $k+1$ n nn o l n in in n o l $k+$ o n o , l in n wo i k- onn . Fo n o , w oo i o

$$r = \cdot \frac{\ln n + (k+1)\ln\ln n + c}{n} \tag{5}$$

n i ion ni in v l in o $\frac{2}{r}$ in v l o l n $\frac{r}{2}$. W i i in w y k- onn ivi y o l in n wo i - n . U in ion 1, i i y o li in v l on in $k+$ n o wi o ili y l $e^{-e^{-c}}$. N x w ivi n o in in v l in o wo (oxi ly) l i : l o l n i o l (Fi 1). Fo in v l w i l o l

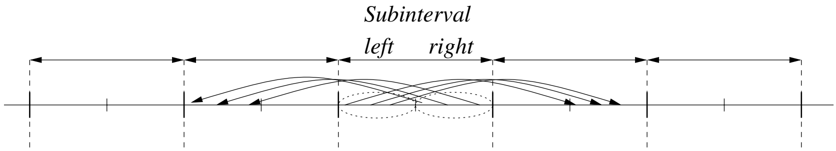


Fig. 1. Alternating the beam direction of the sensors from one subinterval to the next

o n o (l $k+1$) o i n i o l o n o (l o l $k+1$) o l . W n ov ollowin o .

Theorem 2. α, r $k \geq 0$ $c > 0$ n

$$\mathrm{li}_{n\to\infty} \mathrm{P}[\quad (k+1) \quad] \geq e^{-e^{-c}}. \tag{6}$$

2.3 Comparison of Energy Consumption

ni i ion l n o n i i n l ov n n l π. In o o i v $(k+1)$- onn ivi y l in n y on ion $\mathcal{E}^{(k)}_{\text{OMNI}}$ o n wo i

$$\mathcal{E}^{(k)}_{\text{OMNI}} \geq n \cdot \pi \cdot \left(\frac{\ln n + k\ln\ln n + \ln(k!) - c}{n}\right)^2,$$

y o i lly in n wi o ili y l $1-e^{-e^{c}}$. T i n on wi n y on ion $\mathcal{E}^{(k)}_{\text{DIRE}}$ i o i v $k+1$- onn ivi y o n wo o i ion l n o wi wi α (in i n). In i l , $\mathcal{E}^{(k)}_{\text{DIRE}}$ i

$$\mathcal{E}^{(k)}_{\text{DIRE}} \leq n \cdot \frac{\alpha}{\ } \cdot \left(\ \cdot \frac{\ln n + (\ k+1)\ln\ln n + c}{n} \right)^2,$$

y o i lly in n wi o ili y l $e^{-e^{-c}}$. A i l l l ion yi l y o i lly in n i

$$n \cdot \frac{\alpha}{\ } \cdot \left(\ \cdot \frac{\ln n + (\ k+1)\ln\ln n + c}{n} \right)^2 \leq n \cdot \pi \cdot \left(\frac{\ln n + k\ln\ln n + \ln(k!) - c}{n} \right)^2$$

n $\mathcal{E}^{(k)}_{\text{DIRE}} \leq \mathcal{E}^{(k)}_{\text{OMNI}}$. W ollowin l .

Theorem 3. n $k \geq 0$ $c > 0$

$$\alpha \leq \ \pi \cdot \left(\frac{\ln n + k\ln\ln n + \ln(k!) - c}{\ln n + (\ k+1)\ln\ln n + c} \right)^2 \qquad (\ )$$

$\mathcal{E}^{(k)}_{\text{DIRE}} \leq \mathcal{E}^{(k)}_{\text{OMNI}}$ n $e^{-e^{-c}} - e^{-e^{c}}$

3 Sensors on a Unit Square

In i ion w li i o ion o ni . W on i $(k+1)$- onn ivi y n on n y on ion o o ni i ion l v i- ion l n nn . W on i onn ivi y ly o o ni i ion l n i ion l n nn .

3.1 Omnidirectional Sensors

A n o ni i ion l n o o n o ly n in n n ly wi ni o i i ion on in io o ni . Fo ny in $k \geq 0$ n l n on n c l n o v i n i l i r, iv n y o l

$$r = \sqrt{\frac{\ln n + k\ln\ln n + \ln(k!) - c}{n\pi}}. \qquad (\ )$$

T in l o P n o [15][T o 1.1 n 1.] o on ni n i iv n y I n i y

$$\text{li}\ _{n\to\infty} \text{P}\ [\text{n wo i } (k+1)\text{- onn }\ ] = e^{-e^{c}}. \qquad (9)$$

T o oi l i n i iff o l i n i on ni only in w o n o n y ff . T o Fo l 9 iv n o n on o ili y o i vin $(k+1)$- onn ivi y on ni wi l i n i . T o w v ollowin o .

Theorem 4. r $k \geq 0$ $c > 0$ n

$$\text{li}_{\,n\to\infty} \text{P} \,[\quad (k+1) \quad] \leq e^{-e^c}. \qquad (10)$$

T , o i o n y Fo l n wo i $(k+1)$- onn wi o ili y in i y ion 10.

3.2 Directional Sensors

on i o i ion l n o wi $k+1$, w $k \geq 0$ i n in . Fix k n on n $c > 0$. P i ion ni in o $\frac{1}{r^2}$ o lo o i r, w

$$r = \sqrt{\frac{\ln n + k \ln\ln n + c}{n}}. \qquad (11)$$

L ili y i r' o i ion l n o l o l n o i on l o n l wi i n ion $r \times (r)$ (Fi), i. .,

$$r' = \sqrt{\frac{5(\ln n + k \ln\ln n + c)}{n}} = r\sqrt{5}. \qquad (1\ \ )$$

L $N^{(k)}$ n o v i l o n n o n o o o o on in $k+1$ n o . In vi w o I n i y 1,

$$\lim_{n\to\infty} \text{P} \left[N^{(k)} > \frac{1}{r^2} \left(\ln\left(\frac{1}{r^2}\right) + k \ln\ln\left(\frac{1}{r^2}\right) + c \right) \right] = e^{-e^{-c}}. \qquad (13)$$

Now n n o o i r (iv n in ion 11) o on in io o ni . Sin $n > 1/r^2$, w v

$$n = \frac{n(\ln n + k \ln\ln n + c)}{\ln n + k \ln\ln n + c} > \frac{1}{r^2} \left(\ln\left(\frac{1}{r^2}\right) + k \ln\ln\left(\frac{1}{r^2}\right) + c \right).$$

By ion 1 w v will v $k+1$ n o wi o - ili y l $e^{-e^{-c}}$. Now w ovi n " n nn o i n ion" l o i o i n o in w y onn ivi y in ni i n .

N n o in iv n $1, \ldots, t+1$. (A n o v ni i n i i n o y n o lv . T i n on in o in ll i ion .) Fo $i = 1, \ldots, t+1$ n o n i in o lv in o il oni n y l vi i v y in on o i $k+1$ n nn . S n o i in n i k inin n nn o oin n o $j \neq i$ in i . W l i l i $k+1$-onn . S y n o i in lo B w n o l o n o j in lo C. I k n o il, i ill il oni n y l (y no n m) i o l ly liv . No i n i o no m in lo B, no m in lo B n o no m in lo C, w i in n n i o no j in lo C. T o n wo i $(k+1)$- onn . Two i o n oin ollowin

1. So lo v o n k no . T n o n i i i ily on y l .
. T lon ny n nn o i $r\sqrt{5}$, w r i i l n i o v $k+1$ no . (No : in wo n n nn o i on l o r y r n l .)

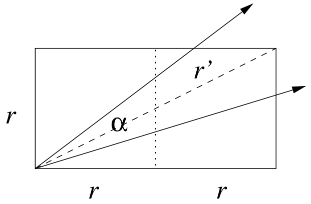


Fig. 2. The radius r' of the directional sensors is determined by the geometry of two adjacent subsquares. in particular it must be chosen so that $r' \geq r\sqrt{5}$

In i l , l o wi , l in y o i ion l n nn $k+1$- onn wi i o ili y. W n ov ol-lowin o .

Theorem 5. α, r $k \geq 0$ $c > 0$ n

$$\lim_{n\to\infty} \mathrm{P}[\quad (k+1) \quad] \geq e^{-e^{-c}}. \qquad (1\ )$$

W no i w no v ni i n i i (. ., i y n o i o i lo ion o PS y) n l o i in T o 5 n i l n i ly ily in i i nn . W no y in ll n o x n nn i n o in o j n n l o n i ov n in i o l in n o y o oxi ly n/k o $\sqrt{n/k}$.

3.3 Comparison of Energy Efficiency

In i ion w o n y on ion o n o n wo o n o ni i ion l v n i ion l n o o in $(k+1)$- onn ivi y. L $\mathcal{E}^{(k)}_{\text{OMNI}}$ n $\mathcal{E}^{(k)}_{\text{DIRE}}$ n y on ion in o ni i ion l n i ion l iv ly, o in $(k+1)$- onn ivi y.

ni i ion l n o n i i n l ov n n l π. In o o i v $(k+1)$- onn ivi y n o ni i ion l n o n y n l in n y on ion o n wo i

$$\mathcal{E}^{(k)}_{\text{OMNI}} \geq n \cdot \pi \cdot \frac{\ln n + k \ln\ln n + \ln(k!) - c}{n\pi},$$

y o i lly in n wi o ili y l $1 - e^{-e^{c}}$. T i n on wi n y on ion $\mathcal{E}^{(k)}_{\text{DIRE}}$ o n wo o i ion l n o wi wi α (in i n) w i y n i i n l ov n n l α. In i , $\mathcal{E}^{(k)}_{\text{DIRE}}$ i

$$\mathcal{E}^{(k)}_{\text{DIRE}} \leq n(k+1) \cdot \frac{\alpha}{} \cdot \frac{5(\ln n + k \ln\ln n + c)}{n},$$

y o i lly in n wi o ili y l $e^{-e^{-c}}$. I i l , wi i o ili y y o i lly in n i

$$\frac{5(k+1)\alpha}{} \cdot (\ln n + k \ln\ln n + c) \leq \ln n + k \ln\ln n + \ln(k!) - c$$

n $\mathcal{E}^{(k)}_{\text{DIRE}} \leq \mathcal{E}^{(k)}_{\text{OMNI}}$. A i l l l ion yi l ollowin o o -in n y on ion o n o n wo o n o ni i ion l v n i ion l n o o in $(k+1)$- onn ivi y.

Theorem 6. n $k \geq 0$ $c > 0$

$$\alpha \leq \frac{}{5(k+1)} \cdot \left(\frac{\ln n + k \ln\ln n + \ln(k!) - c}{\ln n + k \ln\ln n + c} \right) \tag{15}$$

$\mathcal{E}^{(k)}_{\text{DIRE}} \leq \mathcal{E}^{(k)}_{\text{OMNI}}$ n $e^{-e^{-c}} - e^{-e^{c}}$

4 Experimental Results

T o i l l n ov , w il ovi in o n on α, in on oxi ion n ol only y o i lly in n. In o o n i o ow o i w ovi l o o i l ion w o on o i l o l.

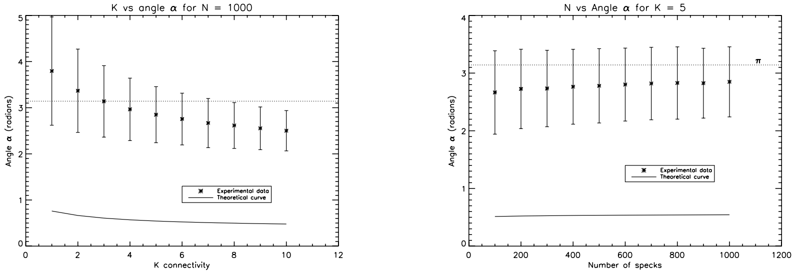


Fig. 3. Simulation results for unit segment

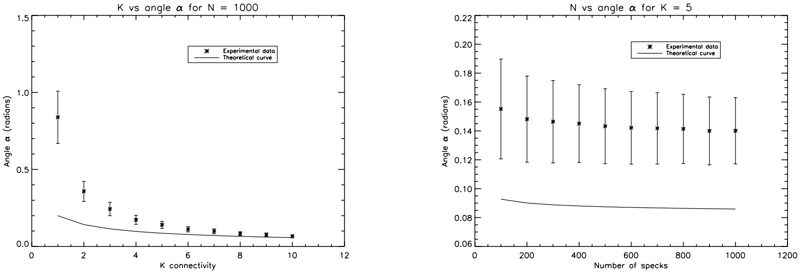


Fig. 4. Simulation results for unit square

Fo o on i n ion l n wo i n ion l w on i v lo k n in o 1 o 10 n v l o n n in o 100 o 1000 y 100. Fi 3 i i l ion l o ni n . Fi i i lion l o ni . x i n on i o o in n n o n o on i ni n o ni . Fo iv n k, low o n on i i o i v k- onn ivi y o o ni i ion l nnn w o in y n in i n o k- n n i o n n o n on i i o i v k- onn ivi y in i ion l n nn w o in in l o i i ov . x i n w 10,000 i n v o ll 10,000 n lon wi o i w o . T n y i n in w o in o l ov n io o n y w lo n o o o i l l . W n l o lo in α (xi wi llow l o i v n y vin) v n o $k = 5$ w ll lo in α v k o $n = 1000$. Plo o o v l o k n n i il .

W o v o i l o n i o v i w ll l o o ll v l o n n k y i ni n ly n i v l o α i n o n low n y o i ion l . Fo wo i n ion l , w o i l v oxi x i nl l i w ll k in . W il o i l i ion o l i ov in y n in , i l will lw y xi .

W i i o oxi ion in o o n o
i ion l y i ov in n ly i .

References

1. J. Agre and L. Clare, An integrated architecture for cooperative sensing networks, IEEE Computer, vol. 33, no. 5, May 2000, 106-108.
2. C. A. Balanis, Antenna Theory: Analysis and Design, 2nd ed. New York: Wiley, 1997.
3. D. Braginsky and D. Estrin, Rumor routing algorithm for sensor networks, 2001. Available at http://lecs.cs.ucla.edu/ estrin/.
4. L. Doherty, L. E. Ghaoui, and K. S. J. Pister, Convex position estimation in wireless sensor networks, in Proceedings of IEEE Infocom, (Anchorage, AK), April 2001.
5. D. Estrin, R. Govindan, J. Heidemann and S. Kumar: Next Century Challenges: Scalable Coordination in Sensor Networks. In Proc. 5th ACM/IEEE International Conference on Mobile Computing, MOBICOM'1999.
6. A. Goel, S. Rai, B. Krishnamachari, Sharp Thresholds for Monotone Properties in Random Geometric Graphs. Stanford University, Manuscript, 2003.
7. P. Gupta and P. R. Kumar, Critical Power for Asymptotic Connectivity in Wireless Networks. Stochastic Analysis, Control, Optimization, and Applications, Birkhauser, 1998.
8. J. M. Kahn, R. H. Katz, and K. S. J. Pister, Mobile networking for smart dust, in Proceedings of MobiCom 99, (Seattle, WA), August 1999.
9. Y. B. Ko, V. Shankarkumar, and N. H. Vaidya, Medium access control protocols using directional antennas in ad-hoc networks, Proc. IEEE INFOCOM'2000, March 2000.
10. E. Kranakis, D. Krizanc, J. Urrutia, Coverage and Connectivity in Networks with Directional Sensors. In proceedings Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Danelutto M., Vanneschi M., Laforenza D. (Eds.), Vol. 3149, Springer Verlag, LNCS.
11. S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, Coverage problems in wireless ad-hoc sensor networks, in Proceedings of IEEE Infocom, (Anchorage, AK), 2001.
12. R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
13. A. Nasipuri, S. Ye, J. You, and R. E. Hiromoto, A MAC protocol for mobile ad hoc networks using directional antennas, Proc. IEEE Wireless Communications and Networking Conference (WCNC'2000), 2000.
14. National Research Council, Embedded, Everywhere: A Research Agenda for Systems of Embedded Computers, Committee on Networked Systems of Embedded Computers, for the Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences, Washington, DC, 2001.
15. M. D. Penrose, On k-Connectivity for a Geometric Random Graph, Random Structures and Algorithms, 15, 145-164, 1999.
16. M. D. Penrose, The Longest Edge of the Random Minimal Spanning Tree, The Annals of Applied Probability, 7(2) 1997, 340-361.
17. M. D. Penrose, Random Geometric Graphs, Oxford University Press, 2003.
18. R. Ramanathan, On the Performance of Ad Hoc Networks with Beamforming Antennas, In the Proceedings of ACM Symposium on Mobile Ad hoc Networking and Computing (MobiHoc'2001), 2001.

19. P. Santi and D. Blough, The Critical Transmitting Range for Connectivity in Sparse Wireless Ad Hoc Networks, IEEE Transactions on Mobile Computing, to appear.
20. S. Shakkottai, R. Srikant, N. Shroff, Unreliable Sensor Grids: Coverage, Connectivity and Diameter, In proceedings of IEEE INFOCOM, 2003, held in San Francisco, March 30 to April 2, 2003.
21. K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie, Protocols for self- organization of a wireless sensor network, IEEE Personal Communications, vol. 7, pp. 16-27, October 2000.
22. A. Spyropoulos, and C.S. Raghavendra, Energy Efficient Communications in Ad Hoc Networks Using Directional Antennas, in proceedings of INFOCOM 2002, New York, June 23-27, 2002.
23. P.-J. Wan and C.-W. Yi, Asymptotic Critical Transmission Radius and Critical Neighbor Number for k-connectivity in Wireless Ad Hoc Networks, Mobihoc, 2004, to appear.
24. B. Warneke, M. Last, B. Leibowitz, and K. Pister, SmartDust: communicating with a cubic-millimeter computer, IEEE Computer, vol. 34, no. 1, January 2001, 44-51.
25. W. Ye, J. Heidemann, and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in Proceedings of IEEE Infocom, (New York, NY), June 2002.

Secure Location Verification Using Radio Broadcast

A n n Vo n Mi il N n o[*]

Computer Science Department,
Kent State University, Kent, OH, 44242
avora@cs.kent.edu, mikhail@cs.kent.edu

Abstract. *Secure location verification* is a recently stated problem that has a number of practical applications. The problem requires a wireless sensor network to confirm that a potentially malicious *prover* is located in a designated area. The original solution to the problem, as well as solutions to related problems, exploits the difference between propagation speeds of radio and sound waves to estimate the position of the prover. In this paper, we propose a solution that leverages the broadcast nature of the radio signal emitted by the prover and the distributed topology of the network. The idea is to separate the functions of the sensors. Some sensors are placed such that they get the signal from the prover if it is inside the protected area. The others are positioned so that they can only get the signal from the prover outside the area. Hence the latter sensors reject the prover if they hear its signal. Our solution is versatile and deals with provers using either omni-directional or directional propagation of radio signals without requiring any special hardware besides a radio transceiver. We estimate the bounds on the number of sensors required to protect the areas of various shapes and extend our solution to handle complex radio signal propagation, optimize sensor placement and operate without precise topology information.

Keywords: location verification, wireless sensor networks, security.

1 Introduction

T o l o lo ion v i ion i y S y l [1]. T o l i o on y i l n o in i l (ov) in oion on . Lo ion v i ion n o in , inv n o y, lo ion- on ol, . Fo x l , on -n o ov n on , i n n ivil onn ion o iv wi l n wo , in , o nin oo o i o i lin n l .

[*] This research was supported in part by DARPA contract OSU-RF#F33615-01-C-1901 and by NSF CAREER Award 0347485.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 369–383, 2005.

Related work. T lo in ion o o in vi wi y i l nvi on n i nov l o o i y. N i l. [ ] o i y ni o on in n n o y . Al n iv ly, in i w x loi o i o nvi on n o olv i y .

A n o o n on i o n o lo ion v i ion in wi l n o n wo [3, ,1]. T ny o o ol i v lo ion v i ion y x loi in iff n w n io i n l o ion n l - o n , . P i l ly, H l. o l l [5], B n l. i - o n ll n - on o o ol [6]. A li i ion o i n i y o i ly i n ili i n o i ly non- F o ni ion w on n o no .

B l n l. [3] lo ion-li i nn l o lo ion v i ion; owv , l o lo ion-li i nn l y i i ili y o i o . Mo ov , i o o no ovi ny on i y n [1]. o n n No l [ ] o - n o ni ion o v i y oxi i y. How v , i il i li io i l o n o in in ow l n i . Kin l. [ ] on in nn l o li i n i ion n o ov , i o o ol o no ovi on i y n i . T - i n w i in in y o ovi lo ion n i ion [9].

Our contribution and paper organization. W o o lo ion v iion o o ol li on o n o io o ni ion n oo ion o n o no . In i iv ly, on ov i io i n l, n o in i vi ini y will iv i n l, w il o n o will no . T n o no n n o i in o i ion , n in n o ov . o o ol i o i n , n i o no i x n n o ili i n o i -o -fli lo ion i ion o .

In n ion o w iv o i l i l o i l . T , w i ol ion o i l o l wi on ion o nvi on n n i y (. . i n l ion, o ni- i ion l n nn o). A w o no i i i i l n ion o o l o i . W n l x ion n x n o ol ion o o li i i ion. To o o w o no n o l y i l o o in in wi o i y no o o i . How v , in n o w i ow o o o ol n in o o in o y .

T i on i ion o i ollow . W lo ion v i ion o l [1] in S ion , in w y llow i o l n n n o ol ion . U in i i , w n n i o o ol o lo ion v i ion. W o lin i o i in S ion 3.

In S ion , w on n i y oly on l o ion on n o l ly wi $O(n)$ n o w n i n o i in oly on. T i o o ol y l v o in o ion o o ion on w ov y o y no (i i y on). In ion, w l o ow n i y (non- oly on l) on n wi $O(S+P)$ n o i i y on o i n o on n i n o n o , w S n P on ' n i iv ly.

In i o o ol, n o v i ion o ov i i o o ion l o i o on . In S ion 6, w ow i n n o lo i o on i y in x v i . In S ion , w ow ow ov n in i i y on wi x v i ion , n w l o i n o o o o ion l o lo i o o ion on i .

W ovi w x n ion o o i o o ol. In i ion o i l o o l in o ni- i ion l io i n l , w i n x - i i l o ion o n io o , in S ion , w x n o o ol o l wi o l x o o l, w i in o n o non- ini i ion o n o ion. In S ion 5, w ovi o i ion o n in v i i ion l io i n l o o o ol. In i v i l o n in i n l wi non- o in, w i i o o i n l o ion . In S ion 9, w ovi o o ol o lo ion v i ion w i y v i l n i in o l l , ini i l n . In S ion 10 w on l y i in ow o o o ol n x n o o l i y y .

2 Preliminaries

Definitions. T lo ion v i ion o l i o v i o - ov i i i lo in i n . A i n o l o o ni in wi o v i w ll ov . A i o il n i y in o o y v i . T v i ov , i i i n in o ion on n v o in o . wi , v i i ov o i no i ion.

T wo in o v i : n n . T l n i ivi in o on o in o v i ' ili y o lo ov : — ov in i on i lw y i i v o in o o ni ion l ; — ov in i on y o y no (l o ov ' n o o ni ion l); n — ov in i on i n v .

Fo i l o ion on v i ion o o ol i i v y oin o i o ion on i l o in j ion on . T v i o ion on . i xi i n w n oin in j ion on n n oin o i o ion on . No i i i n i only nin l o oin in i o ion on . H n , o ion i o ow j ion on n o on o ion on . P o ion i i o ion i o.

Assumptions and threat model. T v i l o o ni ly n li ly on lv . T v i . T i, li io n i y nno i i o ni ion w n v i o i on v i . W o no o on o ni ion i w n v i . T o o o , w on v i o i v il l o o v i n .

I v i n o ov , ov i lw y l o iv i . P ov n i ion i no i . T i , ny n i y oni wi v i i on i ov . T ov i l o oni io n i o io i n l o o n iy x i n . Bo i n l n i ion n ion in nn o .

W on i n io o ion o l o ov . In i o l, i ov n i n l, v y v i wi in o x i n o ov iv i , w il no v i i w y o . T i i n n on i n l n o ov . W l x o ni- i ion li y ion in S ion 5 n i l ion ion in Sion .

T ov y li io . A li io ov o no v o oly wi v i ion o o ol. M l i l ov y oll o v i ion o o ol. In o l i l ov , ov y l o yn oni i i n l ly n i wi i y. I ll li io ov in j ion on , non o i o o .

Problem statement. W o l n o [1].

( ) iv n lo o ion on , i y lo ion v i ion o o ol.

v only i n on o ion on i i lo , i. . on o no v o onn .

3 Location Verification Protocol

Verification protocol. v i ion o o ol l ollow .

x x

Basic Protocol Properties.

Lemma 1.

Proof: **If:** W ow w n l i l li io ov lo
in l , only i ion v i n i j . No
in li y o o li io ov i no li i . Al o, in i n l
n i ion i in n n o , w n on i i ion y ov
v y oin o w i o il ov n i n l. H n , w n i no
o ili y o ov .
A o in o o ni ion l , i ion i w n
l on o n no j o ov ' i n l. Fo -
o o i n l, i n l n o l i no o ov
i n o ov o o . How v , v y ov i no
o n j o n o n o . D o o i n l o ion
ion, i n o iv i n l o ov , n l on
j o v l o i . In i , o in o o ni ion
l , v i j ov . T , oin i l w y
o n o o n j o i in j ion on .

Only if: W ov on o i iv . S o o in oin *p* on
l n , i n o n o i l n o n
j o . L ov lo *p* n o wi ini l i n l
n n y o o o iv i n l. In i , o in
o i n l o ion ion , j o o no ov . By
o ni ion l o o o ol, ov i . By ni ion,
ov i n v in ny oin o j ion on . H n , *p* i no in
j ion on . T , o v y oin in j ion on i i n y o
l o n o o n j o . □

To o l o o lly, w n w o o ion l
o y. By ni ion [10– .5], v i ' Vo onoi ll i i
lo o i v i n o ny o v i . T , ny oin in j o '
ll (in l in o n y) i l lo o j o o n
o . T ollowin o ollow o L 1.

Theorem 1.

ll n o lo ion v i ion o l i
o ion on ni . A non- ivi l ol ion o o l n l on
o . F o T o 1, i ollow Vo onoi ll o o
ni . I n ily own ini n o o j (v i)
o o ni Vo onoi ll i o . Mo ov , o o j o only
on ni ll. H n ollowin o oll y.

Corollary 1. ( )

Lemma 2. (x)

v n o i l i no y i o o L 1. T “only i ” o L in n l o no ol .

Proof: L n o n n j o iv i n a n $b > a + x$ o oin o in . A o in o o - ni ion l , o iv i n l o ov $\lceil a/x \rceil$ i . H n , i n o i n l o ion i :

$$\left\lceil \frac{a}{x} \right\rceil x \;\leq\; \left(\frac{a}{x} + 1\right) x \;=\; a + x \;<\; b$$

T , w n n o iv i n l o ov , j o ill oo o ov o v l o iv i n l. □

v L 1 n lin n n j ion on only. Y wo on o no ov w ol l n . T inin i i i y on . In i on , v y oin i lo o n o n o j o iff n in iv i n i l n i n l in n . T on o xi n o i on i ollowin . T ov in n i i n l y x i i o . Fo ov in i i y on , i i o i l i n l i oo w o v i o iv i . Y w n i n l i in n y x n o , o n o n j o i . A o in o o o ol, v i j ov . How v , oin o i i y on lo o n o n o j o . H n , ov o no ollow o o ol y n i i n l n n o i v n o non o j o o. T , i ov i .

In ol ion o oll y 1 , o ion n i ily l . In , in n o v i i x , o o ’ Vo onoi ll i i i n o n y o o ion on n vi i ily o i . T ollowin l llow o l o ion o oly on l o ion on .

Lemma 3. n $n + 1$

Proof: L l n o n i y oin in o ion on . Al o, w l j o o i o o lin joinin i j o n o on in i o o ion on n . Sin o ion on i onv x, Vo onoi ll o only o o ion on . H n , nion o j o ’ Vo onoi ll ov o i o ion on . A o in o T o 1, o o ol i . By ni ion, o ion ovi y i l n o v i i o l .T o l n o v i i $n+1$.

Lemma 4. n r, $n+1$ $r-x/$

Proof: To i i o n on , o Fi 1. T o ion on on in i l o i r. W o i ion o n o i l n j o o i o ion on , i in oo o L 3. No L 3 ol l o x o i ion o o in i oly on. on i on n i o n i o i $r-x/$. T i n w n v y oin in i i n i n j o i n $r+x/$. H n , o v y oin o i , i n o o i l n o n j o y x. A o in o L , i i in i n on . □

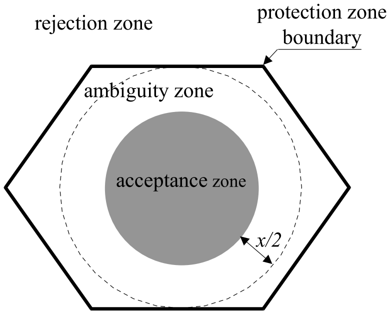


Fig. 1. Zone delineation in case of a polygonal protection zone. Illustration to the proof of Lemma 4

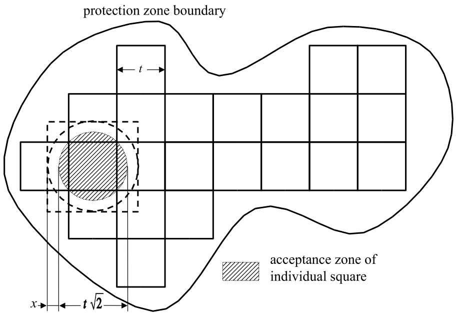


Fig. 2. Covering a zone of arbitrary shape with a constant ambiguity gap. Illustration for the proof of Theorem 3

4 Securing Arbitrary Zones

To i y o i y oly on , w x n o o o ol ollow . A o ion on y o o in o n o ll - on . T - on ly. In o wo , v i o on - on o no in wi v i o no . T ov i in on i i i y v i o l on o on i n - on . U in x n o o ol, w iv o n on n o v i n o o ion on o i y . W o l in ollowin wo o .

Theorem 2. n $O(n)$

Proof: T n o i n l i o i n l n n- i oly on i $n-$. A o in o L 3, i v i o i n l o l ly. T , o l n o v i i o n n- i o ion on i $n-$. T o ollow . □

v ol ion oo o T o , y o n- i lly l v n on i onn . T i y o li o i ionin o ov o n . T ollowin o o n n o v i n y o n i y o ion on n on i on in o n i o n y i wi in on n i n o o n y o o ion on . To i , w n o xi i n o oin in i i y on o n oin o i o ion on .

Theorem 3. S P $O(S+P)$

Proof: on i ll ion o ov o ion on .[1] o Fi o ill ion. L t l n o i o . W l t ll no o in ll ion i l on w o n i no l n $t+x\sqrt{\ }$ w y o n o . I i w ll-nown n o i in $O(S+P)$.

L i ll wi n l n $t+x\sqrt{\ }$ w y o o n on i o inin in ivi lly. By ion i l on . i i i l o n . I i i $t/\sqrt{\ }$. on i on n i i l wi i $t/\sqrt{\ }+x$. i - i ov i i l . T i n o n o oin in i i $t+x\sqrt{\ }$. By on ion, i o l ly in i o ion on . A o in o L 3, i 5 v i o i o l ly. Mo ov , o L in n l will in i n on . o o ll o ll ion. T o in n on i on in o , n i i y i no o n $t+x\sqrt{\ }$. Sin i on n n o v i o ov , o l n o v i i in $O(S+P)$. □

5 Directional Antennas

In i ion , w li io ov ollow o ni- i ion l o o l. M li io ov , ow v , y i wi i ion l n nn , llowin o non- o in in i l i- ion, y i o in o ion . A li io ov

[1] The proof does not depend on the shape of the polygons. The squares are used for simplicity.

n x loi i ion li y o i n l o v i . S ov i n ow o io i n l i n l voi ion y j o o . T , ov y viol i y o o o ol.

on i xi l o in i o ion o i i-ion l i n l. A i n l i ni ly iv in v y oin o i o . β i ini n l on o o on o o -ion o v io i n l n . W li io ov nno i wi i ily ll, i. . β i on n .

T ollowin l i iv l n o L 1. I i ov n i il ly.

Lemma 5. β, β

v ni n ov only o ni- i ion l n nn . H n , n i ion o L li o i .

Theorem 4. $O(r)$ r

Proof: on i i l o i $r-k>0$ i on n i wi i l in i in o ion on w i k i on n in n n o r. o Fi 3 o ill ion. Pl in l o in i l o i i l n j o on i i n i n o $k\cdot$ n$(\beta/\ )$ o o . v on i ion o L 5 i o v y oin o i in i i l . T o , v y oin o i o ion on i in j ion on . A o in o i ion o lo ion v i ion o l l n o v i o ion on .

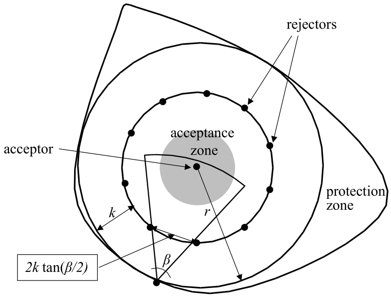


Fig. 3. Placing rejectors to protect against malicious provers with directional antennas. Illustration for the proof of Theorem 4

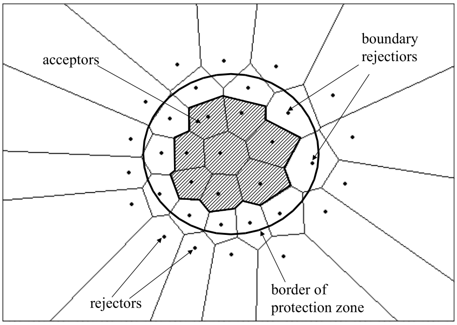


Fig. 4. Zone delineation with random verifier placement

T n o i v i i:

$$1 + \left\lceil \frac{\pi(r-k)}{k \quad \mathrm{n}(\beta/\ )} \right\rceil$$

Sin k n β on n, n o v i in $O(r)$. □

v v i l n i in oo o T o n o n i lly yi l n y n on . Fo non- ivi l ol ion $r-k$ o l no o i l wi i i on in oly on i yin on i ion o T o .

6 Logarithmic Verification Time

A o in o o ni ion l o o o o ol, ov ly o i i n l nil i o v i . T ov in i i n l n y x i . L d l i n w n ny wo oin in n on . Sin o n v i v o in i o ion on , xi n o o i $\lceil d/x \rceil$, i. . i i o o ion l o i o o ion on . How v , wi i l l yo o n o n o i ion o o o ol, i n n o o ion l o lo i o i o on .

In o o o i, w ollowin x ion on l n o o . i, $(i \geq 0)$ $x \cdot {}^{i+1}$ $x \cdot {}^{i}$ $x \cdot {}^{i+1}$

W l o o ni ion l ollow . x

v j ion l no n . H n , i y o o o ol i no ff . B low i o i o n o o ov n o .

Theorem 5.

Proof: T xi o i n o ov i d. T ov i o $i+$ o . T xi i n i n l o ov ov i $x \cdot {}^{i+1}$. T i $x \cdot {}^{i+1} \leq d$. T in lo i o o i , w $i \leq$ lo $(d/x) - 1$.

Sin x i on n , i i in $O($lo $d)$. T , n o o i o o ion l o lo i o o ion on i . □

7 Shrinking the Ambiguity Zone

T i i y on i w v y oin i lo o n o n o j o w iff n in iv i n i l n x. A ov in i i y on v o in o i o o ol i j v n o i i in i o ion on . In i ion, w x n o o ol o ov in i i y on i . T i , in ff , in i i y on . T x n ion i on i o nin i n l o ov o n o i w il no j o o. T ov in i i y on v o in o o ni ion l in S ion 3. I ov i j , i v ollow :

[illegible] z, [illegible] $z/$ [illegible] ([illegible]), [illegible] $z/$ [illegible]

ll no ion l on vio o li io ov . H n , i y o o o ol i no ff y ov o i-ion.

Theorem 6. *a* ([illegible] *b*) [illegible] ([illegible]) [illegible] O(lo $(b-a)$) [illegible]

Proof: v i o n o x o o no n i w only on i w ov in (n n v) i i n l n . S o i $i+1$ i ion o ov i j o i , n o i in j i ion l i ion . T l ion w n a n xi i n ov y ov ' i n l i ollow :

$$a < ix + \frac{x}{} + \frac{x}{2} + \cdots + \frac{x}{j} = ix + x\left(1 - \frac{1}{j}\right)$$

Sin ov i j o i i o in i i y on , i n o ov :

$$b < (i+1)x$$

A in in li y o on , i li yin n in lo i o o i w :

$$j < \text{lo} \; \frac{x}{b-a}$$

Sin x i on n , n o x o i o o ion l o lo - i o iff n w n b n a. □

8 Complex Signal Propagation

T i ion o on i l o ion o l w w iv wi in x i n o o ni ly o io i n l w il ny iv yon i x i n ni ly o no .

In i ion, w x n i n l o ion o l ollow . I ov n i n l, n (i) i i ni ly iv y v i i v i i no o n o x i n r w y o ov ; (ii) i y o y no iv y v i w o i n o ov i w n r n $r + y$ w y i o on n i n ; n (iii) i i no iv y v i o n $r + y$ w y o ov . A wi o i in l ion, r n on i n l n o ov . Di n y, ow v , i on n n in n n o i n l n .

T ollowin wo l iv l n o L 1 n . T oo i il .

Lemma 6. y

Lemma 7. $x + y$

T l i il o on in in o S ion 3 n on n ion l o ly o o l x i n l o ion o l.

9 Arbitrary Verifier Placement

on i ollowin v i n o v i ion o o ol. n in l i , - l l lo ion , v i o i ion i ily on l n . W v i v no nowl o i o i ion o i n ion o o ion on . v i i in o o w i i in i o o i o ion on (Fi). W ollowin o v i l n : i i non- y in ion w n v i ' Vo onoi ll n o i o ion on , n i v i i l o on o i Vo onoi n i o i o i o ion on .

T v i l i ollow :

- v i o i o ion on i j o ;
- v i Vo onoi n i o o i o ion on i l o j o ;
- o v i o .

Theorem 7.

Proof: A o in o l i ion l , o i v i j o . By ion, v i l v i i in i o ion on w o Vo onoi ll o ion on o Vo onoi n i o o i o ion on . A in, y l i ion l , v i i j o . T , nion o Vo onoi ll o j o ov o i o ion on . A o in o T o 1, o o ol o li wi i y o y o lo ion v i ion o l . □

In i ion o Vo onoi n i o n l ll y i i in v i wi o i n i y. Fo x l , wo o v i : i n j o (l l “ ”) n o n i l o (l l “ l ”). T v i n ly o i ion lon o o o ion on . T l v i o o o ion on . How v , n i y o l v i i l o i lo o o . To l n o n i o , v i o “ llo” on in i l l. T v i oxi o Vo onoi n i o y o io n i o . D o i n i y o v i o , l v i w o Vo onoi ll in o o o ion on v i io n i o . H n , i l v i o j o n ov ion i .

10 Practical Implementation Considerations

In in ion , w n lo ion v i ion o o ol n o i li y in ion o o l i y. In i ion, w i w y o l x ion o o o o ol n in o l i y y .

S o ni ion w n v i i vi l o o n ionin o o o o ol. I n o nno i n i o in j o , i nno n n o v i y o lo ion l i o ov . ion o ly o ni ion w n v i n l x y loyin on o ny o o ol v il l o . A oo o i v o ni ion i y in wi l n o n wo i i in [11]. TinyS [1] n TinyPK [13] wo i l i y y o wi l n o .

T li ili y o o ni ion i no jo ion in o o ol. W ov iv ll n o i y o n v i iv ll n y ov n on lv . In lo ion v i ion o o ol, v l in n w n o l lo . Fi , n w n v i y lo . T lo will no ff i y o o o ol v i x o no v i will no n il i v n lly iv . W i n i i no iv , v i o no i i ion, ov i no n i y o o o ol i no o o i . To n ov i v n lly , li l liv y o on n n o in o o in o o o ol. S on , o y ov o l lo o i o v i . T only n io o on n i w n o iv o lly j o o no . In i , ov y l ly . To o n i , j o v o l wi in i ni n n i in S ion . Ano vi l ol ion i o n l i l j o ov j ion on . Fo x l , v l in n n o v i ov in w ol l n n in o ion on . T ov i j w n l on o v i j i . v o o o ol o no in o o n o n i l l n y in o ni ion w n v i . T i , ow v , n n l y in o in o i w i - i n i o o n o i ion. To v o n , i n o o no o j o , ov i no .

Ano i no x li i ly in i i i i l n ion o o o ol. No i ow v , in o o o ol, o i i ion n o iv ov ’ i n l n o only o ni wi i Vo onoi n i o : i n o o ni wi j o o non o i n l, n wi o o i y iv i n l n i i j o i . H n , i l n ion o o o o ol o ili i n o ni ion w n o n i Vo onoi n i o . n w y o o i i o l i v i in o ni ion n o o .

v w ov io n l no o ov o n i lly w ol o ion on . How v , o o o ol n x n o o li i n ov . Fo x l o n l v y oin in n on i no w y o n o n ov ’ xi n .

Acknowledgments

W wo l li o x o i o l l i ion o Volo y y An iy v yy o K n S Univ i y, D vi W n o Univ i y o li o ni , B l y n Tin Y o No olin S Univ i y.

References

1. Sastry, N., Shankar, U., Wagner, D.: Secure verification of location claims. In: Proceedings of the ACM workshop on Wireless security, San Diego, CA (2003) 1–10
2. Naik, V., Arora, A., Bapat, S., Gouda, M.: Dependable systems: Whisper: Local secret maintenance in sensor networks. IEEE Distributed Systems Online **4** (2003)
3. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: Authentication in ad-hoc wireless networks. In: Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2002), San Diego, CA, Internet Society (2002)

4. Denning, D.E., MacDoran, P.F.: Location-based authentication: Grounding cyberspace for better security. In Denning, D.E., Denning, P.J., eds.: Internet Besieged: Countering Cyberspace Scofflaws. ACM Press / Addison-Wesley, New York (1998) 167–174 Reprint from Computer Fraud and Security, Elsevier Science, Ltd, February 1996.
5. Hu, Y.C., Perrig, A., Johnson, D.B.: Packet leashes: A defense against wormhole attacks. In: INFOCOM 2003. (2003)
6. Brands, S., Chaum, D.: Distance-bounding protocols (extended abstract). In Helleseth, T., ed.: Advances in Cryptology—EUROCRYPT 93. Volume 765 of Lecture Notes in Computer Science., Springer-Verlag, 1994 (1993) 344–359
7. Corner, M.D., Noble, B.D.: Zero-interaction authentication. In: Proceedings of the eighth Annual International Conference on Mobile Computing and Networking (MOBICOM-02), New York, ACM Press (2002) 1–11
8. Kindberg, T., Zhang, K.: Context authentication using constrained channels. Technical Report HPL-2001-84, Hewlett Packard Laboratories (2001)
9. Gabber, E., Wool, A.: How to prove where you are: Tracking the location of customer equipment. In: Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, California, ACM Press (1998) 142–149
10. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag, New York (1985)
11. Slijepcevic, S., Potkonjak, M., Tsiatsis, V., Zimbeck, S., Srivastava, M.B.: On communication security in wireless ad-hoc sensor networks. In: 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. (2002) 139–144
12. : Tinysec: Link layer encryption for tiny devices. (http://www.cs.berkeley.edu/nks/tinysec/)
13. : Tinypk project. (http://www.is.bbn.com/projects/lws-nest/)

Sentries and Sleepers in Sensor Networks

Mo . o [1], Yo n - i oi[1], n Ani A o [2]

[1] Department of Computer Sciences,
The University of Texas at Austin,
1 University Station C0500, Austin, Texas 78712-0233, USA
{gouda, yrchoi}@cs.utexas.edu

[2] Department of Computer Science and Engineering,
The Ohio State University, 2015 Neil Avenue,
Columbus, Ohio 43210-1277, USA
anish@cse.ohio-state.edu

Abstract. A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network can use their antennas to communicate in a wireless fashion by broadcasting messages over radio frequency to neighboring sensors in the same network. In order to lengthen the relatively short lifetime of sensor batteries, each sensor in a network can be replaced by a group of n sensors, for some $n \geq 2$. The group of n sensors act as one sensor, whose lifetime is about n times that of a regular sensor as follows. For a time period, only one sensor in the group, called *sentry*, stays awake and performs all the tasks assigned to the group, while the remaining sensors, called *sleepers*, go to sleep to save their batteries. At the beginning of the next time period, the sleepers wake up, then all the sensors in the group elect a new sentry for the next time period, and the cycle repeats. In this paper, we describe a practical protocol that can be used by a group of sensors to elect a new sentry at the beginning of each time period. Our protocol, unlike earlier protocols, is based on the assumption that the sensors in a group are perfectly identical (e.g. they do not have unique identifiers; rather each of them has the same group identifier). This feature makes our protocol resilient against any attack by an adversary sensor in the group that may lie about its own identity to be elected a sentry over and over, and keep the legitimate sensors in the group asleep for a long time.

Keywords: Energy management, Sentry election, Self-stabilization, Sensor Networks, Sentry-Sleeper protocol.

1 Introduction

A n o i y-o ll o wi n n nn n n in o n n n i , o n , , . S n o in n wo n i n nn o o ni in wi l ion y o in - ov io n y o n i o in n o in n wo . D

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 384–399, 2005.

o li i n o io n i ion, n o n wo lly l i-
o . S n o n wo n o ili y, nvi on n l o o i l
li ion in ion ion [1], i oni o in [ ] n i
oni o in [3].

n o ll n in o l in i nin n o n wo i o l n n
li i o n o i . n o o olv i o l i o x loi
i in o n ly loy n wo , ion o n o n o
o l o n i io , w il inin n o y w n
o i n in n wo . T l in n o v i n y
n l n n li i o i i , wi o i ni n ly in
o n o li ion nnin on n o n wo . x l o i
o n o n in [ ], [5], [6], [ ], [ ], [9], [10], [11].

In n , w n li i i o li l o ny, o i ly
ly o l , n o n wo : l n o in n wo y
o o n n o , o o $n \geq$. T o o n n o loy in
lo ion w in l n o wo l v n loy in n wo .
T i o o n n o on n o ollow . Fo i io , only
on n o in o , ll . . - , , y w n o ll
i n o o , w il inin n o , ll , , , o o l o
v i i . A innin o n x i io , l w
, n ll n o in o l n w n y o n x i io ,
n y l .

No n o in o i n i l in v y w y o o
n v in x ly nn in o in i n ,
w n i n o i l n y o o . T i i li no n o
n i n i i in i i o o n o in i o . , v y
n o in o o i n i .

T i n i o wo n o o in n wo , ow v , i -
in i l o w n n o iv , n o n in
w i w n o n o in i own o o i w n
o n o in iff n n y o . No n o in o n o
x n wi o n o in i o in o o l n w n y
innin o i io . A n o l o n o x n
wi n o in j n o in o o o i n , w n
i n o i l n y o i o .

An l n iv o o l n n li i o n o i in
ly o l n wo i o ovi n o wi l y w o
li i i n i li i o l y. How v , i l n iv
o i l li l n o o (w n o in ly
o l n wo i l y o o n n o) ollow . I n o il
in n wo , n n wo n o n o il n o ovi
n wo i i n in o o n l n iv o .

T o o ol y o o n n o o l n w n y
innin o i io i ll . . - - , , , - . . , . T o l o
n y- l o o ol wo- ol :

i. n in n no ll n o in n o o l in . T , in n l on n o in o i w n o n o i n o o .

ii. i io w wo o o n o in n o o w in o o w l o no i . (No i wo o o n o in n o o w in i io , n o o i n o o in io .)

n y- l o o ol o in [5], [11], [ ], [ ], [ ]. T in ion in i n o in “ n o o ” v i in- i l i n i i ; i. . y v iff n y i l lo ion , iff n onn - ivi y, iff n , o iff n i n i . T , n o in n wo i w i on y w on i n i o in n o on iff n i n i i , o y n no only v i i l o ovi o l v l o o n o li ion nnin on n - wo . Unli o o ol , o o o ol o l in n y innin o i io i on ion n o in o ly i n i l; i. . y v i n i l lo ion , onn ivi y, , n i n i . T i o o o ol l l n ili n in ny y n v y n o in o y li o i own i n i y (i. . li o i lo ion , onn ivi y, ...) o l n y ov n ov , n l i i n o l o lon i .

2 Sensor States and Transitions

B o w n x l in in o o n y- l o o ol, w n o x l in, in i ion, iff n o n o n n i ion w n .

v y n o in n o o n in ny on o wo : n i lin o l in . In i lin , n o o no in w i n il i i i o x i (in w i n o x i o ion), o i iv (in w i n o x ivin ion). An ion, w i o ion o ivin ion, o n o on i o n o n lo l v i l o n o , n o , o i o o n o o x i l i .

Al o n o n x i l n “ o- o- l ” n o n ion. I n o x i n , n o n i o i lin o l in . In l in , n o o no in w i n il i i o x i , n i x i o ion n n i o l in o i lin . Fi 1 ow wo o n o n iff n n i ion w n .

T wo in iff n w n i lin n l in o n o . Fi , in i lin , n o n iv n y o n o n x o on in ivin ion , w in

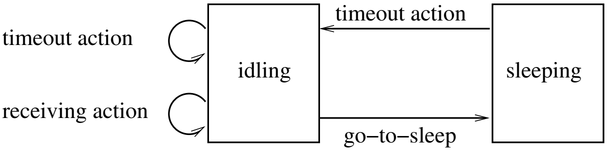


Fig. 1. Two states of a sensor

l in , n o nno o o in i n off i io w ll i o o n n in vi o v n y in i l . S on , on n y w n n o i in i lin i l n on n y w n n o i in l in (i in [1] n [13]). T o , o n o o v i n y o i l , i o l y in i l in lon o i l .

3 Sensor Network Execution

In i ion, w n o l o l o x ion o n o n wo . W i o l o i y o n y- l o o ol in n x ion. W l o i o l o v i y n n ly o o ol in S ion 5 n 6, n o v lo o i l ion in S ion .

T o n o n wo i i i ollowin wo on i ion . Fi , no in o olo y n i in n o in n o n wo . S on , i (u, v) o no u o no v in o olo y in i v y n y n o u n iv y n o v (ovi n i n o v no ny “n i o in n o ” o v n i w n n o u n i).

I o olo y o n o n wo i o n o u o n o v, n u i ll n o v n v i ll n o u. No n o n o n in-n i o n n o -n i o o no n o in n o n wo .

A n x l , Fi ow o olo y o n o n wo . T i n wo ix n o , n n o u in i n wo o -n i o , n ly n o v, v', n v''. T , i n o u n , n i n iv i l n o ly y n o v, n v', n v''. No n o u i o n in-n i o n n o -n i o o n o v' in i n wo .

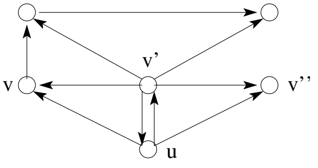


Fig. 2. Topology of a sensor network

W in x ion o n o n wo , l- i o i in n : in n 1, in n , in n 3, n o on. T i io w n on iv in n l. T iff n ivi i on- i x ion o n o n wo o only i in n , n no in i io w n in n . W o i io w n wo on iv in n t n $t+1$ $(t, t+1)$. (T v l o i ni i no i i l o o n n ion o n o n wo o l, w i v l o i ni i o n 100 illi on .)

A i in n , i -o o n o u y x i in u o x i i o ion. x in i o ion o n o u u o i own lo l v i l n o n o on . I y l o u o x n “ i o - < x ion>” w i i - o o u o x i (in) k i ni , w k i n v l o < x ion>. I y l o u o x n “ o- o- l ” w i u o l n il i -o o u x i . T i o ion o n o u i o ollowin o :

```
timeout-expires -> <update local variables of u>;
                   <send at most one message>;
                   <may execute timeout-after <expression>>;
                   <may execute go-to-sleep>
```

To o i i -o , n o u n i li i v i l n “ i . ”. In i ni w n wo on iv in n , i li i v i- l i . i i “ n ” o “no - n ”. Mo ov , i v i l i . i n in i ni , n i o i iv in v l in i ni . wi , i i no - n n no v l in i ni .

I n o u x n “ i o - < x ion>” in n t, n i . o n in i ni $(t, t+1)$ n i v l i v l o < x ion> in i i ni .

I i . i n n i v l i k, w $k > 1$, in i ni $(t-1, t)$, n i . on in o n n i v l i $k-1$ in i ni $(t, t+1)$.

I i . i n n i v l i 1 in i ni $(t-1, t)$, n n o u x i i o ion in n t n i . o no - n in i ni $(t, t+1)$, nl u x “ i o - < x ion>” o i i o ion.

I n o u x i i o ion n n in n t, n ny n o v, i n o -n i o o u, iv o y o in n t, ovi ollowin wo on i ion ol .

i. S n o v o no n ny in n t. (T i on i ion in i i n o v o no x i i o ion t, o i x i i o ion t i x ion o i o ion o no in l n in .)

ii. S n o v no in-n i o , o n n o u, n - in n t. (I v n t o i n in-n i o o

v, o n u, n t, n i i i o wi n y u t wi n l v iv no t.)

I n o u iv in n t, n u x i ivin ion t. x in ivin ion o n o u u o i own lo l v i l . I y l o u o x n “i o - < x ion>” w i i -o o u o x i k i ni , w k i v l o < x ion> in i ni $(t, t+1)$. I y l o u o x n “ o- o- l ” w i u o l n il i -o o u x i . T ivin ion o n o u i o ollowin o :

```
rcv <msg> -> <update local variables of u>;
             <may execute timeout-after <expression>>;
             <may execute go-to-sleep>
```

I ollow o ov i ion i in n , n o u x x ly on o ollowin :

i. u n on , iv no .
ii. u iv on , n no .
iii. u n no n iv no .

In n x ion, w i y o n y- l o o ol in o l o l o n o o o ol in i ion.

4 The Sentry-Sleeper Protocol

T o l o o n y- l o o ol i o o o n n o in l n o w o li i i $N * F$ i ni , w F i li i o l n o , n $1 < N < n$. T n n o in n o o on i n o n wo w o o olo y i lly- onn , i. . wo o o i - i ion w n v y wo no in o olo y.

D in i io , ll , $(n-1)$ n o o n o o in i l in n inin n o i in i i lin . In n, o l in n o i ll , n w n o i ll . A n o n, l w n ll n o in o l n w n y o n x n. T i y l o n ollow y n l ion o n w n y i ov n ov n il i o ll n o in o x .

A n o n, l w , n y lon wi n y oll o o l n w n y o n x n ollow . n o in o o n o io , ll , w o l n i o n ni o ly o n 1 .. $ravg-1$, w $ravg$ i v l n (in i ni) o ol ion io . T n, n o i i o o x i i ol ion io . T n o oo ll ol ion io in o i -o , n i n o l i l n w n y n n w n y n in o o :

sleep(gid, rt)

w gid i i n i o n o o n rt i inin i in n n. Ini i lly, inin i in n n i i n l n o n, w i i tl i ni .

W n n o u in n o o iv l (gid, rt) , n o u o ni n w n y i l o n n n i o l o rt i ni . T , i i i o o x i rt i ni , n o o l . T n o rt in iv l i 1 .. tl. T , o l in io i 1 i ni , n lon l in io i tl i ni .

A l n y n l (gid, rt) , n y o n o io w o l n rp i o n nio ly o n 1 .. $ravg-1$, n i i o o x i rp i ni . W n n y i -o , i n n x l (gid, $rt-rp$) . T n y on n in l , n il inin i in n n o o n ll l w o l n w n y o n x n.

No i n y io i lly n l v n w n ll o n o in o o ly l n nno iv ny . T i i in n o n l ollowin . So n o in o y no iv l (gid, rt) n y n y innin o n. T n o n iv l l (gid, rt') , w $rt' < rt$ n o o l o io o rt' i ni in i n.

In i o o ol, wo (o o) n o , y u n v, in o n l i n i l ol ion io n o y n i l i . T n ff i non o n o in o n iv ny l , in wo olli wi on no . nly n o u n v on i lv n i , n o n o in o o no o ni n w n y n l o n n. How v , o o o ol n on , only on , n o in o v n lly n l o in n t n ll o n o o o l t.

A o l i ion o n o u in o i ollow .

```
sensor u

const gid     : integer,     {group id of sensor u}
      tl      : integer,     {length of a turn}
      ravg    : integer      {avg length of random period}

var   sentry  : boolean,     {Is u sentry?}
      awake   : boolean,     {Is u awake?}
      rp      : 1..2*ravg-1, {length of random period}
      rt      : 0..tl,       {remaining time in current turn}
      g       : integer,     {group id in received message}
      t       : 1..tl        {remaining time in received message}
```

```
begin
   timeout-expires -> if !awake ->           awake := true;
                                             sentry := false;
                                             rp := random;
                                             timeout-after rp

                      [] awake and !sentry -> sentry := true;
                                             rt := tl;
                                             send sleep(gid, rt);
                                             rp := random;
                                             rp := min(rp, rt);
                                             rt := rt-rp;
                                             timeout-after rp

                      [] awake and sentry ->
                                if rt>0 -> send sleep(gid, rt);
                                             rp := random;
                                             rp := min(rp, rt);
                                             rt := rt-rp;
                                             timeout-after rp
                                [] rt=0 -> sentry := false;
                                             rp := random;
                                             timeout-after rp
                                fi
                      fi

[]  rcv sleep(g, t) -> if gid=g  -> sentry := false;
                                awake := false;
                                timeout-after t;
                                go-to-sleep
                      [] gid!=g -> skip
                      fi
end
```

I i i o n o no in i o o ol, n o in o o
o o n y ly on n o i ion wi o o in o ny
iff n in i i n i i y iv n v n o o n o ov
o in o . In n, n o in o o ili y
o o n y. T , n o n x o o n y on v y
n n o o. A n o u w o il o o n y o l iv ly lon io ,
y o $3 * n$ o $5 * n$ n , o l o n o in o
no ollowin o o ol. In i , n o u y i o y w (n
o i n o o) n o o o l .

5 Stabilization of the Protocol

In i ion, w oo o n y- l o o ol i l - ili in .
A . . o i o o ol i n y v l o v i l n i li i

v i l i . o n o *u* in o o ol. No o o o-ol o on o i ni w n wo on iv in n , in v l o ll v i l n ll i li i v i l o no n in ny i ni w n on iv in n .

W v y (w l i i o ill i i) o o-o ol i ollowin on i ion .

1. Fo v y n o *u*, i li i v i l i . i n n i v l i o *tl* i ni . (No i ion i in-in y x ion o o o ol.)
. Fo v y l in n o *u*, v l o i *awake* v i l i l . (No i ion i in in y x ion o o o ol.)
3. Fo v y w n o *u*, v l o i i . i i in o v l o i .v o ny o w n o *v*. (No i - ion i o ili i lly in in y oo in v l *ravg* o l l iv o n o n o in o .)

In o n y- l o o ol, [illegible] i n i ollowin [illegible]:

[illegible]

T o , in l i i , n o l i in n 0 .. $n-1$, n n o n i i in n 0 .. 1.

T o o ol i [illegible] iff i i ollowin wo on i ion [1].

i. [illegible] : S in o ny l i i , x ion o ny ion in ny n o in o o ol yi l l i i .
ii. [illegible] : S in o ny ill i i , o o ol i n o l i i .

Fi , w ow in o ny l i i , x ion o ny ion in ny n o in o o ol yi l l i i . T o o ol wo o on i . In , x ion i i o ion in n o *u* in o . In i , o i ili i o on i w n i o ion i x .

i. T v l o *awake* in *u* i l : In i , *u* on l *u* w o l in (y ion), n v l o *awake* . T , *u* o w , n o inv i n ol .
ii. T v l o *awake* in *u* i n v l o *sentry* in *u* i l : In i , *u* l i l n w n y n o n o in o l y n in l . No no o w n o n x i i o ion n o o n l i (y ion 3). T , *u* i w n o only n y in o , n o inv i n ol .

iii. T v l o *awake* in *u* i n v l o *sentry* in *u* i : In i , wo o on i n in on inin i in n n. Fi , i inin i i i n o, *u* n no l . T , *u* i ill w n i ill only n y in o . S on , i inin i i o, *u* o ni n n i ni , n wi w o n y y in i v l o *sentry* l . T , *u* i ill w , i no n y ny o . In o , inv i n ol .

In on , x ion i ivin ion in n o *u* in o . In i , wo o i ili i o on i w n ivin ion i x .

i. W n *u* iv l o no n o *v* in o : In i , *u* o ni n o *v* i l n y o n n, o *u* o o l o i l in io in iv . T , n o *v* i w n i only n y in o , n o inv i n ol .
ii. W n *u* iv l o n o in iff n o : In i , *u* i no n o no in . T , inv i n ol .

H n , in o ny l i i , x ion o i o ion o ivin ion in ny n o in o yi l o l i i .

N x , w ow in o ny ill i i , o o o ol i - n o l i i wi in ni x ion o ion in o . T wo viol inv i n ollow :

i. A w ll n o in o l in : In i , n o *u* in o i n o x i i o ion wi in *tl* i ni (y ion 1). By x in i o ion o *u*, *u* o w . T , l on n o in o will w wi in *tl* i ni , n n only on o w n o will v n lly o n y.
ii. A w wo o o n i xi in o : In i , only on n y w o i v l i ll , y n o *u*, i - o n n x i i o ion o n l o in n *t* (y ion 3). T o n i iv l o *u* n o o l *t*. T , ll n i x *u* o o l wi in ni x ion o ion .

T o , in o ny ill i i , x ion o i o - ion in o n o o o ol l i i wi in ni x ion o ion in o .

6 Protocol Analysis

o o ol, i in S ion , o o n n o on n o w o li i i $N * F$ i ni , w F i li i o l n o n N i o n i y, ll in o . l ly, w v $1 < N < n$. In i ion, w n ly o o ol n i v l o N.

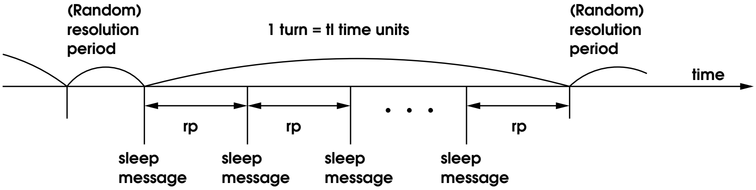


Fig. 3. A time period during protocol execution

Fi 3 ow i io T, on i in o ol ion io ollow y on n o tl i ni . Sin v l n o ol ion io i $ravg$ i ni , w v $T = tl + ravg$ i ni . D in n, n y n l v y n o io rp w o v i $ravg$ i ni . T o , v n o l n y n y n i $tl/ravg$.

L E_{slp} n E_{idl} o n o n y on y n o in l in n in i lin i ni iv ly, n E_{snd} n E_{rcv} o n o n y on y n o o n n o iv iv ly. T wo o i l n o in i io T:

i. : T n o in o o no x o o ol, n in in i i lin . T o n o n y on y n n o in i , E_{nop} i l l ollow .

$$E_{nop} = E_{idl} * (tl + ravg) * n$$

ii. : T n o in o x o o ol. T n y y in i lin n n $tl/ravg$ l o i i io . o $(n-1)$ l y in i lin o ol ion io , iv l , n l o tl i ni . T o , o n o n y on y n n o in i , E_p i l l ollow .

$$\begin{aligned} E_p = \quad & E_{idl} * (tl + ravg) + E_{snd} * (tl/ravg) \\ & +(n-1) * (E_{slp} * tl + E_{idl} * ravg + E_{rcv}) \end{aligned}$$

Table 1. Energy consumption of a sensor (in energy units)

E_{slp}	0.003 per time unit
E_{idl}	30 per time unit
E_{snd}	24.3 per message
E_{rcv}	9 per message

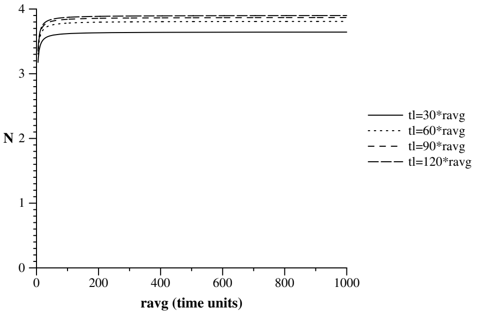


Fig. 4. N vs. $ravg$ when n=4

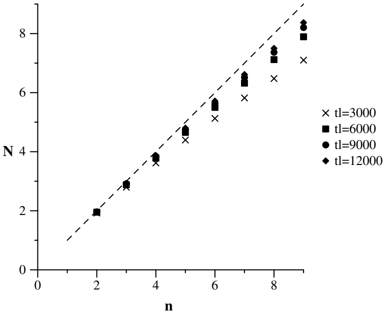


Fig. 5. N vs. n when $ravg$=100

F o ov n ly i , w n i ﬀ iv n o n o
N ollow :

$$N = \frac{E_{nop}}{E_p}$$

W n wo , Fi n 5, o ov o l o
o $tl = 30 * ravg$, $60 * ravg$, $90 * ravg$ n 1 0 $* ravg$ in i ni . In
o o , w v l in T l 1 o E_{slp}, E_{idl}, E_{snd} n E_{rcv}.
T v l o in n y on ion o l in [1], n
ion i ni i 100 illi on , n i io n o
n o o n o iv i 30 illi on .

Fi ow l ion i w n l n o $ravg$ n N w n n=
n $5 \leq ravg \leq 1000$ in i ni . I $ravg$ i 100 i ni o o , n
v l o N no lon n on $ravg$. Si il ly, w n n= n 9, i $ravg$ i 100
i ni o o , n v l o N no lon n on $ravg$.

Fi 5 ow l ion i w n n n N w n $ravg = 100$ i
ni . F o Fi . 5, on n wo o v ion . Fi , tl o no v
on ﬀ on v l o N, i lly w n n i ll. S on , N i lo
o n w n n i ll . D in ol ion io , ll n o in o
n o y w , n o o l o n o n y on y n o
in i io i in n i in . T , o o o ol o
o i n n i ll .

In l x ion o o o ol, n n y n n o o
y n i in i n. T n xi i io w no n o
in o i w o o i n o o . W ll i
i io .

W n i o l l n o ov li i o o o n n o o i l o l . W n n y i in i n, v i io no n o in o i w i $tl/$ (ini i io i o n xi i io i tl). Sin $(n-1)$ n o n i in i n y n , o l l n o ov li i i i ollow :

$$\frac{tl}{\ } * (n-1)$$

T o l l n o i l iv ly ll n li i o o . No o l l n o i l o n o n o in o , no li i o l n o .

7 Simulation Results

W v v lo i l o n i l x ion o o n y- l o o ol. T i i l o i l vio o o o n n o w o o olo y i lly onn n l o llow o on o o o ol tl n $ravg$.

Fo o o i l ion, w v o v l in T l 1 w ll ollowin v l :

- tl= 3000 i ni
- $ravg$= 100 i ni
- T o n o n y iv n o n o , innin o i - l ion, i no o n o in i i lin o 100000 i ni .

W n i l ion o i o o ol o n = , n 9, n lo l in Fi 6 n . Fi 6 ow ff iv n o n o N. i l n v ff iv n o

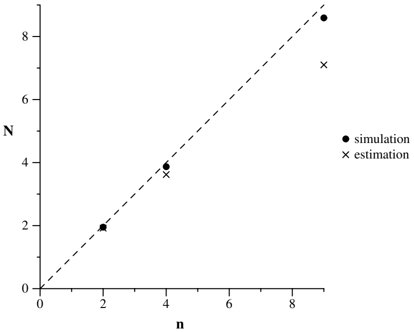


Fig. 6. The effective number of the sensors

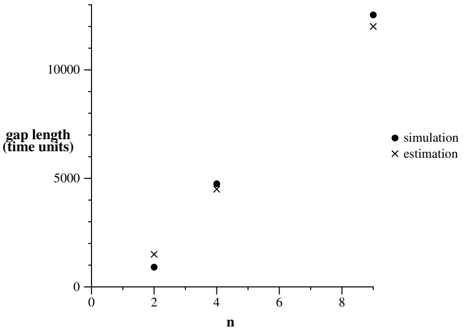


Fig. 7. The total length of gaps

n o ov 100 i l ion n X n i ff iv n o n o . T ff iv n o n o in i l ion i l n in i ion, in i l ion, n o n o o i i n i ov i n o n o n o in o ov i . A i in S ion 6, o o ol o o i n n i ll .

Fi ow o l l n o ov li i o o o n n o . i l n v o l l n o ov 100 i l ion n X n i o l l n o .

F o i l ion l , w ow ff iv n o n o N i lo o n o n o n in o . T i , o o n n o n l n n i li i o n n i li i o l n o y o in o o o ol.

8 Related Work

I i in SBPM[ ] o ivi n o in n wo in o wo , ni n l . S n i y w , n ovi i o ni ion vi n o n in vi , w il l o o l o v i n y. W n n i v n , y n w l o o n n in . How v , in SBPM, n i - l n x . Mo ov , n l o i w n l o o l . In AF[ ], ll n o iv l n in o in i n i in o i l lo ion in o ion. T n, only on n o in o o iv l n n o y w n i i in o in , w il o n o n off i io n o o l .

In S n[ ] n TMP [11], n o x n i n i o in o ion o o w i n o join onn on in n wo . nly n o in on i i in o in , w il o n o n o o l o v n y. In AS NT[5], n o in n wo o n o i iv n i o n lo , n join n wo o olo y only i n o o l l. How v , on n o n iv , i on in o w n il i i .

o o v n y in n o n wo v n o o in [15], [16], [13], [10], [6], [9]. In L A H[15], o o ni ion o , l - oll o n o in l , n n o n o w o ion. In ST M[10], n o in oni o in n off io. I n o n v n , i n on io n w o n o i n y.

L l ion o o ol v n i o in l - o in l - nn l io n wo in [1], [1], [19] n [0]. How v , in n l, o ool ion n in n i l n o ly li n [1], [1], [0]. T o o ol y no l in n o n wo , in n lly n o nno li n w il n o i n in in I 0 .11.

9 Concluding Remarks

In i , w i n y- l o o ol n y
o o n o o l n w n y innin o i io .
o o ol i on ion n o in o v i n i l
i n i i , n o n o o o o n y ly on n-
o i ion. W l o ow o o o ol i l - ili in . T i l ion
l ow o o n n o n l n n i li i 1.95 $*F$ o
$n =$, 3. $*F$ o $n =$ n .59 $*F$ o $n = 9$, w F i li i o
l n o .

o o ol n li o l - l ion l o i l n
n y lo v nly on no in l . By o in o o o ol,
no in l n l l - on n o i ion wi o
o in o ny i n i , o o no wi i i n i o
low i n i o no v n v n ov o .

Acknowledgment

T i wo w o in y D n A v n P oj
A n y (DA PA) on F33615-01- -1901, n in y IBM F -
l y P n i Aw o i y o 000- 003, n in y
T x A v n P o , T x Hi ion oo in in
Bo , n n TA P 1 -9 0 3.

References

1. Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Demirbas, M., Gouda, M., Choi, Y., et al: A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. Computer Networks (Elsevier), Special Issue on Military Communications Systems and Technologies **46** (2004) 605–634
2. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: A Survey. Computer Networks, Elsevier Science **38** (2002) 393–422
3. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson., J.: Wireless Sensor Networks for Habitat Monitoring. In: Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA (2002)
4. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In: Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom), Rome, Italy (2001) 85–96
5. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In: Proceedings of the Twenty First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002), New York, NY (2002)

6. Xu, Y., Heidemann, J., Estrin, D.: Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Research Report 527, USC/Information Sciences Institute (2000)
7. Hui, J., Ren, Z., Krogh, B.: Sentry-Based Power Management in Wireless Sensor Networks. The International Workshop on Information Processing in Sensor Networks (IPSN'03) (2003)
8. Xu, Y., Heidemann, J., Estrin, D.: Geography-informed Energy Conservation for Ad-hoc Routing. In: Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom), Rome, Italy (2001)
9. Younis, M., Youssef, M., Arisha, K.: Energy-aware Management for Cluster-based Sensor Networks. Computer Networks **43** (2003) 649–668
10. Schurgers, C., Tsiatsis, V., Ganeriwal, S., Srivastava, M.: Topology Management for Sensor Networks: Exploiting Latency and Density. In: Proceedings of The ACM Symposium on Mobile Adhoc Networking and Computing (MOBIHOC 2002), Lausanne, Switzerland (2002)
11. Bao, L., Garcia-Luna-Aceves, J.: Topology Management in Ad Hoc Networks. In: Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03), Annapolis, Maryland (2003)
12. Miller, M.J., Vaidya, N.H.: Minimizing Energy Consumption in Sensor Networks Using A Wakeup Radio. In: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'04). (2004)
13. Ye, W., Heidemann, J., Estrin, D.: An Energy-Efficient MAC protocol for Wireless Sensor Networks. In: Proceedings of the Twenty First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002), (New York, NY) 1567–1576
14. Arora, A., Gouda, M.: Closure and Convergence: A Foundation of Fault-Tolerant Computing. IEEE Transactions on Software Engineering **19** (1993) 1015–1027
15. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In: Proceedings of Hawaiian International Conference on Systems Science. (2000)
16. Singh, S., Woo, M., Raghavendra, C.S.: Power-aware Routing in Mobile Ad Hoc Networks. In: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom), Dallas, Texas, United States (1998) 181–190
17. Nakano, K., Olariu, S.: Randomized Leader Election Protocols in Radio Networks with No Collision Detection. International Symposium on Algorithms and Computation (2000) 362–373
18. Jurdzinski, T., Kutylowsk, M., Zatopianski, J.: Efficient Algorithms for Leader Election in Radio Networks. In: Proceedings of the twenty-first annual symposium on Principles of distributed computing, ACM Press (2002) 51–57
19. Jurdzinski, T., Kutylowsk, M., Zatopianski, J.: Weak Communication in Radio Networks. Euro-Par2002, Lecture Notes in Computer Science 2400, Springer-Verlag (2002) 965–972
20. Hayashi, T., Nakano, K., Olariu, S.: Randomized Initialization Protocols for Packet Radio Networks. International Parallel Processing Symposium (IPPS), IEEE (1999)

Clock Synchronization for Wireless Networks

i F n, In n l o y, n N n y Lyn

Massachusetts Institute of Technology, Cambridge MA 02139, USA
rfan@theory.csail.mit.edu, indranil@lcs.mit.edu, lynch@theory.csail.mit.edu

Abstract. Time synchronization is a fundamental service in many wireless applications. While the synchronization problem is well-studied in traditional wired networks, physical constraints of the wireless medium impose a unique set of challenges. We present a novel time synchronization algorithm which is highly energy efficient and failure/recovery-tolerant. Our algorithm allows nodes to synchronize to sources of real time such as GPS when such signals are available, but continues to synchronize nodes to each other, even in the absence of GPS. In addition, the algorithm satisfies a relaxed gradient property, in which the degree of synchronization between nodes varies as a linear function of their distance. Thus, nearby nodes are highly synchronized, which is desirable in many wireless applications.

1 Introduction

Wi l n wo n in in ly i o n i o i i oion. A wi l li ion ow o iv n o i i , i yn oni ion on wi l no o on i n o ny li ion . Fo x l , MA l y o o ol TDMA [5] i i yn oni ion o l olli ion- o l . Ti ynoni ion i n i l in n o n wo , w i oll o y i l nvi on n , n wi i o i o n . Ti yn oni ion i l o n in i -l v l li ion o i n o v n n i n l , n o i y o . W il i yn oni ion in wi n wo i w ll- i o l , wi l i n ni o ll n . T i y on n o ll wi l li ion i n y on v ion. A lo yn oni ion l o i (o o) lly i n i n y o yn oni ion, n voi floo in . In i ion, l oi nno y i lly ly on ow - n y o o l i PS. Ano i i o wi l n wo i n x n o i ly n n in n wo o olo y. T , SA in wi l i on in o n ion in o no il n ov i . L ly, ny li ion in wi l in lo l in n . T i , only n y no in n wo n o i i in o ivi y. T , i l o y o SA i i lo ly yn oni no w i n y, w il o i ly llowin w y no o o loo ly yn oni .

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 400–414, 2005.

In i wo , w n i yn oni ion l o i in
i n o wi l n wo . l o i i n y- i n : no -
o o on lo l (1- o) o yn oni ion o n . I ol
yn i n wo vio : l o i on in o n ion w n
i y no il n join . A nov l o o l o i i i -
o o , i. . yn oni in no wi o ,
n , i. . yn oni in no wi l i . T l-
o i llow in o o in PS o o l i w n i n l
v il l , on in o yn oni no wi o v n in -
n o PS. T , o x l , n o no n o l o i o o
i in wi l i , n o l TDMA o , w i
only i l iv i on no . l o i i l x
o y. In i l , l o i n lo w o wo
no w i i n *d* in n wo i o n y lin n ion o
d, l o ll i . Fin lly, o l o i i i l n y o i l n .
I i li l o y n o ion, n i i o o - o n
wi l no .

T in o i i o ni ollow . In S ion , w i
o l wo on lo yn oni ion. In S ion 3 n , w i
o o l o l n o l . W n o yn oni ion l o i in
S ion 5, n ow i i i o i in S ion 6. Fin lly,
w on l n i o wo in S ion .

2 Related Work

NTP [6] i wi ly loy i yn oni ion vi . NTP li on i-
y o i v , n oo v v o o
o o l i . In on , o l o i wo in n wo wi no in-
o . A o l i vi PS y xi , i in i n .
F o , nli NTP, w ol i ly yn i n wo , n no il-
n join o o wi o i in yn oni ion in
o n wo . S i n n To [ ] n n o i l lo yn oni -
ion l o i . How v , i l o i li on o , n i no i l
o wi l n wo . T i l o i o only in n l yn oni ion,
w il w in x n l n in n l yn oni ion. Al o, [ ] i o li
y n o ol By n in il . l o i only ol o in
il , i i l .

BS [1] i n i n SA i n o wi l n wo . BS o
yn oni ion, in w i no in i o n v n o
i i o . By on , o l o i o on- -fly yn-
oni ion, o w n i n v n o n i o . In
i ion, BS o only in n l yn oni ion.

i S y [ ] i SA o in o in n l n x n l yn o-
ni ion. i S y i v i y in i l n i y o
ion y ll no in lli /wi l n wo . How v , w nno -

n i l n i y in wi l n wo w on i n wo l i o . In i ion, i S y low l ol n n o lo i , n o no i v i n o y. F n i i n [3] l o in in n l n x n l yn oni ion. How v , i l o i i o o l x n o i l wi l y PS in o ion. In i , w in il nli ly o o .

F n n Lyn [ ] in o i n o y o lo yn oni ion. T y ow o v y SA, xi x ion in w i wo no i n d v lo w $\Omega(d+\frac{\log D}{\log\log D})$, w D i i o n wo . How v , i l i low o n on o in o v y no ’ lo i l lo . l o i i lo i l lo o on n o o io o i . T , i low o n o no i ly ly.

3 System Model

y o l on i o : yn i o , ov w i no n , n w i o ion lly in o no o l i . B low, w i ly.

3.1 Nodes

W wi o o l yn i y in w i no n il o join y i y i . To o i , w n N o o . no in N n in i o , n o no n n ny i . T o no w i i i in lo yn oni ion l o i o i on i o w no i . F il o no o l y no n in o w o l in . Join o no o l y o o i n i ion.

no i i wi w lo , w i w in o v i l w o v l n iff n i l n ion o i . D no v l o no i’ w lo i t y $H_i(t)$. W w lo o v y no . Mo i ly, w xi $\rho < 1$[1], o ll no i,

$$\forall t : 1 - \rho \leq \frac{dH_i(t)}{dt} \leq 1 + \rho$$

no i w lo n i iv o o no o o lo i l lo v l . D no v l o no i’ lo i l lo i t y $L_i(t)$. T lo yn oni ion l o i i o n lo i l lo v l o no lo o o , n lo o l i .

[1] In practice, ρ is very small, on the order of 10^{-5} or 10^{-6}.

3.2 Communication Network

No o ni wi o ov in n wo . In o wi l n wo , . ., o n wo , n wo n n in on o w no , w no on i l o o in on lv . o l y i i n wo o i onn y oo ny no il . How v , n wo o l o o l o lo yn oni ion. T , in i , w i li yin ion xi vi l o ni ion lin w n v y i o no $i, j \in N$. W lin i li l , FIF , n o n l y. in l ion, w o v y i, j, xi on n $d_{i,j} < \infty$, ll w n i n j, w i o n o n o i i o n y i o iv y j. Fo i li i y, w $d_{i,j} = d_{j,i}$[2]. Fin lly, w l o n wo $D =$ x$_{i,j}$ $d_{i,j}$.

3.3 GPS Service

W i in ll no i wi PS iv [3], n o- ion lly, PS vi n i in o in no o o l i . T i w n n i ion o no n on ol o no . W o l i o PS no i y n in ion $gps(t)_i$. T i i in n o in o i n l i i t. How v , y no , in n i y iv l i t. T i i i o i o PS o o o n i n wo . How v , w o n in y o v y PS , y in ll no w i w in i in v l $[t, t + D]$ iv $gps(t)$ no l n i $t + D$, w D i i o n wo .

4 The Clock Synchronization Problem

In i ion, w n in n l, x n l n i n yn oni ion o i lo yn oni ion l o i i y. I i no o- i l o i y o i ll i n ll no , w n no llow o il n join/ ov . T , ny i t, yn oni ion o i only i o ol o no w i l i t. W y no i i t i i iv l on PS in o t, n no il in ivin PS. L $S(t) \subseteq N$ no o no w i l i t. W y n x ion i i $S(t) = N$ o ll t. wi , w y x ion i .

[2] If $d_{i,j} \neq d_{j,i}$, we can simply redefine $d'_{i,j} = d'_{j,i} = \max(d_{i,j}, d_{j,i})$, then use $d'_{i,j}$ as the distance between i and j.

[3] Actually, it is enough for one node to have a GPS receiver, and for this node to propagate GPS messages to the rest of the network.

T i ion i n l wi in n l yn oni ion o no , i. ., o n in iff n in lo i l lo v l o ny wo no . L ϵ . Fo lly, w i l o i i y

Requirement 1 (ϵ-Precision). $\forall t \forall i, j \in S(t) : |L_i(t) - L_j(t)| \leq \epsilon$

T y i n l wi x n l yn oni ion o no , i. ., o n in iff n w n lo i l lo v l o ny no n l i . L ϵ . W i l o i i y

Requirement 2 (ϵ-Accuracy). $\forall t \forall i \in S(t) : |L_i(t) - t| \leq \epsilon$

T i n o y w in o in [ ]. I i ll i , iff n in lo i l lo v l o ny wo no w i i n d in o ni ion n wo (i. , no i, j, $d_{i,j} = d$) i o n y $f(d)$, w f i non in n ion o d. T i n o y i i l in li ion w lo o n y no w ll yn o-ni , w lo o w y no n o loo ly yn oni . An x l o n li ion i TDMA. In TDMA, only n y no n olli w n n i in , n only no n w ll yn oni lo o lin i n i ion . Pl [ ] o i ion l o iv ion n i ion o i n o y. yn oni ion l o i i w n o o i n o y, w i n o y ol only o i . Mo i ly, l $T \subseteq \mathbb{R}^{\geq 0}$ on i in o nion o non o-l n in v l . T n w i l o i i y i n o y o ll i in T. o , o o l i o xi i i o T, i. ., xi i $\frac{m(T)}{m(\mathbb{R}^{\geq 0})}$, w $m(\cdot)$ no L on $\mathbb{R}$. L α, β . Fo lly, w i

Requirement 3 ((T, α, β)-Gradient Precision). $\forall t \in T \, \forall i, j \in S(t) : |L_i(t) - L_j(t)| \leq \alpha d_{i,j} + \beta$

5 Algorithm

In i ion, w i o lo yn oni ion l o i . T o-o o l o i i w i n in TI A l n [ ], n i n in Fi 1. B low, w iv n ov vi w o ow l o i o . no in l o i in in wo lo , lo n lo . T lo l lo o no i n i' i o n l i . i' lo l lo n i' i o l lo l lo o ny o no . o ly in , i' lo i l lo i n o xi o i' lo l n lo l lo [4]. i' lo l lo i y o ion l PS in w i i iv . i' lo l lo i y io i in n l yn oni ion

[4] This definition is meant to convey intuition, and is not exactly correct; it is amended in the following paragraphs.

$ClockSync_i, i \in \mathbb{I}$

Constants

```
$0 \leq \rho < 1$
$0 < \tau$
```

State

```
$idle \in$ Boolean, initially $true$
for all $k \in \mathbb{N} : local[k] \in \mathbb{R}$, initially 0
for all $k \in \mathbb{N} : global[k] \in \mathbb{R}$, initially 0
$current \in \mathbb{N}$, initially 0
$next\_sync \in \mathbb{N}$, initially 0
$hardware \in \mathbb{R}$
$max\_gps \in \mathbb{R}$, initially 0
$do\_send \in$ Boolean, initially $false$
$send\_buffer$, a queue of elements of type $\mathbb{R} \times \mathbb{R}$, initially empty
```

Derived Variables

```
$mlocal \leftarrow \max_k local[k]$
$mglobal \leftarrow \max_k global[k]$
$logical \leftarrow \max(mlocal, mglobal)$
```

Transitions

```
input $\mathbf{wakeup}_i$
Effect:
    if $idle$ then
      $idle \leftarrow false$
      $current \leftarrow 1$

input $\mathbf{gps}(t)_i$
Effect:
    if $\neg idle$ then
      if $t > max\_gps$ then
        $max\_gps \leftarrow t$
        $current \leftarrow current + 1$
        $local[current] \leftarrow t$
        $global[current] \leftarrow t$
        $next\_sync \leftarrow \lfloor \frac{t}{\tau} \rfloor + 1$

input $\mathbf{recv}(t, s)_{j,i}$
Effect:
    if $\neg idle$ then
      if $s \geq max\_gps$ then
        if $t > global[current]$ then
          $global[current] \leftarrow t$
          enqueue $(t, s)$ in $send\_buffer$
          $do\_send \leftarrow true$
          if $\frac{t}{\tau} \geq next\_sync$ then
            $next\_sync \leftarrow \frac{t}{\tau} + 1$

input $\mathbf{crash}_i$
Effect:
    $idle \leftarrow true$
    for all $k \in \mathbb{N}$ do
      $local[k] \leftarrow 0$
      $global[k] \leftarrow 0$
    $current \leftarrow 0$
    $next\_sync \leftarrow 0$
    $max\_gps \leftarrow 0$
    $do\_send \leftarrow false$
    empty $send\_buffer$

output $\mathbf{sync}(t, s)_i$
Precondition:
    $\neg idle$
    $t = local[current]$
    $\frac{t}{\tau} = next\_sync$
    $s = max\_gps$
Effect:
    enqueue $(t, s)$ in $send\_buffer$
    $next\_sync \leftarrow next\_sync + 1$
    $do\_send \leftarrow true$

output $\mathbf{send}(t, s)_i$
Precondition:
    $\neg idle$
    $send\_buffer$ is not empty
    $(t, s) =$ head of $send\_buffer$
Effect:
    remove head of $send\_buffer$
    $do\_send \leftarrow false$
```

Trajectories

```
Satisfies
  unchanged:
    $idle, current, next\_sync, max\_gps, do\_send,$
    $send\_buffer$
    $1 - \rho \leq d(hardware) \leq 1 + \rho$

  $\forall k \in \mathbb{N}$ :
    if $\neg idle \wedge (k = current)$ then
      $d(local[k] - hardware) = 0$
      $d(global[k] - \frac{1-\rho}{1+\rho} hardware) = 0$
    else
      $d(local[k]) = 0$
      $d(global[k]) = 0$

Stops at
  $(\frac{local[current]}{\tau} = next\_send) \vee (do\_send =$
$true)$
```

Fig. 1. $ClockSync_i$ state and transitions w i no o . *i*' lo l lo in *i*' w lo . *i*' lo l lo in $\frac{1-\rho}{1+\rho}$ i *i*' w lo . T on o o in o *i*' lo l lo i o *i* o no ov i lo l lo o o no .

W n *i* iv PS i n l, i i lo l lo o v l o i n l. How v , o voi in *i*' lo i l lo w [5], *i* o i n lo l lo v l , n llo n w lo l lo ini i li o i in PS i n l. *i*' , *mlocal*, i o l o *i*' lo l lo , n *i*' o lo l lo v l . Mo ov , *i*' lo i l lo v l i n l o *i*' *mlocal*, n *i*' *mglobal*, w i will n o ly.

T w y n o lo l lo v l ivin PS i - lly i l n in o l o i i li ly iff n o w i ov , o i o n o i . In o l o i , *i* o n y *local* o lo l lo v l , n i n in x *current* in o *i*' n ly iv lo l lo . *i* in *local*[*current*] i w lo , *local*[*k*] on n , o ll $k \neq$ *current*. W n *i* iv PS in , *i* in *current*. T i ff o o in *i*' vio lo l lo n in n w on . T n w lo l lo i ini i li o v l o PS in . *mlocal* i n xi v l in *local*[$\cdot$]. In i ion, *i* o n y *global* o lo l lo v l , n y in i il w y o ow *i* i *local* lo v l i iv PS. *mglobal* i n xi v l in *global*[$\cdot$].

To in in in n l yn oni ion, no x i *sync* ion - oxi ly on v y τ i , w τ i on n . Mo i ly, no *i* o n in x *next_sync*, n w n *i* n *local*[*current*] $= \tau \cdot$ *next_sync*, *i* - o *sync*$_i$ ion. T n, *i* in n *next_sync*. T *sync* ion n o o o (*local*[*current*], *max_gps*), w *max_gps* i l PS v l *i* iv . *max_gps* " i " o ow *i*' *local*[*current*] i . T i , i *i*' v l o *max_gps*, o n ly *i* iv PS in , n o i *i*' v l *local*[*current*].

Now, on i w n *i* iv yn oni ion (t, s), w t i *local*[*current*] o o o no , y *j*, n s i *j*' v l o *max_gps*. *i* *j* iv l n PS v l *i*, n l o *j*' *local*[*current*] i n *i*' *global*[*current*], w i i *i*' n i- o l *local*[*current*] o ny o no . I o on i ion , n *i* o t in *global*[*current*], n o (t, s) o *i*' n i - o in n wo . L ly, i $t \geq \tau \cdot$ *next_sync*, n i no n o *i* o o *sync* w n *i* n i own *local*[*current*] $= \tau \cdot$ *next_sync*, in *j* l- y on *sync* wi i . In i , *i* *next_sync* o $\lfloor \frac{t}{\tau} \rfloor + 1$.

L ly, w i ow l o i l wi no il n join . I no il , i o no in wi yn oni ion on inin

[5] Many applications require logical clocks to be monotonic, in addition to being accurate and precise.

no . T , no in i n o l wi no il . I no join , n i ini i li i o o l v l , n w i o iv i PS in . T PS in ini i li n w no ’ o o o v l , w i no n i i no lly in l o i .

6 Analysis

In i ion, w ow l o i i in S ion 5 i i n i in S ion . W i no ion in oo .

6.1 Notation

L i no , l *var* v i l o i, n l t i . T n w l $i.var(t)$ v l o *var* i i t, ny i ion v o i t. W l $i.var(t^+)$ v l o *var* i i t, i ion v o i t. T , o x l , i $i.current = 1$ i 5, n i iv PS i 5 w i i o in n *current*, n w v $i.current(5) = 1$, n $i.current(5^+) =$.

A in S ion 5, no o *sync* ion oxi ly v y τ i . W l o PS vi no v y T i , o o on n T. T i , o $gps(t)$ o o no i t_1. T n $gps(t')$ o o no i t_2, w $t' > t$, n $t_1 \leq t_2 \leq t_1 + T$. iv n n ion $\xi = gps(t)$, w y t i o ξ. iv n n ion $\phi = recv(t, s)$, w y t i o ϕ.

I no i iv $gps(t)_i$ in , w y PS i i $t > max\_gps$, o i i o n i . Si il ly, i i iv $recv(t, s)_{j,i}$ in , w y *recv* i i $s \geq max\_gps$ n $t > global[current]$, o i i o n i .

6.2 Proof of Correctness

W ov l w i in il x ion, *mglobal* o ny no i n v o n xi *mlocal* o ll no . T i l i o ow l o i i ϵ- y, v n in il on x ion .

Lemma 1. α

$$\forall t \forall i \in N : \underset{i}{\ }\mathrm{x}\, i.mglobal(t) \leq \underset{j}{\ }\mathrm{x}\, j.mlocal(t) + (1-\rho)D$$

Proof. W in y ovin $\forall t :$ $\mathrm{x}_i\, i.global[current](t) \leq$ $\mathrm{x}_j\, j.mlocal(t) +$ $(1-\rho)D$, n ow i i li l . To ov o n , x n i n t, n on i l ϕ w i i iv o i t. S o ϕ w iv i t_2. T . i ϕ i PS, o i i *recv*.

In , w v $i.global[current](t_2^+) \leq i.local[current](t_2^+)$. Al o, in i iv no o l in $(t_2, t]$, w v

$$\begin{aligned} i.local[current](t) &\geq i.local[current](t_2^+) + (1-\rho)(t-t_2) \\ i.global[current](t) &\leq i.global[current](t_2^+) + \frac{1-\rho}{1+\rho}(1+\rho)(t-t_2) \\ &\leq i.local[current](t) \end{aligned}$$

T on in li y ollow i in $i.global[current]$ o $\frac{1-\rho}{1+\rho}$ i i w lo , w i i o $1+\rho$.

In on , w ϕ i $recv$, l j no w i n ϕ, n o ϕ w n i t_1. T n w v

$$i.global[current](t_2^+) \leq j.local[current](t_1) \tag{1}$$

on i l PS ξ j iv i t_1, n o j iv ξ i t_3. L i o ξ s_1.

$t - t_3 \leq D$

Proof. S o o on i ion $t - t_3 > D$. T n in ξ o D i o i, i v iv ξ y i t, y i t_4. on i wo . i $t_4 < t_2$, o $t_4 \geq t_2$.

In , l $s_2 = j.max\_gps(t_1)$. T n, in j o n ξ l i t_3, w v $s_1 > s_2$. Sin i iv ξ, w i i s_1, i $t_4 < t_2$, n $i.max\_gps(t_2) \geq s_1$. B i iv ϕ wi i s_2 i t_2, n i o n ϕ l, n o $s_2 \geq i.max\_gps(t_2) \geq s_1$, w i i on i ion.

In on , w l o on i ion, w n i iv ξ i t_4, i i n ξ l, o i o n o o i iv in i in v l $[t_2, t_4]$ l. In i , i on i - ion ϕ w l l i iv o i t. T , w v $t - t_3 \leq D$. □

Sin PS w i j iv i t_1 o $t_3 \geq t - D$, n j i no n $j.current$ n il l i t_3, n $j.local[current]$ in w i i l $1-\rho$ in i in v l $[t_1, t_3]$. T , w v

$$j.local[current](t) \geq j.local[current](t_1) + (1-\rho)(t_3 - t_1)$$

Al o, in i i no iv ny l in i in v l $(t_2, t]$, w v

$$\begin{aligned} i.global[current](t) &\leq i.global[current](t_2^+) + \frac{1-\rho}{1+\rho}(1+\rho)(t-t_2) \\ &\leq j.local[current](t_1) + (1-\rho)(t-t_1) \\ &\leq j.local[current](t) + (1-\rho)(t-t_3) \\ &\leq j.local[current](t) + (1-\rho)D \end{aligned}$$

W on in li y ollow o n. 1. T , w v own in ll , n o ll t, w v x$_i$ $i.global[current](t) \le$ x$_j$ $j.mlocal(t) +$ $(1-\rho)D$. Now, now l t_k^* k' i w n i in n $i.current$. T n, w v

$$\begin{aligned} i.mglobal(t) &= \underset{k}{\mathrm{x}}\, i.global[k](t_k^*) \\ &\le \underset{k}{\mathrm{x}} \underset{j}{\mathrm{x}}\, j.mlocal(t_k^*) \\ &\le \underset{j}{\mathrm{x}} \underset{k}{\mathrm{x}}\, j.mlocal(t_k^*) \\ &\le \underset{j}{\mathrm{x}}\, j.mlocal(t) \end{aligned}$$

T , w v own $\forall t:$ x$_i$ $i.mglobal(t) \le$ x$_j$ $j.mlocal(t) + (1-\rho)D$.

T n x l in ll x ion , in l in il on on , ny no ' lo i l lo v l i no n l i .

Lemma 2. $\forall t \forall i \in N:$ x$_i$ $i.logical(t) - t \le \rho(T+D) + (1-\rho)D$

Proof. Fix n i n t. Sin $i.logical(t) =$ x$(i.mlocal(t), i.mglobal(t))$, w ow $i.mlocal(t) - t \le \rho(T+D)$. on i l l PS ξ i iv o i t. S o ξ o i t_1, n i o ξ w s. W v $t - t_1 \le T + D$, PS o o w in n wo v y T i , n PS o D i o i. Now, $i.local[current](t_1^+) = s \le t_1$, n i iv no o PS in i in v l $(t_1, t]$, w v $i.local[current](t) \le i.local[current](t_1^+) + (1+\rho)(t-t_1)$. T , w v

$$\begin{aligned} i.local[current](t) - t &\le t_1 + (1+\rho)(t-t_1) - t \\ &= \rho(t-t_1) \\ &\le \rho(T+D) \end{aligned}$$

W v own o ll i n t, $i.local[current](t) - t \le \rho(T+D)$. Sin $i.mlocal(t) =$ x$_k$ $i.local[k]$, w v $i.mlocal(t) - t \le \rho(T+D)$, o ll i n t.

By L 1, w v in il x ion , $i.mglobal(t) \le$ x$_j$ $j.mlocal(t) + (1-\rho)D \le \rho(T+D) + (1-\rho)D$. Now, w o v i il in n x ion, n il nno $i.mglobal(t)$ o in . T , in il on x ion , w l o v $i.mglobal(t) \le \rho(T+D) + (1-\rho)D$. Fin lly, in $i.logical(t) =$ x$(i.mlocal(t), i.mglobal(t))$, w v $i.logical(t) \le \rho(T+D) + (1-\rho)D$, o ll i n t. □

T n x l ny l no ' lo i l lo v l i no l n l i .

Lemma 3. $\forall t \forall i \in S(t): t -$ in$_i$ $i.logical(t) \le D + \rho(T+D)$

Proof. Fix t n n $i \in S(t)$. Sin i i l i t, i iv PS o i t, n no il in PS. on i l

PS ξ i iv o i t, n o ξ o i t_1, n i s. T n $t_1 - i.local[current](t_1^+) = t_1 - s \leq D$. Al o, $i.local[current](t) \geq i.local[current](t_1^+) + (1-\rho)(t - t_1)$. Sin $t - t_1 \leq T + D$, w v $t - i.logical(t) \leq t - i.local(t) \leq D + \rho(T+D)$, o ny i n t. □

o inin L n 3, w ollowin .

Theorem 1 (Accuracy). $\forall t \forall i \in S(t) : |i.logical(t) - t| \leq D + \rho(T+D)$

F o T o 1, w i i ly ollowin .

Theorem 2 (Precision). $\forall t \forall i, j \in S(t) : |i.logical(t) - j.logical(t)| \leq (D + \rho(T+D))$

To v n y in i , PS vi i no in - n ly, o T n i l . Y v n in io wi o PS, no ill o in n l yn oni ion oxi ly on v y τ i . Sin τ y ll n T, w wo l li o n on i ion, in o τ in o T. Un o n ly, i no o n w i ol ll i . T on o i i PS in “in ili y” in n wo , ollow . on i w n no i iv PS i n l ξ. Sin i’ lo i l lo y iff o l i y o $O(\rho T)$, n in ξ i o j i lo i l lo o l i , n $i.logical$ y “j ” y $O(\rho T)$ i iv ξ. How v , in o no y no iv ξ i i, y i io w n i’ lo i l lo j o w , o no ’ lo i l lo v no . In i io , i ion i o n y $O(\rho T)$. n o n , w ow i PS wi in l D i , n i ion i o n y $O(\rho\tau)$. To ov i n , w ow i ol in il x ion in w i PS in o . T n w ow i ol in il x ion wi PS, n n lly, w ow i ol in il on x ion wi PS.

Lemma 4. $\forall t \forall i, j \in N : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$

Proof. Fix n i n j. L m l in $i.logical(t) \geq \tau m$. L t_2 $i.logical(t_2) = \tau m$. L t_1 li i $mlocal$ o ny no l τm. T i , $t_1 =$ in$_s$ $\exists k : k.mlocal(s) = \tau m$. L t_3 li i $mlocal$ o ny no l $\tau(m+1)$. In ollowin n ly i , i o $t_1 \leq t_2 \leq t_3 \leq t$. L $d_1 = t_2 - t_1$, $r = t_3 - t_2$, n $d_2 = t - t_3$. W v $d_1 \leq D$, no o τm n o $sync$, w i i iv no o n D i l . A i iv , w v $i.logical \geq \tau m$. Si il ly, $d_2 \leq D$.

W l i $r \geq \frac{\tau}{1+\rho} - d_1$. In , in no PS in , n xi o in o $mlocal$ o ny no i o $1+\rho$. Sin x$_k$ $k.mlocal(t_1) = \tau m$, x$_k$ $k.mlocal(t_3) = \tau(m+1)$, n $t_3 - t_1 = r + d_1$, w v $(1+\rho)(r + d_1) \geq \tau$, o w i l i ollow .

Now, in no PS in , o in o lo i l lo o ny no i o $1+\rho$, n l $\frac{1-\rho}{1+\rho}(1-\rho)$. T , w v

$$\begin{aligned}
j.logical(t) &\leq j.logical(t_3) + (1+\rho)(t-t_3) \\
&\leq \tau(m+1) + (1+\rho)d_2 \\
i.logical(t) &\geq i.logical(t_2) + \frac{(1-\rho)^2}{1+\rho}(t-t_2) \\
&\geq \tau m + \frac{(1-\rho)^2}{1+\rho}(r+d_2) \\
&\geq \tau m + \frac{(1-\rho)^2}{1+\rho}(\frac{\tau}{1+\rho} + d_2 - d_1)
\end{aligned}$$

Now, in w v $d_1, d_2 \leq D$, n in wo in li i ov , w

$$\begin{aligned}
j.logical(t) - i.logical(t) &\leq \frac{\rho}{(1+\rho)^2}\tau + \left(1+\rho - \frac{(1-\rho)^2}{1+\rho})\right) d_2 + \frac{(1-\rho)^2}{1+\rho} d_1 \\
&\leq \frac{\rho}{(1+\rho)^2}\tau + (1+\rho)D.
\end{aligned}$$

□

Lemma 5. t $[t-D,t]$ $\forall i,j \in N : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$

Proof. Fix n i, j n t. D n t_1, t_2, t_3 in oo o L . Now, in no PS o in i in v l $[t-D,t]$, n $t - t_3 \leq D$, n no PS o in i in v l $[t_3, t]$. on i wo . i no PS o in i in v l $[t_1, t_3]$, o o PS o . In , w n ov l in i il i in oo o L . Fo on , w on i i li v ion, in w i only on PS o in $[t_1, t_3]$. T n l wi l i l PS i n l i i il . L ξ PS in w i o , n o ξ i s, n ξ o j i t_j. Followin oo o L , w n ow $|i.logical(t_j) - j.logical(t_j)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$. Al o, in $t_3 - t_1 \leq \frac{\tau}{1+\rho}$, w v

$$\begin{aligned}
j.local[current](t) &\leq s + \frac{\tau}{1+\rho}(1+\rho) \\
i.local[current](t) &\geq s + (\frac{\tau}{1+\rho} - D)(1-\rho)
\end{aligned}$$

Now, in i only on PS in $[t_1, t_3]$, w v

$$j.logical(t) = \quad x(j.logical(t_j), j.local[current](t))$$

n

$$i.logical(t) = \quad x(i.logical(t_j), i.local[current](t))$$

T , w v

$$\begin{aligned} |j.logical(t) - i.logical(t)| \leq & \quad \text{x}(|j.logical(t_j) - |i.logical(t_j)|, \\ & |j.local[current](t) - i.local[current](t)|) \\ \leq & \quad \text{x}(\frac{\rho}{(1+\rho)^2}\tau + (1+\rho)D, \frac{\rho}{1+\rho}\tau + (1-\rho)D) \\ = & \frac{\rho}{(1+\rho)^2}\tau + (1+\rho)D \end{aligned}$$

T l li y ollow $\rho < 1$. T , w v ov n l in ll . □

Theorem 3 (Strong Precision). t $[t-D, t]$ $\forall i, j \in S(t) : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$

Proof. To ov i o , no i il v on i ion o yn oni ion. S on , i $i, j \in S(t)$, n y v iv l on PS. U in i , l ollow o v y i il i in oo o L 5. W o i oo o l o . □

L ly, w on i i n o y i n . on i wo no i n j w i i n d . J PS in "in ili y" w i i ion $O(\rho T)$ in o $O(\rho\tau)$, *sync* in ili y w i lo i l lo iff n w n i n j $O(\rho\tau + D)$, in o $O(\rho\tau + d)$, i y i n o y. How v , i n o y o ol i t, i no PS n no *sync* in in i in v l $[t - D, t]$. Mo i ly, w v ollowin .

Theorem 4 (Gradient Precision). t *sync* $[t - D, t]$ $i, j \in S(t)$ d $|i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)$

Proof. W ov i l v ion o o , w n x ion i il , n w n no PS in , o i ly o *sync* in . T ll o n ov in i il w y, n y ollowin i in oo o L 5 n T o 3. W o i ll oo o l o .

Fix n i, j n t. L m l in $i.logical(t) \geq \tau m$, n l t_2 $i.logical(t_2) = \tau m$. L t_1 $j.logical(t_1) = \tau m$, n l t_3 $j.logical(t_3) = \tau(m+1)$. L $d_1 = t_2 - t_1$, $r = t_3 - t_2$, n $d_2 = t - t_3$. In ollowin n ly i , i will o on i $t_1 \leq t_2 \leq t_3 \leq t$. By ion o o , j o no iv *sync* in i in v l $[t - d, t] \subseteq [t - D, t]$.

W l i j o no iv ny l *sync* in in $[t_1, t]$. In , o j iv l *sync* ϕ i $t' < t - d$. T n i o

ϕ l $\tau(m+1)$, in o wi j wo l no n ϕ l. Sin $t' < t - d$ n $d_{i,j} = d$, n i iv ϕ o i t. B n w wo l v $i.logical(t) \ge \tau(m+1)$, w i i on i ion. T , in j o no iv l *sync* in $[t_1, t]$, j' lo i l lo in o $1+\rho$ in $[t_1, t]$, n o w v

$$\begin{aligned} j.logical(t) &\le j.logical(t_3) + (1+\rho)(t-t_3) \\ &= \tau(m+1) + (1+\rho)d_2 \end{aligned}$$

Al o, i' lo i l lo in l $\frac{1-\rho}{1+\rho}(1-\rho)$, o w v

$$\begin{aligned} i.logical(t) &\ge i.logical(t_2) + \frac{(1-\rho)^2}{1+\rho}(t-t_2) \\ &= \tau m + \frac{(1-\rho)^2}{1+\rho}(r+d_2) \end{aligned}$$

Now, in j' lo i l lo in y τ o i t_1 o t_3, n j' lo i l lo w o $1+\rho$ in i i , w v $(1+\rho)(t_3-t_1) = (1+\rho)(r+d_1) \ge \tau$. T , $r \ge \frac{\tau}{1+\rho} - d_1$. Al o, w v $d_1, d_2 \le d$, *i.logical* τm (., $\tau(m+1)$) wi in d i *j.logical* τm (., $\tau(m+1)$). T , in wo in li i o ov , w

$$\begin{aligned} j.logical(t) - i.logical(t) &\le \frac{\rho}{(1+\rho)^2}\tau + \left(1+\rho-\frac{(1-\rho)^2}{1+\rho}\right)d_2 + \frac{(1-\rho)^2}{1+\rho}d_1 \\ &\le \frac{\rho}{(1+\rho)^2}\tau + (1+\rho)d \end{aligned}$$

□

Fin lly, w on i o ni ion o l xi y o l o i . W ow no o o ly on lo l (i. . 1- o) o τ i . By o i on, ny lo yn oni ion l o i i no o o o n i n wo in v y yn oni ion io , w i i no i l o n y- on vin wi l no .

Theorem 5 (Communication Complexity). *i* *i*
$\frac{\tau}{1+\rho}$

Proof. By loo in *sync* n *recv* ion in Fi 1, w no o only on lo l o o v l o *next_sync*. T v l o *next_sync* n only in w n *local*[*current*] o o no in y τ, n i l $\frac{\tau}{1+\rho}$ i . □

7 Conclusions

W v n n n y- i n n l - ol n lo yn oni ion l o i w i in in n l n x n l yn oni ion, n w i

i l x i n o y. l o i i i l , n ily i l-
n l on o - o n wi l no .

l o i n i yn oni ion on no w n n wo
i l , i. . in io w n PS o yn oni ion o ion no n ly
o . How v , w n n wo i n l , lo w y l .
T o in low o n xi on o i l i n o non- i n n
i n yn oni ion, low o n o no i i ly ly in o
in , w o no i low o n on o in o
lo i l lo . I i n in in o i l n i l ion w
i yn oni ion n in in ll i , y llowin lo i l lo
o in on n in n l io . Ano in in i ion o
i o i l n o l o i on l l wi l n wo ,
n o o i v vio wi wo o n ov n in
i .

References

1. Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.
2. Rui Fan and Nancy Lynch. Gradient clock syncrhonization, to appear. In *Proceedings of the Twenty-third Annual ACM PODC*. ACM Press, 2004.
3. C. Fetzer and F. Cristian. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 12(2):123–172, 1997.
4. Dilsun Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.
5. Errol L. Lloyd. *Broadcast scheduling for TDMA in wireless multihop networks*. John Wiley & Sons, Inc., 2002.
6. D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Computers*, 39(10):1482–1493, 1991.
7. T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J. ACM*, 34(3):626–645, 1987.
8. P. Verissimo, L. Rodrigues, and A. Casimiro. *Cesiumspray: a precise and accurate global time service for large-scale systems.* Technical Report NAV-TR-97-0001, Universidade de Lisboa, 1997.

Task Assignment Based on Prioritising Traffic Flows*

James Broberg, Zahir Tari, and Panlop Zeephongsekul

RMIT University, GPO Box 2467V, VIC 3001 Australia
{jbroberg, zahirt}@cs.rmit.edu.au, panlopz@rmit.edu.au

Abstract. We consider the issue of task assignment in a distributed system under heavy-tailed (ie. highly variable) workloads. A new adaptable approach called TAPTF (Task Assignment based on Prioritising Traffic Flows) is proposed, which improves performance under heavy-tailed workloads for certain classes of traffic. TAPTF controls the influx of tasks to each host, enables service differentiation through the use of dual queues and prevents large tasks from unduly delaying small tasks via task migration. Analytical results show that TAPTF performs significantly better than existing approaches, where task sizes are unknown and tasks are non-preemptive (run-to-completion). As system load increases, the scope and the magnitude of the performance gain expands, exhibiting improvements of more than six times in some cases.

Keywords: scheduling policies, task assignment, heavy-tailed workloads, load balancing, load sharing, supercomputing.

1 Introduction

The use of a group (or 'cluster') of commodity computers in place of individual and typically expensive servers is becoming more prevalent. Examples include supercomputing clusters (such as the Virginia Tech Terascale Cluster) and high profile websites such as Google and Amazon, among other applications. Such clusters are popular due to their scalable and cost effective nature.

Figure 1 illustrates one common cluster configuration. Tasks, or "jobs" arrive at the central dispatcher, and are dispatched to hosts according to a *task assignment policy*. When a task arrives at the dispatcher, it is placed in a queue, waiting to be serviced in first-come-first-served (FCFS) order. In this paper we assume tasks are not preemptible (that is, they cannot be interrupted), task sizes are not known *a priori* and no load information is available at the dispatcher. This is consistent with many batch and supercomputing facilities (such as those described in [1, 2]) where preemption is not supported due to the enormous memory requirements of tasks.

The choice of task assignment policy used has a significant effect on user perceived performance and server throughput. A poor policy could assign large tasks to overloaded servers, drastically reducing the performance of the distributed system. Therefore, the aim of a task assignment policy is to distributed tasks such that all avail-

* This project is fully supported by the ARC Discovery Grant no. DP0346545 awarded by the Australian Research Council (ARC) for 2003-2005 and Sun Microsystems.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 415–430, 2005.

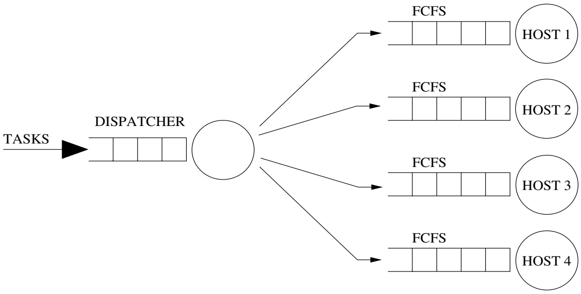


Fig. 1. Distributed Server Model

able system resources are utilised. Obviously it is undesirable to have one server in a distributed system overloaded while another server is sitting idle. However, the question of which assignment policy is the "best" still remains unanswered for many contexts.

Effective load distribution is especially crucial under realistic conditions of extremely heavy traffic demand and highly variable task sizes (i.e. the *workload*) that are commonly experienced in many computing environments [3,4,5,6]. Past research has shown that a heavy-tailed distribution is suitable for modeling these realistic workloads [3,5].

This paper proposes a new load distribution approach, called TAPTF (Task Assignment based on Prioritising Traffic Flows) which deals with certain inherent limitations of existing approaches in the same domain. TAPTF can improve performance under heavy-tailed workloads for certain classes of traffic by controlling the influx of tasks to each host depending on the variability of the workload. TAPTF also introduces multiple queues with hard processing time limits ('cutoffs') at each host. This enables service differentiation at each host, allowing small tasks to be executed quickly without being delayed by larger tasks. To achieve this, tasks that exceed the cutoff on a given host are migrated to the next host's restart queue (to be restarted from scratch).

TAPTF assumes no knowledge of the service requirements of incoming tasks. We are particularly interested in the areas that TAPTF can improve over TAGS, a policy that performs well when there is no preemption and task sizes are not known *a priori*. TAPTF is supported by a rigorous analytical model, based on fundamentals of queuing theory and priority queues.

The rest of this paper is organised as follows. Section 2 provides some background needed for the understanding of the concepts introduced in later sections. In Section 3 a survey of existing task assignment policies is presented. A detailed description of the TAPTF model is presented in Section 4. Section 5 gives an analytic comparison of TAPTF with existing approaches. Section 6 provides a detailed discussion of the analytical comparisons performed in Section 5. We conclude this paper with some closing thoughts on the usefulness of the TAPTF approach in Section 7.

2 Background

Many distributed computing environments exhibit a wide range of task sizes, typically spanning many orders of magnitude. These ‘heavy-tailed’ workloads have been found to exist in many computing environments. Crovella et al. observed that a number of file size distributions found on the World Wide Web exhibit heavy tails, including file requests by users, files transmitted via the network, transmission durations of files and files stored on servers [3, 7]. Other examples of heavy-tailed workloads observed include the size of files stored in Unix file systems [4], and the Unix process CPU requirements measured at UC Berkley [6]. Based on these measurements, workload generating tools such as SURGE [8] have been developed to more accurately ‘stress-test’ servers by generating realistic heavy-tailed traffic. More recently, traffic measurements of the 1998 World Cup [9] and the 1998 Winter Olympics [10] have exhibited heavy-tailed characteristics. The implications of these findings are significant in regards to task assignment policies, given that much of the existing work in the area was formulated under the assumption of an exponentially distributed workload. These so-called ‘heavy-tailed’ distributions have very high variance, where 10% of tasks can take 80% of the CPU resources.

For the purpose of analysis, we assume that the task sizes show some maximum (but large) value. This is a reasonable assumption in many cases, such as a Unix server which enforces a ‘CPU limit’ ceiling on how long a process can run. A *Bounded Pareto* distribution is therefore used, which exhibits the requisite heavy-tailed properties, and has a lower and upper limit on the task size distribution. The probability density function for the Bounded Pareto $B(k, p, \alpha)$ is:

$$f(x) = \frac{\alpha k^{\alpha}}{1 - (k/p)^{\alpha}} x^{-\alpha-1}, k \leq x \leq p \tag{1}$$

where α represents the task size variation, k is the smallest possible task, and p is the largest possible task. By varying the value of α we can observe distributions that exhibit moderate variability ($\alpha \approx$) to high variability ($\alpha \approx 1$). Typical measured values of the α parameter are between 0.9 - 1.3 [3, 5, 7], with an empirically measured mean of $\alpha \approx 1.1$.

In order to compare the relative performance of the various task assignment policies some common metrics are used. We consider the *mean waiting time*, *mean flow time*, and the *mean slow down* of each task assignment policy. The waiting time refers to the time a task spent waiting in queues to be processed. The flow time is the sum of the waiting time and the service time. Slow down refers to the waiting time divided by its processing time.

Consider for a moment that each host in our basic distributed system (depicted in Figure 1) is a M/G/1 FCFS queue, where the arrival process has rate λ. X represents the service time distribution, and ρ represents the utilisation ($\rho = \lambda E\{X\}$). W denotes a task’s expected waiting time in the queue, S its slowdown, and Q is the expected queue length on arrival. Then it follows that,

$$E\{W\} = \frac{\lambda E\{X^2\}}{(1-\rho)} (Pollaczek - Khinchinformula)$$

$$E\{S\} = E\{\frac{W}{X}\} = E\{W\} \cdot E\{X^{-1}\}$$
$$E\{Q\} = \lambda E\{W\}$$

3 Related Work

This section focuses on relevant existing approaches to task assignment, focusing on their strengths, limitations and performance characteristics with respect to dealing with conditions of heavy traffic demand and high task size variation. A more extensive review is available in [11].

Traditionally, classical task assignment policies such as *Random* and *Round-Robin* have been used in distributed systems. Under the Random policy, tasks are statically assigned to each member server with equal probability. Using a Round-Robin policy, tasks are assigned to member servers in a cyclical fashion. Both policies aim to equalise the expected number of tasks at each server, and are often used as a base line to compare with other task distribution policies. The performance of both policies are directly related to the variation of the task size distribution, and deteriorates as the task size variability increases, as tasks are assigned with no consideration of each host's load or the distribution of task sizes. Despite this, Random and Round-Robin are still commonly used in many scheduling environments (most likely due to ease of implementation). It has been shown previously [12] that Random and Round-Robin both have similar performance characteristics.

Dynamic policies aim to improve on classical static policies such as Random and Round-Robin by intelligently assigning tasks based on the current load at each host. The LLF (Least-Loaded-First) approach assigns tasks to the server with the least amount of work remaining, attempting to achieve instantaneous load balance. The work remaining can be approximated by the queue length (Shortest-Queue-First), or assuming the tasks service requirement is known *a priori*, the cumulative work remaining in the queue (Least-Work-Remaining). By keeping the load balanced, the waiting time in queue caused by high task size variation can be reduced. It is known that balancing the load minimises the mean response time [13, 14] in the type of distributed system that we consider in this paper. Despite this, the best performance is not always obtained by balancing the load, especially if you are interested in an alternative (and perhaps more important depending on your views) metric such as mean slowdown. Furthermore, truly balancing the load is a problem in itself given that the service requirement is often not known *a priori*. In such a case you are depending on an approximated measure of load (Shortest-Queue-First for example) to balance incoming tasks fairly amongst the backend hosts. Given the highly variable nature of the task size distribution (where the difference between 'small' and 'large' tasks can be enormous) it is easy to imagine how it is a bad policy to depend only on the *number* of tasks in the queue at each backend host, and the effect on performance that can result from using such information to base task assignment choices on.

A Central-Queue policy (where tasks are held at the dispatcher until a host is idle) has proved to be equivalent to a Least-Work-Remaining policy, illustrating that equivalent performance can be obtained without any prior knowledge of a task's size [1, 12].

While exhibiting similarly good performance under an exponential workload, the performance of a Central-Queue policy is equally poor under more realistic conditions of heavy-tailed workloads. Recently, a variation of the Central-Queue policy was considered - Cycle Stealing with Central Queue (CS-CQ) [2].CS-CQ holds tasks in a central queue at the dispatcher until a host is idle. CS-CQ denotes one host to service short tasks and another to server long tasks, but it can steal cycles from an idle host if available (and it is prudent to do so). The application of CS-CQ is limited to domains where *a priori* knowledge of a tasks size is known. Furthermore, Central-Queue policies require constant feedback between the dispatcher and the backend hosts to notify the dispatcher of an idle host.

Many size-based policies have been proposed to counteract the negative effects of heavy-tailed workloads. Approaches such as SITA-E [12], and EQUILOAD [15] partition the workload into size ranges, which are then mapped to backend hosts. These size ranges are be chosen to optimise various metrics, such as waiting time and slowdown. These policies assume that task sizes are known *a priori* (eg. at the dispatcher), which is not consistent with the model we are evaluating in this paper.

Task Assignment based on Guessing Size (TAGS) [1] is an approach that does not assume any prior knowledge of a tasks service requirement. Like SITA-V, TAGS is slightly counterintuitive in that it unbalances the load, and also considers the notion of 'fairness'. This refers to the desirable property that "... all jobs, long or short, should experience the same expected slowdown." [1]. The TAGS approach works by associating a processing time limit ('cutoff') with each host, so a task is run on a host up until the designated time limit associated with that host. If the task has not completed by this point, it is killed and restarted from scratch at the next host. These cutoffs are a function of of the distribution of task sizes and the outside arrival rate, and can be determined by observing the system for a period of time.

Under higher loads and less variable conditions, TAGS does not perform so well. TAGS gains much of its performance by exploiting the heavy-tailed property, by moving the majority of the load onto host 2, allowing the vast majority of small tasks to be processed quickly on host 1. TAGS also suffers under high loads due to excess - the extra work created by restarting many jobs from scratch. As pointed out in [1], "...overall excess increases with load because excess is proportional to λ (task arrival rate), which is in turn proportional the [overall system] load, ρ."

4 The Proposed Model - TAPTF

In this section we propose a new task assignment policy called TAPTF - Task Assignment based on Prioritising Traffic Flows - to address the limitations of existing approaches in dealing with certain classes of traffic.

4.1 Motivation

Harchol-Balter's TAGS approach [1], while seemingly counter-intuitive in many respects, proved to be a very effective task assignment policy for distributed systems. As such, TAGS provides an excellent point of comparison for any new task assignment policy operating under similar constraints. As described in Section 3, the TAGS policy has a number of desirable properties - the most one important being that it does not assume any prior knowledge of the service requirement of incoming tasks, while still maintaining good performance. The TAGS policy performs admirably under realistic highly variable conditions, exploiting the heavy-tailed nature that is consistent with many computing workloads. Despite this, TAGS can produce significant *excess* at the backend hosts - wasted processing that a task incurs (and the corresponding load placed on a host) when it has been placed in the incorrect queue and is subsequently restarted after exceeding the processing limit associated with a host. A task that is assigned incorrectly is penalized by being stopped, placed at the end of the next host's queue and restarted from scratch (upon reaching the front of that queue). These shortcomings are justified by the fact that, by the very nature of the heavy-tailed workload distribution, the tasks that are penalised can amortise the additional waiting and processing time *for the greater good*. Nonetheless, this is wasteful, but how can the efficiency be improved while still maintaining good performance? In response, TAPTF was formulated to address two keys areas:

- Reducing the variance of tasks that share the same queue.
- Reducing the penalty of wasted processing (*excess*) on the backend hosts - caused by tasks that do not complete their processing in time, and are restarted at another host ('handoffs').

4.2 Techniques

In Section 4.1 a number of shortcomings of the TAGS model were identified that needed to be addressed. As such, TAPTF was designed in order to improve on these key areas. The reasoning behind the techniques that TAPTF uses to address the shortcomings of existing approaches are briefly described in this section.

As illustrated by the Pollaczek-Khinchin formula in Section 2, it can be seen that all performance metrics are dependent on $E\{X^2\}$, the second moment of the task size distribution (ie. the variance) in a queue. We can infer that reducing the variance in the service requirements of tasks at each host can improve performance, reducing the chance of a smaller task being stuck behind a significantly longer task. TAPTF reduces the variance in the sizes of tasks that share the same queue by the use of dual queues (an Ordinary (O) queue and a Restart (R) queue) and task migration, in an effort to group like-sized tasks together.

The *excess* - extra work created by restarting many tasks from scratch - needs to be minimised. TAPTF attempts to reduce the amount of 'handoffs' by placing as many tasks in the most appropriate queue (that is, their final destination) in the first instance as possible - reducing the penalty on both hosts and tasks. This is achieved in two interrelated ways. First, by manipulating the fraction of tasks (q_i) that is dispatched to each host we can increase the number of tasks that are correctly assigned to a suitable host - that is, where they can run-to-completion. Secondly, the reason that tasks can enter the system (and potentially finish) at *any* host is due to the lower boundary cutoff of each Ordinary (O) queue being k, the smallest possible task size. Under TAGS, a task that needs to be processed at Host i (e.g. its size is between s_{i-1} and s_i) must migrate from Host 1 to Host i. In TAPTF for the same task, there is a probability q_i that it will be directly dispatched to Host i (an ideal choice), and a probability $q_i + q_{i+1} + ... + q_n$ that it be assigned to Host i or higher - where it will not be subjected to any handoffs. This practice becomes crucial as task size variation decreases.

4.3 Conceptual View of the TAPTF Model

As seen in Figure 2, tasks arrive at a central dispatcher, following a Poisson process with rate λ. The dispatcher assigns tasks (in a First-In-First-Out manner) to one of the n hosts (say, Host i, where $1 \leq i \leq n$) at random with probability q_i. Using a well known property of the Poisson process, we can infer that the arrival stream to host Host i is also a Poisson process with rate λq_i.

Due to the heavy-tailed characteristics of the task size distribution (as discussed in Section 2), we assume that the distribution of task sizes (that is, the service distribution) follows a bounded Pareto Distribution $B(k, p, \alpha)$ given by Equation (1). A 'cutoff' (s_i) is assigned to each host in the distributed system. Specifically, tasks are processed on hosts with the following conditions:

- Host i's O queue deals only with tasks whose sizes are in the range $[k, s_i], 1 \leq i \leq n$
- Host i's R queue deals only with tasks whose sizes are in the range $[s_{i-1}, s_i], , 1 < i \leq n$

where $k = s_0 < s_1 < s_2 < s_3 < \ldots < s_n = p$. These cutoffs can be computed in order to minimise certain measurable quantities such as mean waiting time or mean slowdown time. Further information on how the cutoffs are chosen is provided in Section 4.4.

Each host (excluding Host 1) provides two queues, an ordinary queue and a restart queue (denoted by O and R respectively). All tasks in the O and R queues are served on a First-Come-First-Served (FCFS) basis. Tasks sent to a given host from the dispatcher join that host's O queue. After a task has moved to the front of the queue it can begin to be processed. If the processing time of a task on a given host exceeds the assigned cutoff limit, the task is stopped, and moved to the restart (R) queue belonging to the next host. This process is repeated until these tasks run to completion at their final (correct)

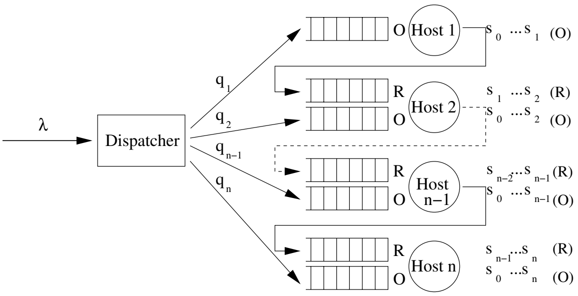


Fig. 2. Illustration of the TAPTF model

destination. Tasks waiting in a O queue have priority of service over those in the R queue at a given host. However, a task which is being served from the R queue will not be pre-empted from service by the arrival of a task into the O queue at a given host. This is the default behavior of the TAPTF (and is denoted as TAPTF-O in the figures in Section 5). It is worth noting that you could also choose to give priority of service to the R queue over the O queue (which we refer to as TAPTF-R).

One way the TAPTF model differs from TAGS is the fixed lower size boundaries at each host ($k = s_0$), so that all tasks with sizes *less than or equal* to a fixed cutoff point can be potentially be processed on a particular host. This means that a task can be dispatched to *any* host initially without being first dispatched to Host 1 (as per the TAGS approach) while preserving the property that a task's service demand is not known *a priori*. In addition, TAPTF uses dual queues at each host in order to speed up the flow of shorter tasks, allowing smaller tasks to be processed quickly in the ordinary queue and migrating larger tasks out of the way, allowing them to group together in the restart queues at subsequent hosts.

4.4 Choosing the Cutoffs

Like most size-based (or similar) policies, the performance of TAPTF is critically dependent on the choice of cutoffs used. From Section 3 we recall that cutoffs refer to the size-range associated with each host. The cutoffs can be chosen to optimise for mean waiting time, or mean slowdown. In order to optimise for mean waiting time, the load must be balanced more evenly amongst the host. To optimise for mean slowdown, load unbalancing techniques are employed, especially under conditions of high task size variation. We have chosen to optimise for both mean waiting time and more importantly, mean slowdown, as it is desirable for a tasks delay to be proportional to its service requirement.

The cutoffs for TAPTF are a function of the task size distribution (in our case defined by the Bounded Pareto $B(k, p, \alpha)$) and the task arrival rate into the distributed system, λ. These parameters can be determined by observing the distributed system for a period of time. Using the mathematical results described in [11], we can work towards obtaining optimal cutoff points (s_i's) for each of our hosts in the TAPTF system. Since our aim is to produce a task assignment policy that minimises the overall expected waiting time or slowdown respectively (depending on our goals), the following optimisation problems need to be addressed:

$$\text{Problem I} \;\; \text{Minimize} \sum_{i=1}^{n} E(W_{iO}) + \sum_{i=2}^{n} E(W_{iR})$$
$$\text{Subject to}\, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n.$$

$$\text{Problem II} \;\; \text{Minimize} \sum_{i=1}^{n} E(S_{iO}) + \sum_{i=2}^{n} E(S_{iR})$$
$$\text{Subject to}\, \rho_{iO} + \rho_{iR} < 1, 1 \leq i \leq n.$$

We can choose to optimise for mean waiting time (described by Problem I) or mean slowdown (described in Problem II).

As described above, the choice of cutoffs depend on the task size variability. From Section 2 we recall that the lower the α parameter, the higher the variability, and the smaller the percentage of tasks is that makes up 50% of the load. TAPTF (which can behave like TAGS by setting $q_1 = 1.0$ when prudent) can exploit this property of the heavy-tailed distribution by running all (or the vast majority) of the (small) tasks on the first host, leaving them under light to moderate load, while the largest tasks filter down to be eventually processed by the latter hosts.

As the variability decreases (α increases) we can no longer exploit the heavy-tailed property so easily. The average size of the tasks we consider 'small' slowly gets bigger as α increases. As such we have to choose our cutoffs accordingly, as well as manipulating the fraction of tasks that are assigned to the latter hosts. We still exploit the heavy-tailed property by processing larger jobs on the latter hosts, but we are not unbalancing the load to the extent we could when variability was higher ($\alpha \leq 1$). As α approaches 2.0, the task size variation is lower, and the other hosts have to start pulling their weight in order to maintain good mean waiting time and slowdown. TAPTF exploits this knowledge to provide better performance in those areas.

5 Analytical Comparison

In order to gauge the usefulness of the TAPTF approach, an analytical comparison with TAGS and Random was performed. Random is included as a baseline, whereas TAGS provides the best point of comparison as it operates under similar constraints (i.e. no *a priori* knowledge of a task's service requirement) to TAPTF. These approaches were evaluated under a variety of conditions and their performance compared using metrics discussed in Section 2 - mean waiting time and mean slowdown.

A range of α values were considered, from 0.5 to 2.0, demonstrating a wide range of task size variation, from extreme task size variation ($\alpha \approx 0.5$) to low task size variation ($\alpha \approx$ .0), and everything in between. Each α value was evaluated for different system loads (ρ) - 0.3 (low load), 0.5 (moderate load) and 0.7 (high load). For the sake of brevity the results for moderate load have been omitted and are available in the extended paper [11]. These comparisons were performed for two and three host systems, after which we could no longer find optimum s_i's with the computational resources available to us. This is not a big problem in itself as noted in [1], as an n Host distributed system (where $n >$) with a system load ρ can always be arranged in such a way to provide performance that is as good or better than the best performance of a two host system (where n is even). This holds true for any task assignment policy.

The analytical comparison was performed in Mathematica 5.0 [16], using the mathematical preliminaries discussed in [11]. The generalised TAPTF mathematical model is also used to model the behavior of TAGS by setting $q_1 = 1.0$ (and subsequently q_2 ... q_n to equal 0) - negating the dual queues and multiple entry points and making it behave identically to TAGS. For each scenario, optimum cutoffs are found with respect to mean waiting time and mean slowdown for both TAPTF and TAGS using the NMinimize function in Mathematica to produce the best (and fairest) comparison. This is achieved by finding the s_i's in each instance that produce local minimums for the expected waiting time, $E(W)$ and the expected mean slowdown, $E(S)$.

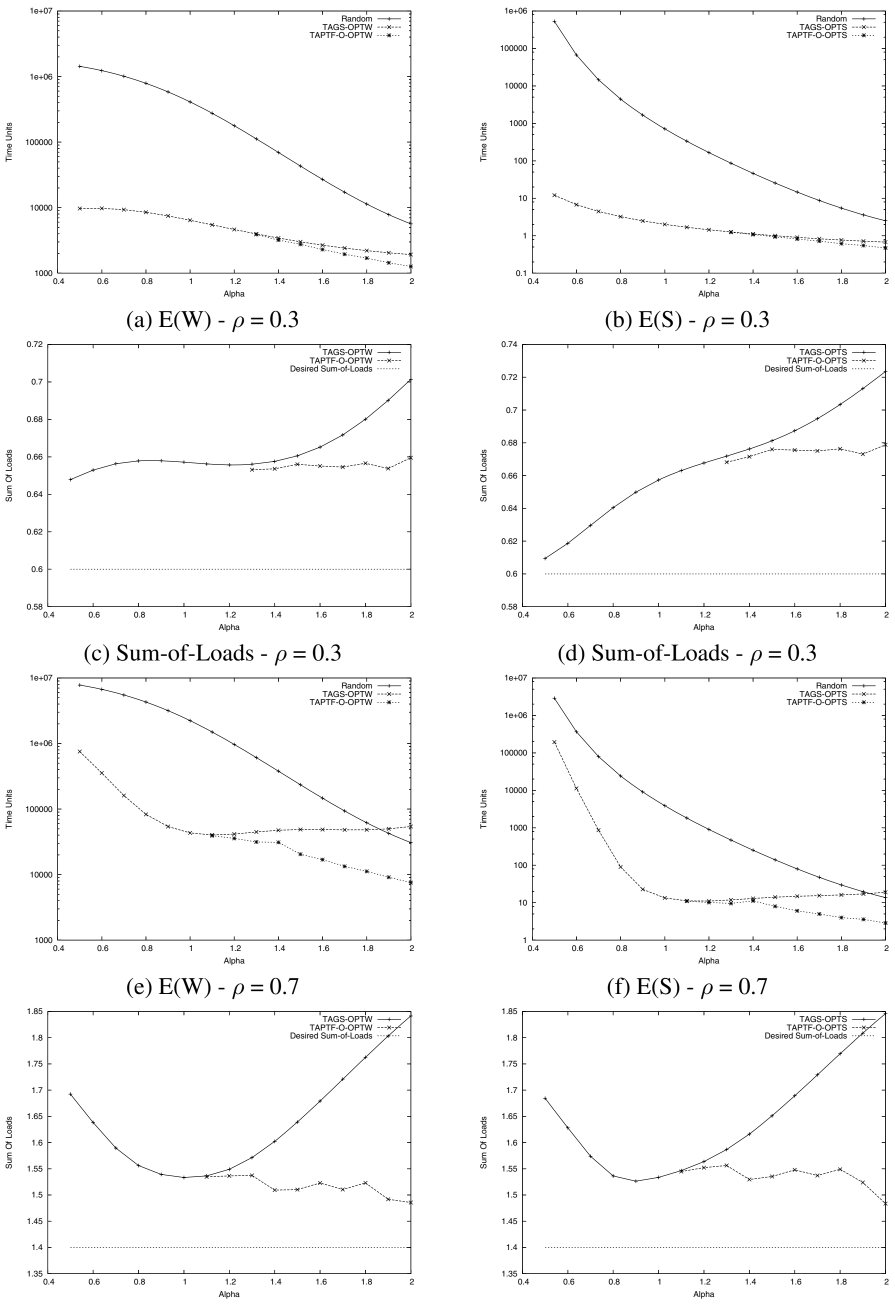
Task assignment policies that assume *a priori* knowledge of task sizes are not evaluated in this section, as we are motivated by a more pessimistic (and less restrictive) view of the distributed model, where this information is not guaranteed to be available.

In the interests of clear and meaningful results, comparisons of mean waiting time and mean slowdown are performed using the respective TAPTF and TAGS policies optimised for that metric, as described in Section 4.4. The Random policy is included as a baseline for comparative purposes in each instance. Note that the expected waiting time and slowdown graphs are presented on a log scale for the y-axis. Results for TAPTF are only shown where they are better than TAGS, as TAPTF can reduce to TAGS (and achieve identical performance) as described above.

5.1 Two Hosts

Figures 3(a) and 3(b) show the mean waiting time and slowdown respectively under a low system load ($\rho = 0.3$). From our analysis the TAGS policy achieves better mean waiting time and slowdown under conditions ranging from extreme to high variation (where α is between 0.5 and 1.2). The areas where the TAPTF policy improves on TAGS are highlighted on the graphs. It can be observed that in conditions of moderate to low variation (where α is between 1.3 and 2.0), the TAPTF policy can achieve better performance with respect to mean waiting time and slowdown. This performance increase can be attributed to the use of dual queues and by assigning tasks to all servers (or a subset thereof) rather than feeding all tasks into the first host, as per the TAGS approach. Table 1 gives a breakdown of the fraction of tasks dispatched to Host 1 (denoted by q_1) or Host 2 (denoted by q_2). From the table we can see that as variation increases (and α decreases) TAPTF approaches TAGS-like behaviors for optimal performance. We can see where α = 1.3, almost all tasks (99%) are dispatched to Host 1. As variation increases further (where α is between 0.5 and 1.2) TAGS-like behavior produces the best results. Conversely, when variation decreases it pays to assign some tasks to the second host. As the variation decreases (and α approaches 2.0) we can afford to assign more tasks to the second host. Figures 3(c) and 3(d) again highlight the effect of decreasing variance on TAGS - as α decreases, the amount of excess load generated by the TAGS policy increases significantly, while the TAPTF maintains consistent load. As the fraction assigned to Host 2 (q_2) increases, so to does the factor of improvement over TAGS, both in expected waiting time and slowdown in addition to system load.

Figures 3(e) and 3(f) show the mean waiting time and slowdown respectively under a high system load ($\rho = 0.\ $). The TAPTF policy betters TAGS over a larger range of task variation scenarios than occurred under low load (with TAPTF demonstrating lower mean waiting time and slowdown where α is between 1.1 and 2.0). It can be observed that TAGS suffers significantly under a high system load. As highlighted in Table 1 we are seeing an increased fraction of tasks dispatched to the second host in order to maintain superior performance to the TAGS policy. From Figures 3(g) and 3(h) we can observe a sharp increase in system load (and subsequently excess) where $\alpha > 1.0$. It can be seen that as α approaches 2.0 the factor of improvement over TAGS increases in all metrics.



(a) E(W) - $\rho = 0.3$

(b) E(S) - $\rho = 0.3$

(c) Sum-of-Loads - $\rho = 0.3$

(d) Sum-of-Loads - $\rho = 0.3$

(e) E(W) - $\rho = 0.7$

(f) E(S) - $\rho = 0.7$

(g) Sum-of-Loads

(h) Sum-of-Loads

Fig. 3. Performance of a two host distributed system under low and high load. Expected waiting time $E(W)$, slowdown $E(S)$ and corresponding system load comparisons (desired versus actual Sum-Of-Loads) are shown

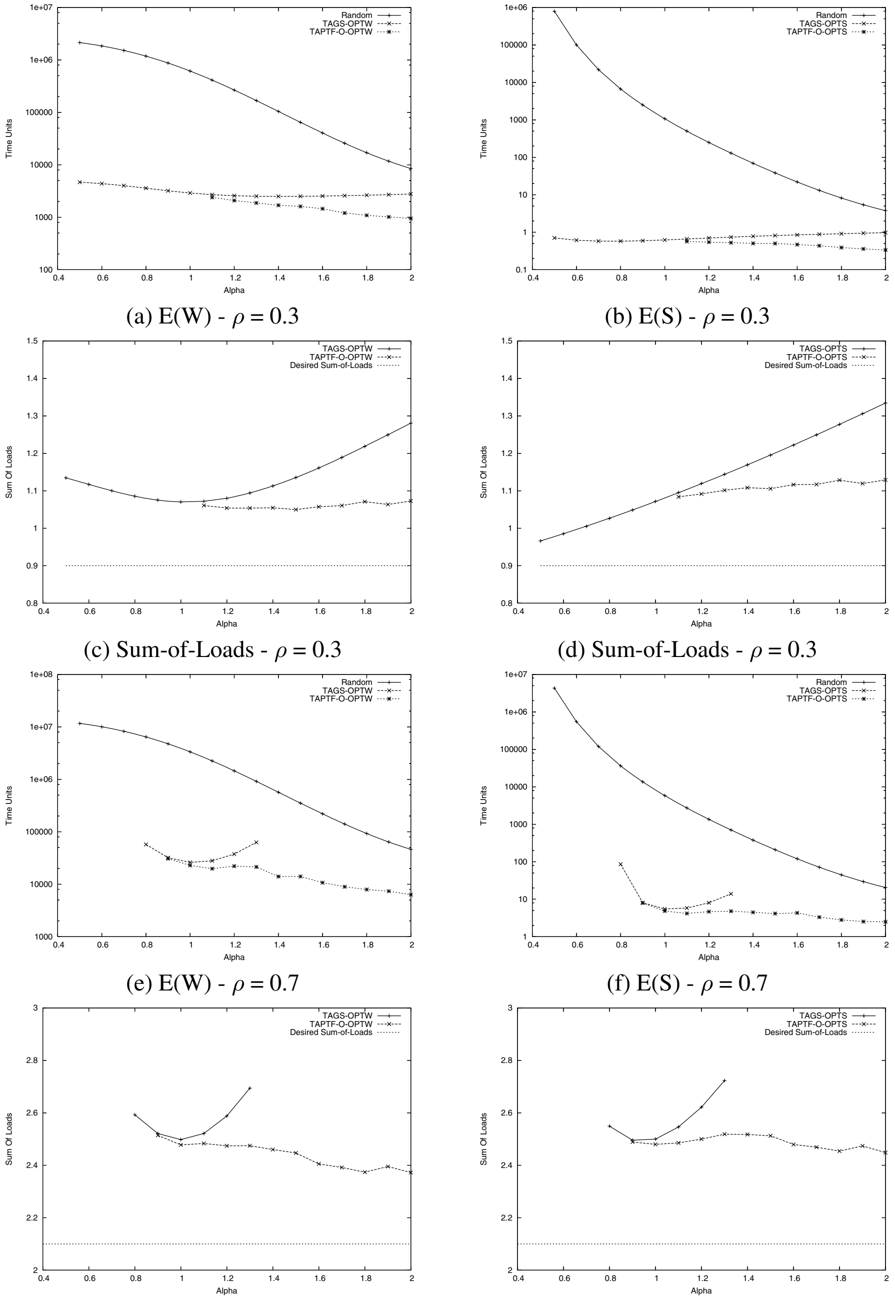
Table 1. Distribution of tasks in TAPTF - 2 Hosts

	$\rho = 0.3$		$\rho = 0.5$		$\rho = 0.7$	
α	q_1	q_2	q_1	q_2	q_1	q_2
1.1	1.00	0.00	0.99	0.01	0.99	0.01
1.2	1.00	0.00	0.99	0.01	0.95	0.05
1.3	0.99	0.01	0.95	0.05	0.90	0.10
1.4	0.95	0.05	0.95	0.05	0.80	0.20
1.5	0.95	0.05	0.90	0.10	0.75	0.25
1.6	0.90	0.10	0.80	0.20	0.75	0.25
1.7	0.85	0.15	0.80	0.20	0.70	0.30
1.8	0.80	0.20	0.75	0.25	0.70	0.30
1.9	0.75	0.25	0.70	0.30	0.70	0.30
2.0	0.75	0.25	0.66	0.33	0.60	0.40

5.2 Three Hosts

Figures 4(a) and 4(b) show the mean waiting time and slowdown respectively under a low system load ($\rho = 0.3$). It can be observed from the graphs that TAPTF performs better over a large range of α values, showing improved performance with respect to mean waiting time and slowdown where α is between 1.1 and 2.0. Table 2 gives an indication of how TAPTF distributed the load more intelligently as the task size variation decreases. As the variation decreases a significant amount of tasks are dispatched to the second host (denoted by q_2), and as α approaches 2.0 we can see more tasks being dispatched to the third and final host (denoted by q_3). The final host in a TAGS system typically processes only the largest tasks - as variation decreases this practice is shown to be poor, as demonstrated by TAPTF's superior performance. Figures 4(c) and 4(d) highlight the benefit of the TAPTF approach under high to low variation (where α is between 1.1 and 2.0) showing consistent system loads while TAGS exhibits a sharp increase. As α approaches 2.0, the TAGS policy is producing significant excess load, which is a worrying sign under such a low arrival rate into the distributed system.

Under a high system load ($\rho = 0.$), the mean waiting time and slowdown are depicted in Figures 4(e) and 4(f). Similar difficulty to those experienced under a system load of 0.5 occurred in finding cutoffs for many α values under the three host, $\rho = 0.$ scenario for the TAGS policy. That is, it was impossible to find optimum cutoffs that satisfied the requirement that the load must be below 1.0 at all hosts. This is confirmed when looking at the corresponding Sum-Of-Loads measurements shown in Figures 4(g) and 4(h), showing the Sum-Of-Loads approaching 3.0 (indicating that some or all of the hosts were approaching overload) where α is less than 0.8 or greater than 1.3. Table 2 shows the fraction of tasks (q_i) allocated to each backend server. We can see to handle the increased system load, a larger proportion of tasks are being assigned to the second and third host on average to cope. Indeed, when α is 2.0, each backend host is allocated a fairly equal share of the incoming tasks (where $q_1 = 0.$, $q_2 = 0.3$ and $q_3 = 0.3$). Again it can be observed that, as the system load has increased, the range of α values where TAPTF outperforms TAGS is still similarly large - where α is between 0.9 and 2.0.



(a) E(W) - $\rho = 0.3$

(b) E(S) - $\rho = 0.3$

(c) Sum-of-Loads - $\rho = 0.3$

(d) Sum-of-Loads - $\rho = 0.3$

(e) E(W) - $\rho = 0.7$

(f) E(S) - $\rho = 0.7$

(g) Sum-of-Loads - $\rho = 0.7$

(h) Sum-of-Loads - $\rho = 0.7$

Fig. 4. Performance of a three host distributed system under low and high load. Expected waiting time $E(W)$, slowdown $E(S)$ and corresponding system load comparisons (desired versus actual Sum-Of-Loads) are shown

Table 2. Distribution of tasks in TAPTF - 3 Hosts

	$\rho = 0.3$			$\rho = 0.5$			$\rho = 0.7$		
α	q_1	q_2	q_3	q_1	q_2	q_3	q_1	q_2	q_3
0.9	1.00	0.00	0.0	0.95	0.05	0.0	0.95	0.05	0.0
1.0	1.00	0.00	0.0	0.90	0.10	0.0	0.90	0.10	0.0
1.1	0.90	0.10	0.0	0.80	0.20	0.0	0.80	0.20	0.0
1.2	0.80	0.20	0.0	0.75	0.25	0.0	0.80	0.20	0.0
1.3	0.75	0.25	0.0	0.70	0.30	0.0	0.70	0.30	0.0
1.4	0.70	0.30	0.0	0.70	0.30	0.0	0.60	0.30	0.1
1.5	0.60	0.40	0.0	0.60	0.40	0.0	0.60	0.30	0.1
1.6	0.60	0.40	0.0	0.60	0.30	0.1	0.50	0.30	0.2
1.7	0.60	0.30	0.1	0.50	0.40	0.1	0.50	0.30	0.2
1.8	0.60	0.30	0.1	0.50	0.40	0.1	0.50	0.30	0.2
1.9	0.50	0.40	0.1	0.50	0.30	0.2	0.50	0.30	0.2
2.0	0.50	0.40	0.1	0.50	0.30	0.2	0.40	0.30	0.3

6 Discussion

An analytical representation of the Random load distribution policy was included as a baseline for comparison against TAGS and TAPTF. As discussed in previous work by Mor Harchol-Balter [1] and illustrated by the Pollaczek-Khinchin formula shown in Section 2, all performance metrics for the Random policy are directly proportional to the variance of the task size distribution. As such, as the task size variation increases, and α decreases, the expected mean waiting time and slowdown explode exponentially in all the scenarios examined.

From the figures presented in Section 5.1 and Section 5.2, it is clear that TAGS (or at least TAGS-like behavior) is the best policy under conditions of extreme to very high variation. As mentioned previously, TAPTF is an adaptable task assignment policy, which can behave identically (and reduces analytically) to TAGS (eg. set q_1 to 1.0) when it is prudent with regards to obtaining the best performance for a given scenario. In effect, the TAPTF policy encompasses TAGS ability to exploit a highly variable task size distribution, as well as remaining flexible enough to handle instances of lower variation and higher system loads by virtue of its many parameters that can be manipulated where required.

In areas of lower variation (and even low system load) we can see the benefit of dispatching tasks to hosts other than the first (highlighted by Table 1, Figures 3(a) and 3(b)). It is clear that as variation decreases, it pays to dispatch a growing proportion of tasks to the second host. This is largely due to the fact that we can no longer exploit the heavy-tailed property of the task size distribution, as the variation between the sizes of tasks decreases, and the average size of so-called *small* tasks increases.

TAGS suffers to a greater extent under higher loads, as an increase in excess (wasted processing caused by handoffs) and growing average queue lengths combine to have a detrimental effect on performance under conditions of moderate to low task size variation. It can be observed that as the system load increases, the task variation range where the TAPTF policy betters TAGS becomes larger, and the factor of that improvement (in both mean waiting time and slowdown) increases. For example, consider the two host case. Consider the results shown in Figures 3(a) and 3(e), depicting the mean waiting time under system loads of 0.3 and 0.7 respectively. TAPTF betters TAGS when $\alpha \geq 1.3$ under a low system load of 0.3. When the system load is high (0.7), TAPTF exhibits superior performance than TAGS when $\alpha > 1.0$. Similarly, consider when $\alpha =$ .0 in each of these scenarios. Under a system load of 0.3, TAPTF exhibits an factor of improvement of approximately 1.5 over TAGS. When the system load is 0.7, TAPTF shows a substantial improvement over TAGS - by a factor of 6.6.

Section 5.2 presents some interesting results for the 3 host scenario. We find that in some cases, as variation increases (and α decreases), the mean slowdown for the TAGS policy actually improves - to a certain point. Consider Figures 4(a) and 4(b), depicting a two host system under a low system load of 0.3. We observe a fairly flat and consistent response from the TAGS policy for the expected mean waiting time and slowdown over the range of α's shown. Slowdown gradually decreases as α approaches 0.7, then increases slightly as α reaches 0.5. This is because as the variation of tasks sizes becomes larger, TAGS can increasingly exploit the heavy-tailed property of such a distribution through choosing effective cutoffs that enable small tasks to be processed quickly, while ensuring large tasks are moved to latter hosts and do not unduly delay smaller tasks. This ensures good results with regards to overall metrics like mean waiting time and slowdown under conditions of extreme to highly variable task size distributions.

Despite the different behavior exhibited for the 3 host scenario, TAGS is still bettered by the TAPTF policy under conditions ranging from high to low task size variation due to the same factors as under the 2 host scenario. Again we see the benefits achieved by dispatching a proportion of tasks to all hosts, not just the first. This is especially true as the system load increases - so to does the factor of improvement of TAPTF over TAGS. The advantages of the generic and flexible TAPTF model are highlighted in Table 2 (and subsequently Figures 4(a) to 4(h)). In several instances (Figures 4(e) to 4(h)) we were unable to find optimum cutoffs for TAGS that satisfied the constraint that the load must remain below 1 at all hosts.

7 Conclusion

In this paper we have presented a new approach to task assignment in a distributed system, TAPTF (Task Assignment based on Prioritising Traffic Flows). TAPTF is a flexible policy that addresses the shortcomings of existing approaches (outlined earlier in this paper) to task assignment. TAPTF demonstrated improved performance (both in mean waiting time and mean slowdown) in key areas where the TAGS and Random policies suffer. Most significantly, TAPTF exhibited improved performance under low to high task size variation and high system load by reducing the excess associated with a large number of restarts and by intelligently controlling the influx of tasks to each back-end host. We found for two and three host scenarios that as system load increases the range of α parameters where an improvement was shown, and the magnitude of improvement increased. Given that TAPTF can encompass the best characteristics of existing approaches, as well as improving on them in what are considered critical scenarios of heavy traffic load and highly variable task sizes, we consider TAPTF to be a worthy policy for load distribution in environments where tasks are not preemptible and task sizes are not known *a priori*.

References

1. Harchol-Balter, M.: Task assignment with unknown duration. Journal of the ACM **49** (2002) 260–288
2. Harchol-Balter, M., Li, C., Osogami, T., Scheller-Wolf, A., Squillante, M.S.: Analysis of task assignment with cycle stealing under central queue. In: Proceedings of 23rd International Conference on Distributed Computing Systems (ICDCS '03). (2003) 628–637
3. Crovella, M., Taqqu, M., Bestavros, A.: Heavy-Tailed Probability Distributions in the World Wide Web. Chapman & Hall (1998)
4. Gordon Irlam: Unix file survey (1993) Available at http://www.base.com/gordoni/ufs93.html.
5. Harchol-Balter, M.: The effect of heavy-tailed job size distributions on computer system design. In: Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics. (1999)
6. Harchol-Balter, M., Downey, A.B.: Exploiting process lifetime distributions for dynamic load balancing. ACM Transactions on Computer Systems **15** (1997) 253–285
7. Crovella, M.E., Bestavros, A.: Self-similarity in World Wide Web traffic: evidence and possible causes. IEEE/ACM Transactions on Networking **5** (1997) 835–846
8. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: Measurement and Modeling of Computer Systems. (1998) 151–160
9. Arlitt, M., Jin, T.: Workload characterization of the 1998 world cup web site. IEEE Network **14** (2000) 30–37
10. Iyengar, A.K., Squillante, M.S., Zhang, L.: Analysis and characterization of large-scale web server access patterns and performance. World Wide Web **2** (1999) 85–100
11. James Broberg, Zahir Tari, Panlop Zeephongsekul: Task assignment based on prioritising traffic flows. Technical Report TR-04-05, Royal Melbourne Institute of Technology (2004)
12. Harchol-Balter, M., Crovella, M.E., Murta, C.D.: On choosing a task assignment policy for a distributed server system. Journal of Parallel and Distributed Computing **59** (1999) 204–228
13. Crovella, M.E., Harchol-Balter, M., Murta, C.D.: Task assignment in a distributed system: Improving performance by unbalancing load. In: Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. (1998) 268–269
14. Schroder, B., Harchol-Balter, M.: Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. In: 9th IEEE Symposium on High Performance Distributed Computing. (2000) 211–220
15. Ciardo, G., Riska, A., Smirni, E.: EQUILOAD: a load balancing policy for clustered web servers. Performance Evaluation **46** (2001) 101–124
16. Research, W.: Mathematica version 5.0 (2003)

A Novel Distributed Scheduling Algorithm for Resource Sharing Under Near-Heavy Load*

D. v l o[1], F io P o i[2], M i o D o io[3], n F li M. . F n¸ [2]

[1] COPPE – Engenharia de Sistemas e Computação,
UFRJ, Rio de Janeiro, Brazil
d.carvalho@ieee.org, felipe@cos.ufrj.br

[2] NCE/Instituto de Matemática, UFRJ, Rio de Janeiro, Brazil
fabiop@nce.ufrj.br

[3] Istituto di Cibernetica – CNR, Pozzuoli (NA), Italy
m.degregorio@cib.na.cnr.it

Abstract. This paper introduces SER^H – *Scheduling by Edge Reversal with Hibernation*, a novel distributed algorithm for the scheduling of atomic shared resources in the context of dynamic load reconfiguration. The new algorithm keeps the simplicity and daintiness of the *Scheduling by Edge Reversal* (SER) distributed algorithm, originally conceived to support the *heavy load* condition. Both SER and SER^H distributed algorithms share the same communication and computational complexities and can also be seen as graph dynamics where the messages exchanged between a processing node and its neighbors are represented as "edge reversal" operations upon directed acyclic graphs representing the target distributed system. Nevertheless, SER^H allows such distributed system to deal with the situation of having processing nodes leaving the heavy load behavior and going into a "hibernating" state, and *vice versa*. It is shown here that SER^H has a communication cost approximately 25% lower than the traditional Chandy and Misra's distributed solution, when operating near to heavy load conditions. In order to illustrate the usefulness of SER^H in this interesting situation, an application in the distributed control of traffic lights of a road junction is also presented here.

Keywords: dining philosophers problem; distributed algorithms; distributed traffic light control; graph dynamics; mutual exclusion; resource-sharing.

1 Introduction

Dij ' Dinnin P ilo o P o l (DPP) [9] i i l n l
i w i o n i o i o i i y (DS),
on n y, lo voi n , i n , liv n , l x l ion n -
v ion voi n , x o . A n o i i l o i v n
vi o n l x l ion in o o ,
i. ., o nno y o n on o i . In

[*] Supported by CNPq, CAPES (Brazil) and CNR (Italy).

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 431–442, 2005.

i l , n y n Mi ' i i l o i [ ] n on i
i i ol ion o l x l ion o l i y DPP.
N v l , i in on n n y n Mi ' l o i ov i ily
lo DS , on o n o ni ion o o v in i l ly
in in i ion , in vy lo i , i. ., w n o in
no i o in o in on n n o o in on o i
o , y o i l.

In i n , (S), i l n ow l i-
i l o i , w o i in lly on iv o o DS n
on i ion. S w ly li y B n ni [10] in o l o
in inin loo - o in o n wo n n o n y
n Mi ' DPP ol ion [ ]. B o n ni v li i o n
S o i n NP- o l n o o l o n in o i l
o n ovi y S yn i ov iv n i i y
[, 5]. S wo ollow : (i) i i y i i y n
n i $G = (N, E)$, w $N = \{1, ..., n\}$ i o o in no
n E i n ollow : i R_i i o o y no i in o o
o , n $(i, j) \in E$ xi w n v $R_i \cap R_j \neq \emptyset$, i , no i n j
l on o i o ; (ii) n ini i l y li o i n ion ω i n
ov E; (iii) ll, n only, no in ω, i. ., no vin ll o i o i-
n o lv , v i o o on o n n
v ll o i , o in no in n w y li o i n ion
ω'. T i n n i o in no in DS nno o i l-
n o ly on o i o . S i yn i n y
n l i ion o (iii) ov G. on i in G ni n , on n ly,
ni n o o i l y li o i n ion ov G, v n lly i ion, i. .,
o l n t, will o . An in in o y o S li in
, in i ny iv n io , no o , i. ., o in ,
n m o i [ ], n in " i n ", in lon n o ion, on ll
o in l n o G.

No i S i o l i i ion o v y no in G, n -
l i i o vily lo DS. In in in n l i l i ion
o o G' o in no in , o ily o no , o
vin no n o o in ov o , no will n iv ly
in in o n o inin S - iv n vily lo y .
T i in o S ^{H} –
n li ion o S in w i o in no v o i ili y o l vin
vy lo vio n oin in o " i n in " , n . In
o o ill ln o S ^{H} in i in in i ion, n li-
ion in i i on ol o li o o j n ion i in o
.

S ^{H} iff o S in o on o i n w n wo
n i o in no in G (in i , l i- i i n). " v l "
o x ly in S w n v no v in vy lo i ion;
i no in n o o in o " i n in " , j o in , only

o o i in l S ni v ; n in i in (x) “ v l” y n i o in no i li i nin o in i in no j n in o i n ion. L vin i n ion n only y o l on n i o in no , o n in il, in l in on ion o S ^{H} o n .

1.1 Mutual Exclusion Algorithms

T i l i o l x l ion l o i , n in [5]. T xono y o o y yn l [3] n in l :

- **Permission-based algorithms** – w n o n o o , i i ion o n i o in no . Sy y n o n o o o l io i i , y li n /o jo i i . S [1, , , 11, 13, 1 , 0, ,].
- **Token-based algorithms** – o in no n o only w n i i own o i l o i o j o i o o o . S [3, 16, 1 , 19, 1].
- **Centralized coordination algorithms** – in i , wo vio yl y o in in n l ol ion. P o in no in n o in o n o n l oo in o , w i n i in o nin n o i o j nin ni y o in o .

i i xono y, i i l x l ion l o i oin o in o o in n o [ ]. M l x l ion lo i n n l in ny o o S i n , o in y , i o- i , o i , o n wo , , ., o i l i i li ion vi o on n [1] n o il o in [6].

S ^{H}, o on i ion, i inly i ion- l o i , in i i o n y n Mi ’ o o DPP [ ] n B o n ni’ S [ ]. Mo ov , ill o l n iv o on yn i o i ion o n (in n y n Mi ’ o), lly y n o o n ow wi lo l nowl [1 , 15].

1.2 Chandy and Misra’s DPP Solution

n y n Mi ’ DPP i i ol ion n n i $G = (N, E)$ in o o n n i o oo - on in y . W n v wo n i o in ilo o “ n y”, y y (xi n o n i o yin o o i l n o ly) i o n wi l o n $\bar{G}$ on inin o no n o G, wi n y li o i n ion n ov i . e in $\bar{G}$ n n (“ n”), i. ., i o no o o o i n i o . T o i n ion o i o ll y n ion $\omega : E \rightarrow N$

$\omega((i,j))$ i no vin n. W n no i iv
o j, i o ly n o i $\omega((i,j)) = j$. I i i “ n y”,
o n i o i o o y + . In o-
o i , w n $\omega((i,j)) = i$, i n o only o in , i yin
j’ .

2 SER^H

– S ^{H} o in v-
io o S o x ion n vy lo wi o i ili y o l in
no o o ion, w i in “ i n in ” ill o n i i wo
y no i ill o in . T i n w n ion li y n i v wi-
o in in y o i o ion l n o ni ion o l xi i
o S .

An in o l i ion o ow S ^{H} wo i own in Fi 1 n . T
xi n o o ni ion nn l w n wo no n y
l on o i o . T n ionin o o ni ion nn l i
on oll y i o i ion , ● n ○. Ini i lly, i i i in
w y n y li o i n ion ω i n in , in Fi 1( ).
A in S , in no v i o o in y in -
o . A , in i n o in no v , in i
in Fi 1( ), n yn i volv . A ow in i no w v n

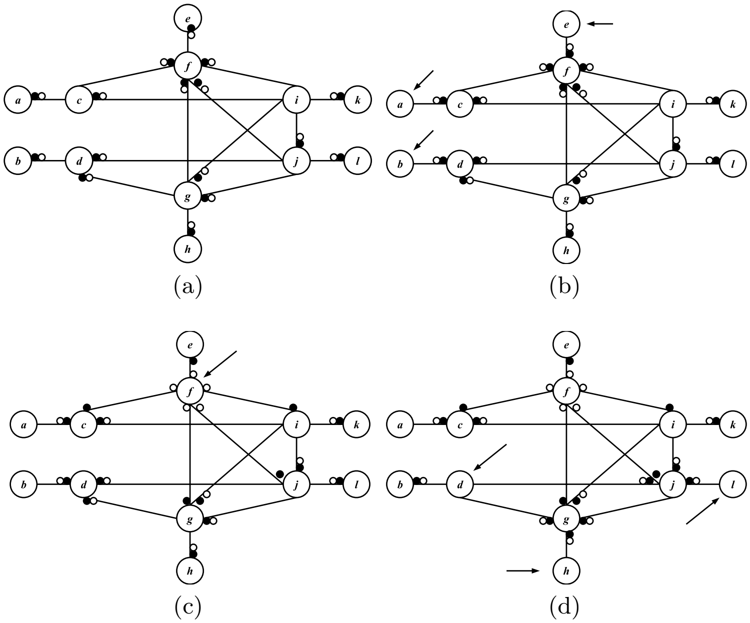
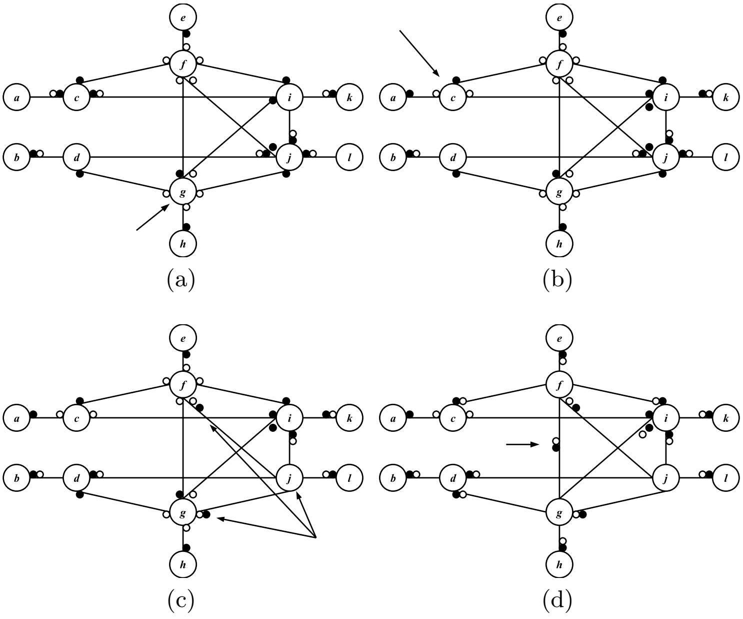


Fig. 1. An example of execution of SER^H

o in . T no ion o “ in no ” in S ^{H} i iff n o in S : in no o y o “ownin ” ll ●- i ion (L ll i ● . , ...). Mo ov , v ion o ●- i ion lon o o only w n no own o on in ○- i ion o . T i i in vio i in o in o o i l n n w i n in . v Fi 1( ): no f i o n i n in ; in i , i v only ●- i ion ow i n i o , w il in ○- i ion . No i no $c, d, e, h, k,$ n l ●- in no ; y l o o on o . Followin x l , no d, h n l v o n v i i ion o in o ov onv n ion, in Fi 1( ). T inin ●- in no v no v i i ion y ill o in . v now no g: i j o ●- in no , n w n i ni i o ion i will v i ●- i ion n n i n in . W o l o v , in i , i o no v ●- i ion ow f i o no own o on in ○- i ion o . A i o n , ll f i i n in , n i i no i in o wi i n i o . T n l i ion i own in Fi ( ).

Fi ( ) ow no c n in H- . W n o ●- in no - i o w n i o , i v ●- i ion ow i n in n i o own in Fi ( ). T i i ion o y no j w n w in f n g, w i y i n v ll ○- i ion ow i n i o , x j, in Fi ( ). U on ownin ○- i ion ,



(a) (b)

(c) (d)

Fig. 2. An example of execution of SER^H (continuation)

n i o i i ly v o on in •- i ion , w ll w n o lly i in S ^{H} . il will l o i- , . ., i ion n in Fi ( ), w i ion ov in o o i i ion lon nn l lin in f n g.

2.1 Formal Description of SER^{H}

v y no in y y in on o lly x l iv : (*R*), o v l (*W*), o (*H*). W no y $s(i)$ n o no i. i nnin o w i in ; i n in . Al o, ollowin n i ion ξ li :

ξ^{RW} – o w n no xi *R*- n n *W*- ;
ξ^{RH} – o w n no n *H*- in o ;
ξ^{WR} – o w n no in i o in ll o ;
ξ^{HW} – o w n no i wo y n i o .

Fi 3 ow n i ion in S ^{H}. v no n n *H*- only vin n in *R*- ; in i ion, w n no l v *H*- , i n ily n *W*- . T l o i lw y xi l on no in *R*- , in nnin no n y o w n i o in i n in no . W will n o i i in S ion 3.

S ^{H} y o , n ly $\mathbb{PM}$, $\mathbb{PP}$ n $\mathbb{MM}$. A o x-l in l , in i v l in wo i in i , $\mathcal{G}^{+}$ n , $\mathcal{G}^{-}$, o vin G j n . T y n in o - i wi o ni ion nn l , o on in o xi in in G. An i o n o o ni-ion nn l li in i l io i y : l o y $\mathbb{PP}$ o i o j n y $\mathbb{MM}$ in o o i i ion, in i l-n o ly n o nn l (i, j), y $\mathbb{PM}$ i n o i o j. A

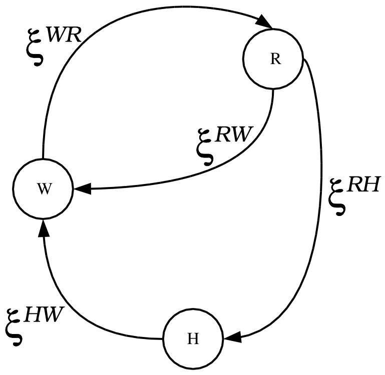


Fig. 3. State transitions

n x l , on i no f n g in Fi ( ): i i ion o n o w n wo n i o in no l vin H- i l n o ly; in i y y w n wo no i , w o l l o lo . T io i y i ov voi o i ili y.

i i in no wi in S ^{H} o wo ool n v i l o i o o ni ion nn l, i. ., o n i o in G. D no y N_i o n i o o no i. T v i l , P_i^j $(i \in N, j \in N_i)$, i w n no own o on in ●- i ion, n l o wi . M y ℙℙ n ℙM o in o v l in o on n i o in no .

M y ℙM l o in i v l in v l $\mathcal{G}^-$, in w i n o i n ion on nin : i vin o i n ion o on in in $\mathcal{G}^+$, i o v in in $\mathcal{G}^+$. T o i n ion o $\mathcal{G}^-$i no y v i l M_i^j $(i \in N, j \in N_i)$ n n l o o i y y 𝕄𝕄. T o- o o S ^{H} n [6].

2.2 SER^H Initialization

T o n v l n o in in in l li i $\mathcal{G} = (N, \mathcal{E})$. in $\mathcal{E}$ i n n o i l o o (i, j, c), w $c \in \{\bullet, \circ\}$. lon in o o l l ●, n i i in i in l o i y lo l v i l P_i^j. lon in o v l l l ○, n i i in i y lo l v i l M_i^j. W will loy x ion o o wo l l ● n ○ lin in wo n i o in no . T n:

$$\forall (i, j, \bullet) \in \mathcal{E} \quad \omega((i, j, \bullet)) = i \longleftrightarrow P_i^j = true \qquad (1)$$

$$\forall (i, j, \circ) \in \mathcal{E} \quad \omega((i, j, \circ)) = i \longleftrightarrow M_i^j = true \qquad (\ )$$

Fo l in o i n ion o n o i l o no in ini i li ion o $\mathcal{G}$.

Rule 1. $\mathcal{E}$ $\omega((i, j, \bullet)) = \omega((i, j, \circ))$

Rule 2. $\mathcal{E}$ $\omega((i, j, \bullet)) = i$ $\omega((i, j, \circ)) = j$

Rule 3. $\mathcal{E}$ $\omega((i, j, \bullet)) \neq \omega((i, j, \circ))$

l n 3 i ly v y in iv no n v own o ● n ○ i ln o ly.

In o o n l , w n o lly $\mathcal{G}^+$ n $\mathcal{G}^-$. T on i o i n $\mathcal{G}^+ = (N, \mathcal{E}^+)$ i yin (3). (ll i i i on in y l l ●.)

$$\forall (i,j,\bullet),(i,j,\circ) \in \mathcal{E} \quad \omega((i,j,\bullet)) = \omega((i,j,\circ)) \rightarrow (i,j) \in \mathcal{E}^{+} \qquad (3)$$

T o i n $\mathcal{G}^{-}= (N, \mathcal{E}^{-})$ i n i il ly nnin - i o $\mathcal{G}$ in y l l o:

$$\forall (i,j,\bullet),(i,j,\circ) \in \mathcal{E} \quad \omega((i,j,\bullet)) \neq \omega((i,j,\circ)) \rightarrow (i,j) \in \mathcal{E}^{-} \qquad (\ )$$

Rule 4. $\mathcal{G}^{+}$ $\mathcal{G}^{-}$

2.3 SER^{H} Correctness

T l low ow S ^{H} v l 1, n 3. D o l o , oo n o n [6].

Lemma 1.

- ξ^{RW}, ξ^{RH} ξ^{HW}
- ξ^{WR}
- ξ^{RW}, ξ^{RH} ξ^{HW} $\mathcal{G}^{+}$ $\mathcal{G}^{-}$

T l ov i li :

Theorem 1. ^{H} $\mathcal{G}^{+}$ $\mathcal{G}^{-}$

3 Simulation Results

T DPP o l llow ilo o o n i o " in in " o " n y" ny oin o i x ion. In o o l o i , no n wi o *H*- o *W*- w n i n o o . How v , S ^{H} wo wi wo i ion : , no wo y ny nnin n i o in no in o o l v *H*- ; on , no i o y on vy lo on i ion o l in o o . Sin S i no i n o l wi i y lo on i ion , in i ion w n i l ion l ow o o S ^{H} o wi n y n Mi ' l o i n ow o ov o o i ion in l li o l . (ll S ^{H} n S v x ly w y n vy lo on i ion.)

3.1 Message Costs

In o o o o o S ^{H} n n y n Mi ' l o- i , w i l n yn ono i l ion o on inin v n no o lly in onn (K_7). T i l ion o n y n Mi ' l- o i w l y iv n o ili y p_h o n in no o " in in " o " n y" n o i l ion y l . n o n , S ^{H} i l ion w ov n y o ili i p_r n p_w: p_r i o - ili y o no o in in vy lo on i ion o

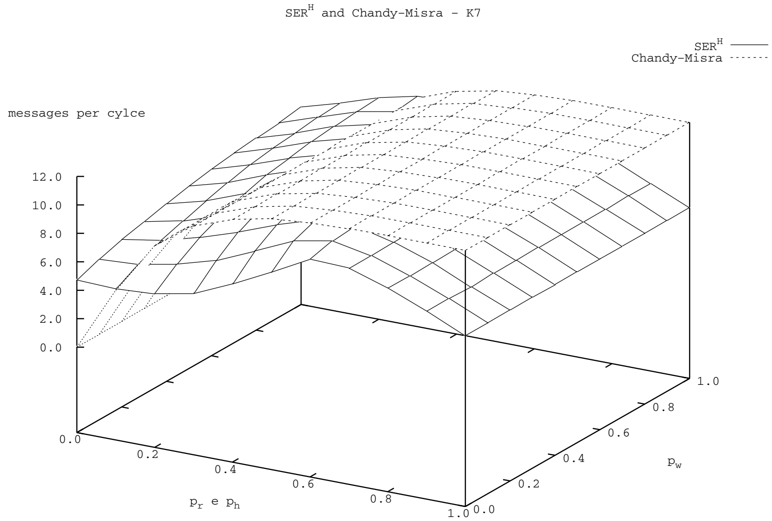


Fig. 4. Number of messages per cycle in SER^H and in Chandy and Misra's algorithm, $0 < p_r, p_h \leq 1$ and $0 < p_w \leq 1$

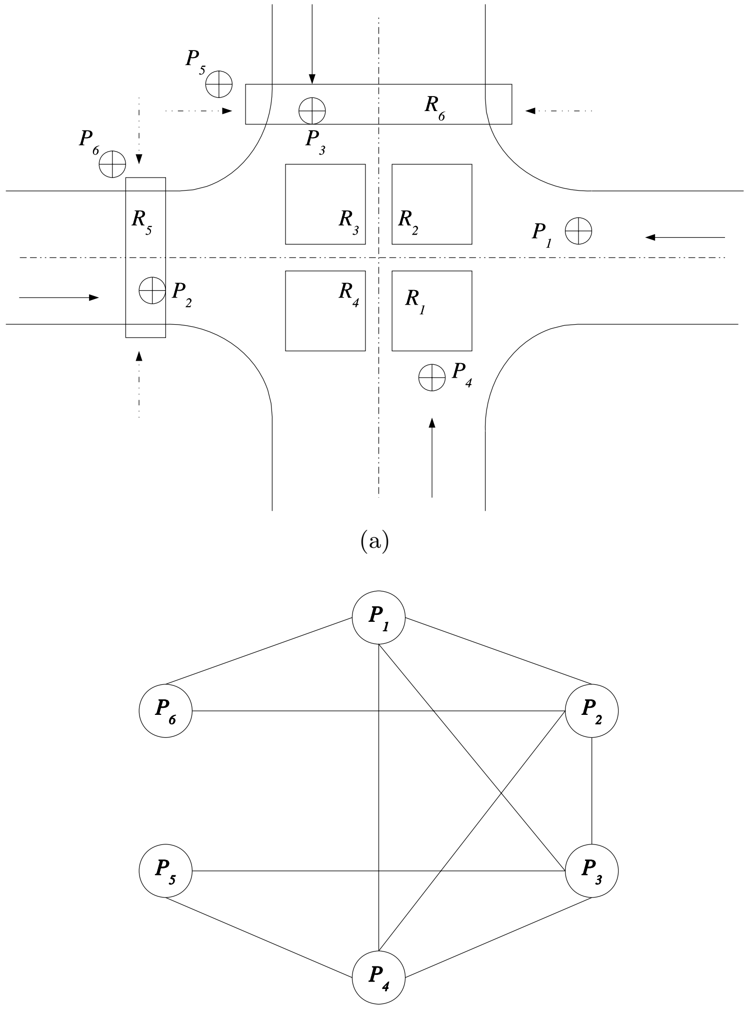
o , n p_w i o ili y o nnin no o i w in i n in n i o in no . B i , no nno n H- w n ll i n i o in H- in o o v n lo l in ion o l o i .

Fi ow n o y l x n y y v . o ili i $p_h, p_r,$ n p_w. v S ^{H} l n n y n Mi ' l o i w n y o vy lo . Mo ov , S ^{H} l o y n y n Mi ' w n i i o in in vy lo . T i i o ly o n y n Mi ' l o i x n wo w n n i o in no i (n) n S ^{H} n o i li i ly in only on .

3.2 Distributed Traffic Light Control of a Road Junction

W o l o j n ion n in Fi 5( ) in S ^{H} n n y n Mi ' l o i . T o l on i o w no li n y i l l l $P_i(i \in N)$, n o onfli ion l l R_j. n o in o o vi- o onv n ion n l in i i in Fi 5( ). T v i l n i n ov n w o l y i i yn ono wi iv n v i l / i n iv l o ili y (n ni ion o i n o n [6]).

In n y n Mi ' x ion, li wi non- y l n n i in n l o " n y" n n w i o i n o o .



(a)

(b)

Fig. 5. Road junction and the graph model

In o o i y i ion n li , S ^{H} i l n-
ion n i ion l no , lin o o i in l o in no . T i no
wo “w ” o w li o in no w o -
o in o non- y; o wi , li o in no
n H- w n i o y. T i vio ov o
on i ion n ov . T i l io i y o ni ion nn l
i l n wi i l o o ol. T S ^{H} i l-
n ion ow o ni ion o 1 %, 1%, n % low n n y
n Mi ’ v ion w n y w o in wi i n iv l

o ili y o 10%, 50%, n 0%, iv ly (v i l iv l o ili y o 0% in ll .)

4 Conclusions

A n w i i lin l o i in n - vily lo y w in o n i o n on . I li ion on i- i on ol o li v l i ln in i in in i- ion. I w own on i ly low o ni ion o w i v n n - vy lo on i ion, o wi n y n Mi ’ l o- i . F i o in on ow li iv o i ion on o olo y o iv no i on n y o l in y o i i in . Mo ov , li ion o S ^{H} on o o l i n o yn ono i in o low- ow i i l i i n y i n l.

References

1. D. Agrawala and A. El Abbadi. An efficient solution to the distributed mutual exclusion problem. In *Proceedings of the* 8^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 193–200, August 1989.
2. D. Agrawala and A. El Abbadi. Exploiting logical structures in replicated databases. *Inf. Proc. Letters*, 33:255–260, 1990.
3. A. Arnold, M. Naimi, and M. Tréhel. A $\log n$ distributed mutual exclusion algorithm based on the path reversal. *Journal of Parallel and Distributed Computing*, 34:1–13, 1996.
4. Valmir Barbosa and Eli Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4):562–584, October 1989.
5. Valmir Carneiro Barbosa. *Concurrency in Systems with Neighborhood Constraints.* Ph.D. thesis, UCLA Computer Science Department, Los Angeles, 1986.
6. D. Carvalho, F. M. G. França, and F. Protti. Pseudo-code of Scheduling by Edge Reversal with Hibernation. http://www.if.ufrj.br/ ∼ carvalho/serh.html, 2004.
7. O. S. F. Carvalho and G. Roucariol. On mutual exclusion in computer networks. *Communications of ACM*, 26(2):145–147, 1983.
8. K. M. Chandy and Jayadev Misra. The drinking philosopher’s problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, October 1984.
9. E. W. Dijkstra. Hierarchical ordering of sequencial processes. *Acta Informatica*, 1(2):115–138, 1971.
10. Eli M. Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18, 1981.
11. K. D. Gifford. Weighted voting for replicated data. *Proceedings of the* 7^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 150–159, 1989.

12. Kenneth Goldman and Joe Hoffert. A modification to the chandy-misra dining philosophers algorithm to suport dynamic resource conflict graphs. submetido para Information Processing Letters.
13. N. Plouzeau J. M. Helary and M. Raynal. A distributed algorithm for mutual exclusion in an arbitrary network. *The Computer Journal*, 31(4):289–295, 1988.
14. Yuh-Jzer Joung. Asynchronous group mutual exclusion. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, pages 51–60, New York, June 1998. Association for Computing Machinery.
15. J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, November 1990.
16. G. Le Lann. Distributed systems: towards of formal approach. In *IFIP Congress*, pages 155–160, North-Holland, 1977.
17. N. A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. *Proceedings of* 7^{th} *ACM Symposium on Operating Systems Principles*, pages 137–151, 1987.
18. M. A. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.
19. J. A. Martin. Distributed mutual exclusion on a ring of processors. *Science of Computer Programming*, 5:256–276, 1985.
20. H. Garcia Molina and D. Barbara. How to assign votes in a distributed system. *Jornal of the ACM*, 32(4):150–159, 1985.
21. K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.
22. M. Raynal. Prime numbers as a tool to design distributed algorithms. *Inf. Process. Lett.*, 33(1):53–58, 1989.
23. Michel Raynal. A simple taxonomy for distributed mutual exclusion algorithms. *Operation Systems Review*, 25:47–50, 1991.
24. Injong Rhee. A fast distributed modular algorithm for resource allocation. In *Proceedings og the* 15^{th} *International Conference on distributed Computer Systems (ICDCS '95)*, pages 161–168, 1995.
25. P. C. Saxena and J. Rai. A survey of permission-based distributed mutual exclusion algorithms. *Computer Standards and Interfaces*, 25:159–181, 2003.
26. Walter, Welch, and Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks: The Journal of Mobile Communication, Computation and Information, Kluwer*, 7, 2001.
27. J. L. Welch and N. A. Lynch. A modular drinking philosophers algorithm. *Distributed Computing*, 6:233–244, 1993.

Internet Computing of Tasks with Dependencies Using Unreliable Workers*

(Extended Abstract)

Li o n o M l wi

University of Alabama, Tuscaloosa, AL 35487, USA
{lgao, greg}@cs.ua.edu

Abstract. This paper studies the problem of improving the effectiveness of computing dependent tasks over the Internet. The distributed system is composed of a reliable server that coordinates the computation of a massive number of unreliable workers. It is known that the server cannot always ensure that the result of a task is correct without computing the task itself. This fact has significant impact on computing interdependent tasks. Since the computational capacity of the server may be restricted and so may be the time to complete the computation, the server may be able to compute only selected tasks, without knowing whether the remaining tasks were computed by workers correctly. But an incorrectly computed task may render the results of all dependent tasks incorrect. Thus it may become important for the server to compute judiciously selected tasks, so as to maximize the number of correct results.

In this work we assume that any worker computes correctly with probability $p < 1$. Any incorrectly computed task corrupts all dependent tasks. The goal is to determine which tasks should be computed by the (reliable) server and which by the (unreliable) workers, and when, so as to maximize the expected number of correct results, under a constraint d on the computation time. We show that this optimization problem is NP-hard. Then we study optimal scheduling algorithms for the mesh with the tightest deadline. We present combinatorial arguments that completely describe optimal solutions for two ranges of values of worker reliability p, when p is close to zero and when p is close to one.

1 Introduction

T i in v lo in lin o y o i ovin li y o
l o x n li ly ov In n . W in o o in-
o i l o i i ion o l , ow o l i NP- , n n y
o l i o w w iv o i l olyno i l i l-
o i .

[*] Contact author: Grzegorz Malewicz, Department of Computer Science, University of Alabama, 116 Houser Hall, Tuscaloosa, AL 35487-0290, USA, Phone (205) 348-4038, Fax (205) 348-0219.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 443–458, 2005.

T i l n o n ili o onn o In n .
H n in i ow n n l ion o i i o
n o li vol o wo iny ion o o o
o i ion l, o n li , o . S v l l i l-
n ion o In n S o xi o y [,11,16, 1]. T n l
olvin o l o o o l n o . . . . . In i l n-
ion , o , ll . . , llow ny o o , ll . . ,
o i n ownlo i o o w . T n o w
o v , ownlo o i i , x
, n n l o v . T i o . W n
n o wo i l , In n S o i v i o -
in . Fo x l , S TI@ o oj o i o 5 . 9
T flo [3]. T In n S o l o ll In n o W
o in l o , o Hi -T o o in i [ ].

S TI@ o [9] l o o o oj w n on
lin wi i y o l . n o i o n li y o l
n y wo . So o , v n in
wo , li l ; y will o ly x i n y v .
How v , wo o only n li l . T i y y n o v
in o l o nin n il o x l y ov lo
o o , o y y ivin ly l i o v o i n wo o
o o in in n iv in i n on S TI@ o li o
on i ni o wo . S v l w o o o i ov li y
o l o . T n o o lin li ili y o wo
on i o y o in ion wi wo [10,], in o w i
w i n o w i wo [0], n in o l i l wo [11],
n v i yin l n y wo [6, , 5, 3].

I in n l i i n n lly i l o v lo o
in w x o ly, wi o in x-
on li l o (. [1]). T i i l y i ni n on n
on o in o ion i y i y li . In ivi l
o o ion y i o ion lly in n iv , n o i y
n li i o x oo ny o on li l o y
. on n ly, v y no lw y now i iv n w x-
o ly o no . W n v n n i , n l o
i in o , n ll n n , v n i x o ly, y o
in o l , i ly i in n , i ly o in i ly, on
l o on w x in o ly. I l i l
in in o y v i i on o l n o o
l , w il o y v low i . T o i l y y n
i y i o n o j i io ly l w i o l x -
li ly, n w i o n l o n li l wo , o o xi i
o l n o o l .

T i in v lo in lin o y o in in n
o o l o x on n li l wo , w n v -

n n i . L i o off . A n iv lin o wo l
o x ll on li l o only. o , n i no
n o j i io l ion ll. How v , n o li l o
y i ll o o n o n li l o . T o ,
w n wo i i n o li l o only, l iv ly o i wo l
n o o l n i o ion. W o l o ion
i y in l in n li l o in o ion, o o -
in n o o l . T w x n o o
l , o iv n i y li i n n i w n
, i l o wo : li ili y o o n -
lin o o l o ion. l i o l i o lly n n i
l ion i .

n n l w y o o l n li ili y o o i o o -
ili i in . o will x o ly wi in o ili y.
T i ion o l j i , o x l , y on o o
o ion o in In n S o i ov lo o o [9].

Tow i n w o l o l o n In n S o .
o l x n In n l in o n ly y o n [19].
T o ion i o l y ni i y li . no in
n . T i n n o n n o o in
y . o i x wi o ili y p_i n
wi o ili y 1$-p_i$. T i o ili y i ll o o . T
y o o l y . Ini i lly ll o o
l wi n l . A ny i i t w l o ,
y i, n n li i l l . T n w l li i l l
wi l wi o ili y p_i, n wi l
wi o ili y $1 - p_i$. Any o no v ny l
ll i n v x l i l wi n li i l l . Any
i x in o ly l o ll n n ; o i
l will in o v n i x o ly. T i d y w i
ll o x . T o l i o in w i o
o l x w i n w n, o o xi i x n o
o l . Solvin i o i i ion o l i i o n . n wo l li
o li o i l i lin o ow o ff iv ly n i ly x
o ion o o o n n , in n li l o .

T o o i i o y i v ion o lin o l .
W on i ll (wo i n ion l) i o o o k^2 no
n in o k ow n k ol n . no n o no in
n x ol n (i i xi) o ow, n n o no in n x
ow (i i xi) o ol n. oi o i o iv y
onv ni n w y o o ion n y
i in i (. [19]). W inv i ow o o i ly
o i l i. ., w x lin d o $k-1$. W i in l o
w o li ili y i 1; i o i ll v . Any o o
li ili y $0 < p < 1$; i o i ll wo . ion

i in l li l v n wo li ili y p o n l “ oxi ion” o n In n S o o o o n li l wo .

W no v n i l o nno in o o in n l wi o o in on o , on o l ill o - n n ly on n li l o n jo i y vo in o in o i ov li y o l . S n o i o o on l o i o i w w w n o li l o o i l (w i i on- y i o y in l o), w il ill o inin li y o i l y j i io ly i nin o o .

Contributions. T i in v lo in lin o y o xi i - in n o o l o wi n n i x n li ly ov In n . i on i ion ollow :

( ) W in o o ili i l o l in n o in wi n li l wo , n n w o in o i l o i i ion o l .

( ) W ow o i i ion o l i NP- y in o ion o B l n o l Bi i S P o l . In o - l i NP- v n w n i o i i o y in l (li l) v n (n li l) wo .

( ) W iv olyno i l i o i l lin l o i o n i lin $d = k-1$, v n wo , w wo li ili y p ll in o wo n o v l . W ow x ion i x- i i w n x o ly in - o , n v x x ly on “l v l” o . W on wo lin i . T i n on v l o li ili y p. T i i w n li ili y i lo o 1. W o l ly i xi l l in i i . T v o l x “ n l” ny i . S i lly, ny i , i o n o “ li i l ” n x iv n n on in n x o . T o i on l “l v l” o . A i i , v o l x o n n o on li i l , w i n o o n l on i on l l v l (y wo , in w i oi o no , w ow), n wo o l x ll o li i l . In- i iv ly, i w n p i lo o 1, n o i l l “ - n n iv n”. T on i i w n li ili y i lo o 0. W l o o l ly i xi l l in i i . T v o l x n “ ” ny i . S i lly, v o l i x - in o ow n i o ol n, o i o l x in l o ol n n o o ow. In i iv ly, i w n p i lo o 0, n o i l l “ n o iv n”. T on- ion wo i in i i , w li v , n i o n on i ion o i in i o l non- ivi l n in in o o i l ol ion (w in o x lo).

Paper organization. T o i ollow . In S ion , w n o l o In n S o in wi n li l wo , n o l no i i ion o l o xi i in x n o o l o . In S ion 3, w ow o i i ion o l i NP- . T n, in S ion , w iv olyno i l i o i l lin l o i o . N x , in S ion 5, w i l wo . Fin lly, in S ion 6, w on l n i wo . D o li i ion o oo o i o i x n .

2 Definitions and Preliminaries

A i y li $G = (V, E)$, o o o , on n no o o o n in o ion flow w n (ll ni in i). W o n o no . A i n $u_1, u_2, \ldots, u_k$ o wo o o no i n o u_i o u_{i+1}, o $1 \le i \le k-1$. In no n v $u_1 = u_k$. Fo iv n no u, $P(u)$ i o o u i. ., o ll no v, i n o v o u; $C(u)$ i o o u i. ., o ll no v, i n o u o v; $A(u)$ i o o u i. ., o ll no v, i o v o u; n $D(u)$ i o o u i. ., o ll no v i o u o v. No $u \notin P(u)$, $u \notin C(u)$, $u \notin D(u)$, n $u \notin A(u)$. A u $P(u) = \emptyset$ i ll , n w n $C(u) = \emptyset$ n u i ll .

A i w n x n y w o . A l wo o on n . T o on n i n ion x n l n $t \ge 1$ n n o $x(t)$ x i t. T i $\mu \ge 1$ $x(1), \ldots, x(\mu)$ i no y, n i ion o ll . T n μ i ll o l . x ion o ny on ni o i . A n only x w n ll i n o l y v , o o ny $1 \le t \le \mu$, $x(t)$ o w o n o in $x(1) \cup \ldots \cup x(t-1)$. T on o on n i n ion c n l n v n n n $c(v)$ no in x v. W m o in y . I ny o x o on ni o i , o o ny $1 \le t \le \mu$, n ny i, n $\left|c^{-1}(\{i\}) \cap x(t)\right|$ o x y o i i t i o on . Fo ny i l on l (x, c).

o . W n o i x , n wi o ili y p_i o x u , in n n ly o xion o o . How v , wi o ili y $1 - p_i$ o x . W ll p_i o o . S in o xion ff l o v y in $D(u)$. In i iv ly, n in o ly x u o l o ny n n v, l o u i , i ly o in i ly, w n v i x . W y , i n ll i n o x o ly. In on , l o i in o ,i i o on o i n o i x in o ly.

W n o x n o o l o iv n l (x,c). In o o u o o o ly, v y in $A(u) \cup \{u\}$ o o ly. T n ion c n w i o x o . So y in n n , o ili y l o u i o i o $\prod_{v\in A(u)\cup\{u\}} p_{c(v)}$. L E_u in i o n o v i l l o 1 i l o u i o , n 0 o wi . T n o l n o o l i l o $E = \sum_{u\in V} E_u$. By lin i y o x ion

$$\mathsf{Exp}\left[E\right] = \sum_{u\in V} \mathsf{Exp}\left[E_u\right] = \sum_{u\in V} \prod_{v\in A(u)\cup\{u\}} p_{c(v)} \ .$$

o l i o n l (x,c) xi i i x ion.

Constrained Computing with Unreliable Workers

A G n n in o ion flow w n , lin d, n m o wi li ili i $p_1,\ldots,p_m$.

Fin l (x,c) wi n o d xi i x n o o l .

T i o on w i in l o , ll v , wi li ili y 1, n ny o o , ll wo , li ili y $0 < p < 1$. In i o o i i ion o l i l o l ion. S o R i o v x . W ll i . L $E(R)$ n o v i l l o n o o l o l wi R o x y v . T n x n o o l i l o

$$\mathsf{Exp}\left[E(R)\right] = \sum_{u\in V} p^{|(A(u)\cup\{u\})\setminus R|} \ .$$

No i x ion n on G n R, o no n on n x in w i v n x , no i lin on in n viol o i v x o n on i . T ivi lly, x ion i xi i w n ll x y v , $R = V$. How v , n i y n i n o l i l , o i i w n v i o o x ny . W loo in o v R xi i x ion, n n ion x, o on i x y v ny oin o i in x n n o x i o d. W o i i v ion o o l In n S o in wi Un li l Wo (ISUW).

3 Complexity of the Problem

W on i i NP- o olv o l o In n S o- in wi Un li l Wo . T oo i o o o wo . W nown NP- o l o l ll B l n o l Bi i S - P o l ([5] o l T) o n “in i ” o l o l in w o nion i ll. T n w ow ow o iv n w o ny in n o in i o l in n l o i n ol ion o o- l o In n S o in wi Un li l Wo . T i will i i ly i ly on in o in wi Un li l Wo i l o NP- .

Many Subsets with Small Union (MSSU)

Non y $S_1,\ldots,S_n$ o $[n]$, i nion i $[n]$, n n $a \leq n$ n $b \leq n$.

n a o l w o nion in li y o b?

W iv ion i o B l n o l Bi i S P o- l (B BS) ([5] o l T , n [1] o n l n n) o MSSU.

Lemma 1.

T ion i o B l n o l Bi i S P o- l (B BS) ([5] o l T , n [1] o n l n n). ll in o l w iv n i i n n k n w w n o now i on in n in o l i i wi k no on l n k on i .

L ny i i G on $n-1$ no n k. on i n x n G' wi on x no n i i ol . N lly, G' i l o i i . v i l n o l i i wi k no on l n k on i in G, i n only i i in G' (i ol no in G' nno lon o).

W now n n in n o M ny S wi S ll Union P o l . L M o l n o j n y ix o G'. No o o ow n n i - o ol n n ll wi on , o i ol no . W n S_i, $1 \leq i \leq n$, o i i v o o i l o ol n i o M. So S_i i non y (o o o ow) n i nion $S_1 \cup \ldots \cup S_n$ i x ly $[n]$ (o i - o ol n). L $b = n - k$ n $a = k$.

T G' l n o l i i on k no , i n only i w n n ow n ol n o M o o l k y k o n M o only. B i n on i n only i w n l $a = k$ o , o nion o l in li y o $b = n - k$. T i o l oo .

W n iv olyno i l i T in n o ion o MSSU o ISUW. In o n o ion w on i i wi o i wi in n l n o i wi o .

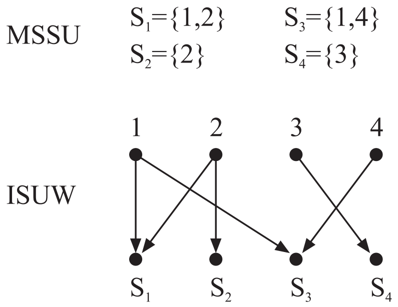


Fig. 1. Reduction

Theorem 1.

W ny in n o MSSU P o l n ow ow o n w ion o in o l , in n l o i xi i x ion o in n o ISUW P o l .

L $S_1, \ldots, S_n$ ny non y $S_1 \cup \ldots \cup S_n = [n]$ n a n b n o n. W on n in n o ISUW P o l . T i i . I wo “l v l ”: “ o o ” o on o , w il “ o ” o on o l n o $[n]$. So l v l n , n o l n o in i n. W l n o o i o o o j, i l n i i in S_j. S Fi 1 o n x l o on o iv n . W no i o i lin o l on o o , nion o i $[n]$. Mo ov , o o i lin o o , i non y. W n lin o $d = b + 1 \le$ n, n li ili y o wo o $p = 1/n^2$. L R v xi i x n o o l n on in .

W ny xi ol ion, in l in R, v i l . Fi , $|R| = d$. In , in li y o R nno l , n lin on in wo l viol , n i nno ll , n x ion o l i ly in y x in on o on v wi o viol in on in . S on , l on o R lon o o o l v l. In , i R d on o l v l, n on o wo l x i d o l . B i il , n o i il o l only x i $d + 1$ o l , viol in lin on in . T i , o i il on , l on o R on o l v l. T o v ion i ly ny v xi i x ion in li y d n l on on o n l on on o o l v l. No o ny v wi o i , i ivi l w y o x n n on in . T v on in in o oo will v o i , n o w o no x li i ly on n ion x in in o oo .

T v R' wi x ion R,
x ly $d-1$ o o R' lon o o l v l. In , w on
lon wo in R lon o o o l v l, w n
ov o o o R n n w o o R wi o in
x ion. T w n on ovin n in n il x ly on
o R lon o o o l v l, n v in x ion. S o
wo i in v n w o R lon o o o
l v l. Sin $d \le n+1$ n wo o R o o l v l, n
i u o l v l i no in R. L R' v l o R
x v i x l u i in l in . l ly, R' in li y d
n l on o o n l on o l v l. I
in o n R' no ll x ion. In R, u on i p o
x ion n v on i p^h, o o $h \ge 0$ (w n v n only
on R n $h = 0$). In R', ow v , u on i 1 n v on i l
$p^h \cdot p$. In i ion, on i ion o ny o n v u n
will in w ll. No o will n i on i ion. So iff n
in x ion i l $\left(1+p^h \cdot p\right) - \left(p+p^h\right) = (1-p) - p^h\,(1-p) \ge 0$. In
i iff n x ly 0, R i xi l. T x ion
o R' i x ion o R.

S o xi x ion i z. W ll y in in z,
w n n w w a in in n o MSSU P o l ,
nion o in li y o b.

W v n i v R', x ion o
R' i o R, n $b = d-1$ o o R' on
o l v l n on i on o o l v l. L n o ow
on i o z. T inin $n-b$ o no x y v .
T on i ion o o o x ion i $b+(n-b)/n^2$. W now
y on i ion o o o . L o o n no
o o i x y v . I ll n o o o on
b o x y v , n o o will on i
x ly $1/n^2$; l k n o o o u n o u
on b , $0 \le k \le n$. ll o o n .
So o inin $n-k$ o o n i on
o b x y v . H n o o will on i
o $1/n^4$ o x ion. L now o n o i on o o
x y v . I $k \ge 1$, n o o on
k , o wi x ion o l in y x in
v ny o k in . T , w n $k \ge 1$, x ion z i in
in v l $[y, y+1/n^3]$, w $y = b+(n-b)/n^2+1+(k-1)/n^2$. I $k = 0$, n
o o l on n i no x y v , n
o x ion z i in in v l $[y', y'+1/n^2+1/n^3]$, w $y' = b+(n-b)/n^2$.
on n ly, $n+1$ in v l , o $k = 0, 1, , \ldots, n$, o no ov l . T
xi k o o w o n o o o b . Sin b
i nown, v l o k n in y in in z. v l o i
i no o i l i ly o n k o o w o n

o i o o *b* o , n xi x ion wo l i ly l n *z*.

Corollary 1.

4 Optimal Algorithms for the Mesh

In i ion w n o i l ol ion o lin o l o In n S o in wi Un li l Wo on . W x lin o i on o i l on iv n . Un i on in , w o l ly i o i l ol ion o wo n o v l o li ili y *p* o wo . W n li ili y i lo o o, n v xi i x ion i n only i i on in only on in o n o “ ” . T wo in . W n li ili y i lo o on , n v xi i x ion i n only i i on in only “ n l” . T x on n i lly ny . T in o ion n n on i o ny o i l v . T n o i l lin l o i iv n o wo n o wo li ili y. In i l , n n l v n , n o in o i l n in o i li y o n .

4.1 Preliminaries

A M_k, o ny iv n $k \geq 1$, i wi no $V = \{(i,j) \mid 1 \leq i,j \leq k\}$. T i n o ny no (i,j) o no $(i+1,j)$, lon o no lon o . Si il ly, i n o (i,j) o $(i,j+1)$. W in o o i n ion o . S i lly, no $(1,1)$ i No -W no , n no (k,k) i So - no . S Fi o n x l o n i o i n ion. A o l ni ion o o i n ion o l l o . W Fi o o “l ”, “ i ” . A ℓ i o no (i,j) o M_k $i+j=\ell+1$. T x ly $k-1$ non- y l v l o M_k. T l v l i ion no o . Fo ny no on l v l ℓ, i no n , n n i on l v l $\ell-1$, i no il , n il i on l v l $\ell+1$. ol n j i o no v on oo in l o j. ow i i o no v oo in l o i.

W in wi l x o o n o i l ol ion o o i o l . T n l ny v xi i x ion v x ly on on l v l, no o n no w .

Lemma 2. $k \geq 1$ M_k R x x $k-1$ ℓ $1 \leq \ell \leq$ $k-1$ R

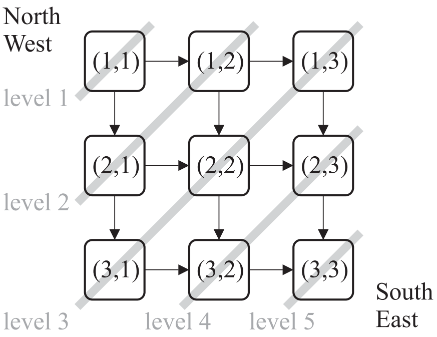


Fig. 2. Mesh M_3

T i l on i ly i li lin . Sin ny v R xi i x ion x ly on l v l, w n ivi lly n n ion x i w n x lin on in i . W i ly l x ion l v l y l v l (- o). T w o no x li i ly on ny n ion x o ny v , in in in n o i x n ivi lly n .

T n x ion w n o n w i : W i o l l on l v l? W no o ny li ili y p o wo , $0 < p < 1$, i i lw y o x on v n o , in o n x ion will i ly in . Un o n ly, on ny iv n l v l no o l (no i n o ny o on l v l). T o , i i l o v ion o no l i w i o iv n l v l o l x y v . W n iff n i ion l o i in . In in o ion w n wo o i l l o i , on w n p i lo o 1 n o w n p i lo o 0.

4.2 Optimal Algorithm for Workers with High Reliability

T i ion o l ly i o i l v w n li ili y o wo p i lo o on . W o v n i i w n v x wi o n n . T i in w i o ny o l v l v o l x . Fo v n l v l , l ion i i o , w ow i o no , x ion will no ow w oo .

T n x l x l in lon p i lo no o 1, i i v x wi i ly o n n . T oo o v wi o n n i ly o o x ion n ny wi w n n no w i o in v .

Lemma 3. *G* n p $(1-1/n)^{1/n} <$ $p < 1$, *u* *w* $D(w)$ *w*

$D(u)$ u, $|D(w)| \geq 1 + |D(u)|$, R u w $R \cup \{w\}$ $R \cup \{u\}$

No $(1-1/n)^{1/n}$ i y o i lly lo o $1-1/n^2$.

T l l o l ion o p lo o 1. n n o ny l v l o M_k, o y " n l" lo ion o l v l v o n n o on l v l. T l v l will v in l " n l" in n o i l R. T in i , ow v , i ny v n l v l wo " n l" v n o n n . T xi n o oo o i i y. n x o l i o on i i i y no ff on x n o o l .

Fo iv n M_k, w ll v R i i i o o o i . I on in (i,i), o ny $1 \leq i \leq k$, n , in i ion, o ny $1 \leq i < k$, i $(i,i+1)$ o $(i+1,i)$, no o . No o ny n l v , l v l o M_k on in x ly on o . W ov x ion o n l v y no i in i $(i,i+1)$ lon o R, n w n l wi $(i+1,i)$ wi o n in x ion.

Lemma 4. R R' R R'

W o v ion v lo o o ov o on - o o i l ol ion w n p i lo o 1.

Theorem 2. $k \geq 1$, $\left(1-1/k^2\right)^{1/k^2} < p < 1$, $d = k-1$ S M_k S

4.3 Optimal Algorithm for Workers with Low Reliability

T i ion o l ly i o i l v w n li ili y o wo p i lo o o. T n wo . W in y owin ny v xi i x ion on in i ll o o ow o ll o l o ol n. T i i own y o vin i off: i ow on i o x ion, n o on i li l , n vi v . A y i n i li o l o ol n.

Lemma 5. $k \geq 3$ $0 < p \leq 1/6$ S M_k S S

T l i i ly i li ny o i l v on in i ll o o ow o ll o l o ol n, w n v

$0 < p \leq 1/6$ n $k \geq 3$. T i l ion w i o l v l 1 o k lon o n o i l v . W o o l v l $k+1$ n i ? T n l ovi n in iv n l i ion lon p i ll. T y o v ion iv i o oo i on l v l $b+k-1$ n i on i li l o o on i ion (b,k) wo l w n in l in v .

Lemma 6. $k \geq 3$ $0 < p \leq 1/(\ k)$ $\leq b \leq k-1$ S M_k

( ) S $b-1$ (b,k)

( ) S $b-1$ (k,b)

S

iv n M_k, w ll v R n i i i o o o i . S i on in only o o ow n i o ol n, o only o l o ol n n o o ow. T ollowin o o l ly i o o i l v S o ll no wo li ili y. W n ov o in o v ion v lo o .

Theorem 3. $k \geq 3$ $0 < p \leq 1/(\ k)$ $d = k-1$ S M_k S

5 Related Work

T o in o i l o i i ion o l in o in i i l o wo nown o l . In N wo li ili y P o l ([5] o l ND 0) w iv n w e il o ili y $p(e)$, V' o v i , n n q. T o l i o i i o ili y i l q, o i o v i in V' i onn in v i , no il . o l i i il , w i ll n o v n x o ly (w i l in o xi n o wi no il o no o ll n o). How v , o o i i ion o l i iff n , w o n n o no o w i ll n o v no il . In iff n o l , ll N wo S viv ili y P o l ([5] o l ND 1), w iv n w n no il o ili y, n n q. T o l i o n o i o ili y i l q o ll $\{u,v\}$, i o on o i n oin no u o v il . T o l i i il w l o on i o ili i il in , ow v in o w on i o lo l n n i , w loo ll n o o no . T ny o n in n lin o l [ ], i o j iv i iff n y y i lly i ini i in n n o no o l o il . A o ili i o l i il o o o l i i y S n [ ] o in n n . T n in . A on i o n . T p wo . n i n , i i n o wo ; o ion i llow , n n n i n n i llow , oo. A o ion f o wo i l y. A l y wo n in o l wi o ili y s, in n n ly o o l . T o l i o o l o o l i “ i l ” no (v l o i ili y o o). A o on i wo i ni : (1) o wo y v i yin i l o i o — i l i wo li ili y n x l l y wo o o ion, in ion o l y wo ov i , ( ) n n ly o n il in n o l — i l in on n in l i o i ili y o wo in l y. A o ow o in ion o wo ni i v n o . l v li in i l ion.

T lin o l i in In n S o in , o n o l i in i . T o o n [19] n o n- n Y wy [1] in o o li o yin o l o lin o o n li i l o llo ion o wo (n o x ion) xi o i l . T i llow on o ili wo w ll, n l o l n li li oo o “ i lo ” n w n o - ion ll o l o li i l . T i n i y o i l l o v l i ni n ili o lly ni o . T o M l wi l. [13] x n i wo vi o olo y o vi in o i l l o o l o o l x . T o in vi o o i ion o i oll ion o i l il in - lo . T in o- i o l o i o o iv n o x o i il in - lo , n io i y l ion on il in - lo . W n il in - lo o i ly in l , n l o i lly. Mo iv y on ion in [13] in nno l o i lly, M l wi n o n [1] o l lin i in w i llo- o wo in io i lly. i li y i lw y o i l wi in i n w wo , i vin i y n il o i i iv ly o l x o - ion. How v , i v ion n olv o i lly in olyno i l i . M l wi l. [15] ow ow o in o o ion in n o n wo il , y o i ly n in o ion o i onn wo .

M l wi [1] in o ll l lin o l w i y li o lin t n i n n i n o x on n n li l wo . Wo i x j o ly wi o ili y $p_{i,j}$. T o l i o n i n Σ, i ow wo i n o (o i ly in ll l n n n ly) o o x ion, o o ini i x o l ion i . T i n n l ll l lin o l i own o NP- w n i o on n wi n l o NPw n i o on n n o wo . T o l xi y l on wi olyno i l i l o i o o l w n o wi n n o wo o on n .

6 Conclusions and Future Work

T i n v lo in lin o y o xi i in x n o o l o x on n li l o , w n v n n i . W in o o in o i lo i i ion o l , ow o l i NP- , n v o i l olyno i l i l o i o i v ion o o l .

y o n v l v n o ollow- . W i i olyno i l i o i l lin l o i ? I on n o - oxi ion l o i o n l o l ? How o ff iv ly l w n o i i own li ili y p_i? How o o i l yn ony, o - i l yn ony, (w n y v io o n o i o o) ff lin i ion ? n o l on i iff n o i i ion o l o x- i i in x n o o ly o in (in o on in , w n xi i in li li oo in will o ly o -). Un li ili y o o o l o l in iff n w y n o ili i lly. W o l o in n f o will in o ly x . W i w i o l x on li l o , w il n v y i w i o o f will x - in o ly. W i o l x on li l o , o o xi i wo - (i. ., low) n o o l o ?

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of Obfuscating Programs. (CRYPTO) (2001) 1–18
2. Crescenzi, P., Kann, V. (eds.): A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/wwwcompendium/node173.html
3. Du, W., Jia, J., Mangal, M., Murugesan, M.: Uncheatable Grid Computing. 24th International Conference on Distributed Computing Systems (ICDCS) (2004)
4. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition. Morgan Kaufmann (2004)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)
6. Golle, P., Mironov, I.: Uncheatable Distributed Computations. RSA Conference – topics in Cryptography (2001) 425–440
7. Golle, P., Stubblebine, S.: Secure Distributed Computing in a Commercial Environment. 5th International Conference Financial Cryptography (FC) (2001) 289–304
8. The Intel Philanthropic Peer-to-Peer program. http://www.intel.com/cure
9. Kahney, L.: Cheaters Bow to Peer Pressure. Wired News, February 15 (2001) http://www.wired.com/news/technology/0,1282,41838,00.html

10. Kondo, D., Casanova, H., Wing, E., Berman, F.: Models and Scheduling Mechanisms for Global Computing Applications. 16th IEEE International Parallel & Distributed Processing Symposium (2002)
11. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M.: SETI@home - massively distributed computing for seti. Computing in Science & Enginering, Vol. 3(1) (2001) 78–83
12. Malewicz, G.: Parallel Scheduling of Complex Dags under Uncertainty. (2005) submitted for publication
13. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: On Scheduling Complex Dags for Internet-Based Computing. 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS) (2005) to appear
14. Malewicz, G., Rosenberg, A.L.: On batch-scheduling dags for Internet-based computing. Typescript, University of Massachusetts (2004) submitted for publication
15. Malewicz, G., Russell, A., Shvartsman, A.: Distributed Cooperation During the Absence of Communication. 14th International Symposium on Distributed Computing (DISC) (2000) 119–133
16. The Olson Laboratory Fight AIDS@Home project. http://www.fightaidsat home.org
17. Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Applied Mathematics, Vol. 131(3) (2003) 651–654
18. Rosenberg, A.L., Yurkewych, M.: Optimal Schedules for Some Common Computation-Dags on the Internet. IEEE Transactions on Computers (2005) to appear
19. Rosenberg, A.L.: On Scheduling Mesh-Structured Computations on the Internet. IEEE Transactions on Computers, Vol. 53(9) (2004)
20. Rosenberg, A.L.: Accountable Web-computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 14(2) (2003) 97–106
21. The RSA Factoring By Web project. `http://www.npac.syr.edu/factoring`
22. Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems, Vol. 18(4) (2002) 561–572
23. SETI@home: Current Total Statistics. `http://setiathome.ssl.berkeley.edu/totals.html`, May 9 (2004)
24. Sun, X.H., Wu, M.: GHS: A performance Prediction and Task Scheduling System for Grid Computing. 17th IEEE International Parallel & Distributed Processing Symposium (2003)
25. Szajda, D., Lawson, B., Owen, J.: Hardening Functions for Large Scale Distributed Computations. IEEE Symposium on Security and Privacy, (2003) 216–224

Author Index