Vincent Gramoli
Rachid Guerraoui (Eds.)

LNCS 7853

Networked Systems

First International Conference, NETYS 2013
Marrakech, Morocco, May 2013
Revised Selected Papers

Lecture Notes in Computer Science 7853

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

Vincent Gramoli Rachid Guerraoui (Eds.)

Networked Systems

First International Conference, NETYS 2013
Marrakech, Morocco, May 2-4, 2013
Revised Selected Papers

Springer

Volume Editors

Vincent Gramoli
NICTA and University of Sydney, NSW, Australia
E-mail: vincent.gramoli@sydney.edu.au

Rachid Guerraoui
EPFL, Lausanne, Switzerland
E-mail: rachid.guerraoui@epfl.ch

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-40147-3 e-ISBN 978-3-642-40148-0
DOI 10.1007/978-3-642-40148-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013944639

CR Subject Classification (1998): C.2, F.2, E.5, K.4

LNCS Sublibrary: SL 5 – Computer Communication Networks and Telecommunications

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

NETYS, the International Conference on Networked Systems, provides a forum to report on best practices and novel algorithms, results and techniques in networked systems. It brings together researchers and engineers from both the theory and practice of networked systems: multi-core architectures, middleware environments, storage clusters, as well as social, peer-to-peer, sensor, wireless, and mobile networks.

This volume contains the papers selected for NETYS 2013, the First International Conference on Networked Systems, held in Marrakech, Marocco. There were 74 papers submitted to the conference among which the Program Committee selected 17 contributions for regular presentations (23%). Each regular presentation is accompanied by a 15-page paper in this volume. Every submitted paper was read and carefully evaluated by the Program Committee in 198 reviews with 20 of them written by external reviewers. Revised and expanded versions of several selected papers will be considered for publication in a special issue of the *Computing Journal*.

The program also included three invited lectures by Willy Zwaenepoel (EPFL, Switzerland), Jonathan Ledgard (EPFL, Switzerland), and Mira Mezini (Technische Universität Darmstadt, Germany).

The Best Paper Award was given to Diego Didona, Pascal Felber, Derin Harmanci, Paolo Romano, and Joerg Schenker for "Identifying the Optimal Level of Parallelism in Transactional Memory Systems." The Best Student Paper Award was given to Yahya Benkaouz and Mohammed Erradi for "A Distributed Protocol for Privacy Preserving Aggregation."

The Program Committee also considered 16 papers for short presentations among the papers that generated substantial interest from the members of the committee but that could not be accepted as regular presentations. Each short presentation is accompanied by a five-page paper in this volume and presents ongoing work or recent results. It is expected that these results will appear as full papers in other conference proceedings or journals.

We would like to thank the General Co-chairs, Mohammed Erradi and Bernd Freisleben, for their help, and the numerous sponsors of the event: DAAD, IBM, Technicolor, INRIA, Telefonica, Google, Microsoft Research, Thinline, University Mohammed V, and ENSIAS.

May 2013

Vincent Gramoli
Rachid Guerraoui

Table of Contents

On the Consensus Number of Non-adaptive Perfect Renaming

Armando Castañeda[1] and Michel Raynal[2,3]

[1] Department of Computer Science, Technion, Haifa 32000, Israel
[2] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[3] Institut Universitaire de France

Abstract. In the (n, M)-renaming problem, there are n processes, each with an initial name known only from itself, and these processes have to compute new names from the set $\{1, ..., M\}$, despite asynchrony and any number of process crashes, such that no two processes have the same new name. If $M = n$, the renaming is said to be *perfect*. If M is only on function on the number n of the processes in the system, the renaming is said to be *non-adaptive*. In contrast, if M is on function on the number of processes that actually participate in a given execution, renaming is *adaptive*. The consensus number of a concurrent object is an integer which measures its synchronization power in presence of any number of process crashes.

This paper investigates the consensus number of non-adaptive perfect renaming objects. It shows that, while a non-adaptive perfect renaming object for two processes (ports) has consensus number 2, its consensus number decreases to 1 when the number of processes which can access it increases beyond 2.

Keywords: Common2, Computability power, Concurrent object, Consensus number, Consensus object, Object port, Process failure, Renaming, Task, Wait-freedom.

1 Introduction

Wait-freedom and the consensus number hierarchy. The notions of wait-freedom and a consensus number have been introduced by M. Herlihy [11]. The implementation of a concurrent object is *wait-free* if any invocation of an object operation does terminate if the invoking process does not crash, whatever the behavior of the other processes (which means that these processes can be concurrently accessing the internal representation of the object, can be very slow, or can even crash in the middle of an operation).

A *consensus* object for n processes (i.e., an object with n ports), sometimes denoted n-consensus, provides the processes with a single one-shot operation propose() (one-shot means that a process invokes the operation at most once). A process p_i invokes propose(v_i), where v_i is the value it proposes to the consensus object. This invocation returns it a value v, and we say that the invoking process decides v. The object is defined by the following properties: a decided value is a proposed value (validity), no two different value are decided (agreement), and if a process invokes propose() and does not crash while executing this operation it decides a value (termination).

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 1–12, 2013.

A main result of [11] is the design of an algorithm (called *universal construction*) that builds, from atomic read/write registers and consensus objects, a wait-free implementation of any object defined from a sequential specification. This steers the design of wait-free object implementations back to the implementation of consensus objects. The notion of a *consensus number* of a class of objects $\mathcal{X}$, denoted CN($\mathcal{X}$), has been introduced to that end. It is defined as follows in [11]:

> "The consensus number for $\mathcal{X}$ is the largest n for which $\mathcal{X}$ solves n-process consensus. If no largest n exists, the consensus number is said to be infinite."

This means that, in a read/write system of n processes, enriched with objects whose consensus number is greater than or equal to n, it is possible to wait-free implement any object defined by a sequential specification.

It is shown in [11] that consensus numbers define an infinite hierarchy: CN(read/write registers) = 1, CN(test&set) = CN(queue) = 2, ..., CN(compare&swap) = CN(LL/SC) = $+\infty$.

The renaming problem. An M*-renaming* [3] object for n processes (i.e., an object with n ports), sometimes denoted (n, M)-renaming, provides the processes with a one-shot operation, denoted rename(), that allows them to obtain distinct names from a name space whose size is M. It is assumed that processes invoke rename() with distinct inputs (their initial names) taken from an input space of size $N > M$.

It has been shown that (n, M)-renaming objects can be wait-free implemented using only read/write register if and only if $M \geq 2n - 1$ [4,12], except for some specific values of n for which (n, M)-renaming objects can be implemented with $M = 2n - 2$ [5]. If follows that for $M \geq 2n - 1$ (or $M \geq 2n - 2$ for the specific values of n) the consensus number of (n, M)-renaming is the same as the one read/write registers, namely, it is 1.

Let us observe that a renaming object can output large names (as long as they belong to the constrained output space) in executions in which few processes invoke the object. For example, it can be the case that a single process invokes an (n, M)-renaming object and receives M, the largest name of the output space. That is to say, the output space does not "adapt" to the number of processes that actually invoke a renaming object, it does depend only on the total number of processes. Thus, if the size of the new name space depends only on n, renaming is said to be *non-adaptive*, denoted NA_renaming.

The adaptive version of renaming is defined as follows. Let p, $1 \leq p \leq n$, denote the number of processes that invoke the operation rename(). *Adaptive* renaming, denoted A_renaming, requires that the size of the new name space be a function of p. Several algorithms implementing $(n, 2p - 1)$-A_renaming objects have been proposed ([6] surveys some of them). $M = 2p - 1$ is the lower bound for the size of new name space, for every value of n.

It is known that, for any n, (n, p)-A_renaming, also called *adaptive perfect renaming*, can be solved from n-port test&set objects (i.e., objects which can be accessed by n processes) and vice-versa, and consequently adaptive perfect renaming has consensus number 2 [6].

The family Common2. A noteworthy family of objects whose consensus number is 2 has been identified in [1,2]. This family, called *Common2*, includes all the objects that (a) have consensus number 2 and (b), for any $n \geq 2$, are wait-free implementable from any number of objects of a class $\mathcal{X}$ such that CN($\mathcal{X}$) = 2.

The equivalence between adaptive perfect renaming and test&set [6] shows that adaptive perfect renaming belongs to the family *Common2*. It has been shown that test&set objects, swap objects, fetch&add objects, A_renaming objects and stacks belong to *Common2*. Differently, while a queue object has consensus number 2, it is still an open problem to known if it belongs to *Common2* [1].

Content of the paper. This paper is on the consensus number of (n, n)-NA_renaming, also called *non-adaptive perfect renaming*. First, the paper shows that it is possible to solve 2-process consensus using $(2, 2)$-NA_renaming objects, while it is not possible to solve 2-process consensus from (m, m)-NA_renaming objects, whenever $m \geq 3$. Then, in the light of these results, the paper analyses what is the consensus number of non-adaptive perfect renaming, according to the classical definition of a consensus number. Then, the paper proposes a refinement of the definition of a consensus number that better reflects the computability power of non-adaptive perfect renaming. Finally, it discusses the relation of non-adaptive perfect renaming with *Common2*.

2 Computation Model

As the computation model is widely used in the literature, we do not explain it in detail (see [6] for a detailed description). We restate only the aspects of this model which are the most important to make the paper self-contained.

Read/write wait-free system model. This paper considers the usual asynchronous, wait-free shared memory system where at most $n - 1$ out of n processes p_1, ..., p_n can fail by crashing. Processes communicate by accessing single-writer/multi-reader atomic registers. The subscript i is called the *index* of p_i. The *participating* processes in a run are the processes that take at least one step in that run. Those that take a finite number of steps are *faulty* (sometimes called *crashed*), the others are *correct* (or *non-faulty*). A non-participating process is a faulty process. The algorithms designed for this computation model have to work despite up to $n - 1$ faulty process. In some sections, in addition to registers, processes cooperate through objects that implement some tasks.

Identities. Each process p_i has an identity denoted id_i. An identity is an integer value in $[1..N]$, where $N \geq 2n - 1$. We assume that in every initial configuration of the system, the identities are distinct: $i \neq j \Rightarrow id_i \neq id_j$. A process knows n but does not know the identity of the other processes.

Index-independent algorithm. Generally speaking, in an index-independent algorithm, indexes are used only for addressing purposes, namely, when a process p_i writes a value to an array of 1WnR registers A, its index is used to deposit the value in $A[i]$, and when p_i reads A, it gets back a vector of n values, where the j-th entry of the vector is associated with p_j; however, the processes cannot use indexes for computation.

Formally, an algorithm $\mathcal{A}$ is *index-independent* if the following holds for every run r and every permutation $\pi()$ of the process indexes. Let r_π be the run obtained from r by permuting the input values according to $\pi()$ and, for each step, the index i of the process that executes the step is replaced by $\pi(i)$. Then r_π is a run of $\mathcal{A}$. Consider a permutation $\pi()$ such that $\pi(i) = j$. The index-independence ensures that p_j behaves in r_π exactly as p_i behaves in r: it decides the same thing in the same step. In an index-independent algorithm, if the output of p_i in r is v, then the output of $p_{\pi(i)}$ in r_π is v, i.e., the output of a process does not depend on indexes, it depends only on the inputs (ids) and on the interleaving.

Additional communication objects. In addition to read/write registers, processes may be allowed to communicate through one-shot (m, m)-NA_renaming objects, $m \geq 2$. Hence, each process can invoke a renaming object at most once. Each of these objects $\mathcal{X}$ is assumed to be *sequentially deterministic* [8]. This means that $\mathcal{X}$ behaves deterministically in every non-concurrent invocation by a single process, namely, the output of $\mathcal{X}$ in a non-concurrent invocation only depends on its internal state (just before the invocation) and the input. In other words, the only source of non-determinism is concurrency.

An an example, let us consider an execution in which $\mathcal{X}$ is invoked first by p, then q and r invoke it concurrently (after p's invocation has finished) and finally s invokes $\mathcal{X}$ alone. Then, $\mathcal{X}$ behaves deterministically in the non-concurrent invocations of p: the output $\mathcal{X}$ produces only depends on its initial internal state and p's input. In contrast, $\mathcal{X}$ behaves non-deterministically in the concurrent invocations of q and r: the behavior of $\mathcal{X}$ depends not only on its current state (after p's invocation) and the inputs, but also on the interleaving of the internal computation steps of $\mathcal{X}$ during the invocation. Finally, $\mathcal{X}$ behaves deterministically in the last invocation.

Note that (n, M)-NA_renaming is trivial if there is no restriction on how a process can use its index: each process p_i just decides i. Therefore, in order to avoid trivial solutions, it is required that any implementation of an (n, M)-NA_renaming object must be *index-independent*, namely, the new name of a process is by no means a function of its index, it is a function of only its input name and the execution.

3 Solving 2-Process Consensus from Non-adaptive Perfect Renaming

k-set agreement [9] is a generalization of consensus in which at most k distinct values can be decided (thus 1-set agreement is consensus). The computability power of M-renaming and k-set agreement [9] is compared in [7]. A consequence of these results is that:

1. It is possible to solve 2-process consensus from $(2, 2)$-NA_renaming objects.
2. It not possible to solve 2-process consensus from (m, m)-NA_renaming objects where $m \geq 3$.

This section presents a simple direct proof of these two results (instead of proving them using the results in [7]). This proof is based on a sequentially deterministic (m, m)-NA_renaming object $\mathcal{X}$.

Theorem 1. *There is an asynchronous wait-free implementation of a 2-process consensus object from read/write registers and* $(2, 2)$*-NA_renaming objects.*

Proof. Let $\mathcal{X}$ be a $(2, 2)$-NA_renaming object. A *solo-execution* of $\mathcal{X}$ is an execution in which exactly one process invokes $\mathcal{X}$. The fact that $\mathcal{X}$ is sequentially deterministic implies that there must be a value $x_{winner} \in \{1, 2\}$ such that in every solo-execution with input 1, no matter the participating process p, $\mathcal{X}$ outputs x_{winner} to p. Note that this is true because $\mathcal{X}$ is index-independent, and hence the output of $\mathcal{X}$ cannot depend on the index of p.

```
function propose(v_i) is
(01)  M[i] ← v_i   % Each entry of M is initialized to ⊥
(02)  j ← (i + 1) mod 2   % id of the other process
(03)  if M[j] = ⊥
(04)     then ℓ_i ← X.rename(1)
(05)     else if i < j
(06)             then ℓ_i ← X.rename(2)
(07)             else ℓ_i ← X.rename(3)
(08)          end if
(09)  end if
(10)  if ℓ_i = x_winner
(11)     then return(M[i])
(12)     else return(M[j])
(13)  end if
```

Fig. 1. Solving consensus among two processes from $(2, 2)$-NA_renaming (code for P_i)

Fig. 1 depicts an implementation of a consensus object for two processes, P_0 and P_1, using $\mathcal{X}$ and x_{winner}. Each process P_i first announces its proposal in M (line 01), and then checks if (according with its view) the other process, P_j, participates in the execution (line 03). Then, the processes dynamically compute the values used for invoking $\mathcal{X}$ (recall that processes call an (n, M)-NA_renaming object with distinct inputs in the space $[1, \ldots, N]$, $N > M$). If P_j does not participate ($M[j] = \perp$), then P_i invokes $\mathcal{X}$ with input 1 (line 04); if P_j participates, then P_i calls $\mathcal{X}$ with either 2 or 3, depending if i (its id) is smaller than j (lines 05-08). Finally, P_i decides its proposal if it gets x_{winner} from $\mathcal{X}$, otherwise decides the proposal of P_j (lines 09-12).

The termination property of consensus directly follows from the code and from the fact that $\mathcal{X}$ is wait-free. We now check that the validity and agreement properties also hold. Consider first a solo-execution of the algorithm. Then, the unique participating process P_i invokes $\mathcal{X}$ with input 1 (line 04). As already explained, in such execution, P_i gets x_{winner} from $\mathcal{X}$, and thus it decides its proposal v_i (line 10). Now, consider an execution in which both processes, P_0 and P_1, participate. Note that it is not possible that both P_0 and P_1 see that the condition in line 03 is true (however, it is possible that both see that the condition is not true, or one processes sees that it is true, while it is not true for the other process). Therefore, at most one process invokes $\mathcal{X}$ with input 1. If

P_0 and P_1 see that the condition is true, then one of them invoke $\mathcal{X}$ with input 2, while the other invokes it with input 3. In any case, P_0 and P_1 invoke $\mathcal{X}$ with distinct input vales. Since $\mathcal{X}$ solves $(2,2)$-NA_renaming, it outputs x_{winner} to exactly one process, which decides its proposal; the other process decides the proposal of the process that got x_{winner}. The theorem follows. $\square_{Theorem\ 1}$

Theorem 2. *For every $m \geq 3$, there is no asynchronous wait-free implementation of a 2-process consensus object from read/write registers and (m,m)-NA_renaming objects.*

Proof. Assume, for the sake of contradiction, that there exists such an object $\mathcal{A}$ that implements consensus among 2 processes, P_0 and P_1, from read/write registers and (m,m)-NA_renaming objects. The idea of the proof is simple, namely, a read/write based object $\mathcal{B}$ is obtained by replacing each (m,m)-NA_renaming object in $\mathcal{A}$ with a read/write wait-free object that solves $(2,3)$-NA_renaming; then we will see that in every execution of $\mathcal{B}$, P_0 and P_1 reach consensus, thus consensus is read/write wait-free solvable, which is a contradiction.

Fig. 2 contains an asynchronous wait-free implementation of a $(2,3)$-NA_renaming object for two processes, P_0 and P_1. It is essentially the same mechanism used in Fig. 1 to decide the input a process uses for invoking $\mathcal{X}$. Clearly, the implementation is wait-free. Also, it is is index-independent since the output of a process only depends on if it is running alone, or if its input name is smaller than the input name of the other process. Also, observe that it is not possible that P_0 and P_1see that the condition in line 03 is true, and thus at most one process decides the output name 1. If P_0 and P_1 see that the condition is true, then the process with smaller input name decides 2, while the other process decides 3. In any case, P_0 and P_1 decide distinct output names in the range $[1,2,3]$. Therefore, the algorithm in Fig. 1 is a correct implementation of $(2,3)$-NA_renaming. Moreover, any instance of the algorithm clearly is sequentially deterministic.

```
function rename(v_i) is
(01)  M[i] ← v_i;   % Each entry of M is initialized to ⊥
(02)  j ← (i + 1) mod 2;   % id of the other process
(03)  if M[j] = ⊥
(04)     then return(1)
(05)     else if min(M) = v_i
(06)             then return(2)
(07)             else return(3)
(08)          end if
(09)  end if.
```

Fig. 2. A read/write wait-free algorithm that solves $(2,3)$-NA_renaming (code for P_i)

From $\mathcal{A}$, we obtain an algorithm $\mathcal{B}$ for two processes P_0 and P_1: each (m,m)-NA_renaming object in $\mathcal{A}$ is replaced with a distinct instance of the algorithm in Fig. 2. Thus, in $\mathcal{B}$, each P_i executes exactly the same instructions as in $\mathcal{A}$, except that each

time it invokes an (m, m)-NA_renaming object, it actually invokes a read/write wait-free object solving $(2, 3)$-NA_renaming. Note that the resulting algorithm $\mathcal{B}$ uses only read/write operations.

The key observation is that in every execution of $\mathcal{B}$, P_0 and P_1 cannot distinguish that they are not calling genuine objects that solve (m, m)-NA_renaming: each P_i receives a unique value in the range [1,2,3] which is correct because $m \geq 3$, and thus each each (m, m)-NA_renaming object in $\mathcal{A}$ can output (valid) values in the range [1,2,3] whenever invoked. Also, each sequentially deterministic (m, m)-NA_renaming object is replaced with a sequentially deterministic $(2, 3)$-NA_renaming object.

Therefore, for every execution of $\mathcal{B}$, there is an execution of $\mathcal{A}$ in which P_0 and P_1 behave exactly in the same way. This implies the following:

- There cannot be an execution of $\mathcal{B}$ in which a correct process that executes infinitely many computation steps and does not decide because this would imply that there is an execution of $\mathcal{A}$ in which there is a correct process that executes infinitely many computation steps and does not decide, contradicting that $\mathcal{A}$ is wait-free.
- There cannot be an execution of $\mathcal{B}$ in which correct processes decide on distinct values because this would imply that there is an execution of $\mathcal{A}$ in which correct process decide on distinct values, contradicting that $\mathcal{A}$ a solves consensus.

Therefore, we conclude that $\mathcal{B}$ is asynchronous read/write wait-free implementation of a 2-process consensus object. A contradiction. $\square_{Theorem\ 2}$

4 The Consensus Number of Non-adaptive Perfect Renaming

4.1 Refining the Consensus Number Definition: Definition A

The base consensus number definition does not explicitly consider the number of *ports* of an object of the type $\mathcal{X}$ (i.e., how many different processes are allowed to access the object). Hence, what do we mean when we say that test&set, for example, has consensus number 2? To the best of our knowledge, the literature interprets the base consensus number definition as considering that every m-port test&set object, $m \geq 2$, can solve consensus among 2 processes. More precisely, this implicit interpreation can be captured by the following definition (definition A).

> The consensus number for $\mathcal{X}$ (denoted $\text{CN}_A(\mathcal{X})$) is the largest n such that there is an algorithm that wait-free solves n-process consensus from read/write registers and m-port objects of type $\mathcal{X}$, for every $m \geq n$. If no largest n exists, the consensus number is said to be infinite.

This definition seems to fit the intuition of [11]. Basically, it places no constraint on the number of processes that can a priori access objects of the type $\mathcal{X}$ (as it places no constraint on the number of processes that can access atomic read/write registers).

Thus, under Definition A, non-adaptive perfect renaming has consensus number 1. This is because, for $n \geq 3$, (n, n)-NA_renaming objects cannot solve 2-process consensus, Theorem 2.

4.2 Refining the Consensus Number Definition: Definition B

As we have already seen, (a) it not possible to solve 2-process consensus from (m, m)-NA_renaming objects where $m \geq 3$, Theorem 1, but (b) it is possible to solve 2-process consensus from $(2, 2)$-NA_renaming objects, Theorem 2. This means that, in a system of $n = 2$ processes enriched with $(2, 2)$-NA_renaming objects (two-port objects), Definition A becomes inconsistent.

So, what is the consensus number of non-adaptive perfect renaming? Hence the following weaker Definition B, which is an alternate interpretation of the original definition of consensus numbers [11]. Under this definition the consensus number of $(2, 2)$-NA_renaming is 2.

> The consensus number for $\mathcal{X}$ (denoted $\text{CN}_B(\mathcal{X})$), is the largest n such that there is an algorithm that wait-free solves n-process consensus from read/write registers and n-port objects of type $\mathcal{X}$. If no largest n exists, the consensus number is said to be infinite.

As we can see, "m-port objects of type $\mathcal{X}$ for every $m \geq n$" of Definition A is replaced here by "m-port objects of type $\mathcal{X}$ where $m = n$". Note that for every class of objects $\mathcal{X}$, $\text{CN}_B(\mathcal{X}) \geq \text{CN}_A(\mathcal{X})$.

Definition B is in agreement with the fact that $(2, 2)$-NA_renaming has consensus number 2 and (m, m)-NA_renaming has consensus number 1 for $m > 2$. This means that there are objects whose instances have a consensus number which depends on the number of processes (ports) m for which they are instantiated.

This seems deeply related to the following observation. As an (m, m)-NA_renaming object is non-adaptive, it follows that, when $m > 2$ and only two processes invoke this object, they can obtain new names greater than 2, namely, the object does not behave as a $(2, 2)$-NA_renaming object, when only two processes participate.

5 $(2, 2)$-NA_Renaming Belongs to *Common2*

This section shows that $(2, 2)$-NA_renaming belongs to *Common2*. The proof follows from the combination of three lemmas [7,6,10].

Theorem 3. *Both* (m, p)-A_renaming *and* $(2, 2)$-NA_renaming *belong to Common2.*

First, recall that a *test&set* object for n processes, denoted n-T&S, provides the processes with a single one-shot operation participate(). This operation allow processes to obtain either 0 (*winner*) or 1 (*looser*) in a way that exactly one process gets 0, in every execution. Let A $\simeq$ B means that A and B have the same computational power, namely, A implements B and B implements A.

Proof. The theorem follows directly from the lemmas below: by Lemmas 1 and 2, it follows that, for every $m \geq 2$, (m, p)-A_renaming $\simeq$ 2-T&S; thus, for every $m \geq 2$, $(2, 2)$-NA_renaming $\simeq$ (m, p)-A_renaming $\simeq$ 2-T&S $\simeq$ m-T&S, from Lemma 3. Therefore, (m, p)-A_renaming and $(2, 2)$-NA_renaming belong to *Common2*. $\square_{Theorem\ 3}$

For completeness, the proofs of Lemmas 1, 2 and 3 are presented below.

Lemma 1. [6] $\forall\ m \geq 2$*:* m-T&S $\simeq (m,p)$-A_renaming.

Proof. It is easy to implement m-T&S from (m,p)-A_renaming: each p_i invokes an (m,p)-A_renaming object $\mathcal{X}$ using i as input, and decides 0 (*winner*) if gets 1, otherwise decides 1 (*looser*). Due to the specification of (m,p)-A_renaming, in every execution in which p processes participate, $\mathcal{X}$ outputs distinct names in the range $[1,\ldots,p]$. Thus, exactly one process gets 1 from $\mathcal{X}$, and consequently decides 0; the other process receive values distinct from 1, and hence decide 1.

In an implementation of (m,p)-A_renaming from m-T&S, the processes share an n-dimensional array of m-T&S objects. Each p_i invokes in order the objects in the array; if it gets 0 from the j-th object, then it decides name j (and does not invoke the other objects). Since a m-T&S object outputs 0 to exactly one process, at most one process decides j. Also, if exactly p processes participate, in the worst case, a process invoke the p-th object and then decides. $\square_{Lemma\ 1}$

Lemma 2. [10] $\forall\ m \geq 2$*:* m-T&S $\simeq$ 2-T&S.

Proof. As $m \geq 2$, building a 2-TS object from an m-TS object is trivial. So, the interesting construction is the one in the other direction. The corresponding construction is simple. It is based on the following two principles. Let $\mathsf{participate}_2()$ denote the operation on a base 2-TS object, and $\mathsf{participate}_\mathsf{m}()$ denote the operation on the constructed m-TS object.

First, in order to satisfy the index independence property, the transformation first uses an underlying renaming object that provides the processes with new names that they can thereafter use "instead of" their indexes. Renaming algorithms that satisfy the index independence property and use only atomic registers do exist (e.g., see [6]). These algorithms provide a new renaming space whose maximal size is $M = 2m-1$. So, the new name of a process p_i is an integer in the set $\{1,\ldots,2m-1\}$ that is independent of its index i. This underlying base renaming object is denoted $BASE\_AR$.

Let $nbc = C(2m-1,2)$ (the number of distinct subsets of two elements in a set of $2m-1$ elements). Let us order these nbc subsets in an array $SET\_LIST[1..nbc]$ in such a way that $SET\_LIST[x]$ is the two-process set that define the xth subset. Moreover, let $BASE\_TS[1..nbc]$ be an array of nbc base 2-T&S objects.

The principle that underlies the second part of the construction is the following. First, the two processes that define $SET\_LIST[x]$ are the only ones that can access $BASE\_TS[x]$. When a process p_i invokes $\mathsf{participate}_2()$, starting from the first base object $BASE\_TS[x]$ it belongs to, it scans (one after the other and in the increasing order on their indexes) all the sets $BASE\_TS[x]$ it belongs to. If $BASE\_TS[x].\mathsf{participate}_2()$ returns 1 (loser), p_i stops scanning and returns 1 as the result of its invocation $\mathsf{participate}()$. Otherwise, p_i is a winner among the processes that access $BASE\_TS[x]$; it then proceeds to the next object of $BASE\_TS[1..nbc]$ to which it belongs. If there is no such object (p_i has then "successfully" scanned all the subsets it belongs to), it returns 1 as the result of $\mathsf{participate}_\mathsf{m}()$.

The construction is described in Fig. 3. The local variable pos_i keeps p_i's current scanning position in $SET\_LIST[1..nbc]$. The function $\mathsf{next}(new\_name_i, pos_i)$ returns the first entry y (starting from pos_i+1 and in increasing order) of $SET\_LIST[1..a]$ such

```
operation participate_m():
(01)  new_name_i ← BASE_AR.rename();
(02)  pos_i ← 0;
(03)  while (true) do
(04)    pos_i ← next(new_name_i, pos_i);
(05)    res_i ← BASE_TS[pos_i].participate_2();
(06)    if (res_i = 0) then return (1)
(07)                   else if (pos_i = last(new_name_i))
(08)                                then return (0) end if
(09)    end if
(10)  end while
```

Fig. 3. From 2-TS objects to an m-TS object (code for p_i)

that new_name_i belongs to $SET\_LIST[y]$. Finally, the predicate $\mathsf{last}(new\_name_i)$ returns *true* iff $pos_i = nbc$ or new_name_i belongs to no set from $SET\_LIST[pos_i+1]$ until $SET\_LIST[a]$. The statement $\mathsf{return}(v)$ terminates p_i's invocation.

It trivially follows from lines 06 and 07 that only the values 0 (winner) or 1 (loser) can be returned. Let us now show that the invocation of $\mathsf{participate_m}()$ by a correct process p_i terminates. If p_i executes $\mathsf{return}(0)$ at line 06, it terminates. So, let us assume that p_i never executes $\mathsf{return}(0)$ at line 06. It follows that it is a winner in each base object $BASE\_TS[y]$ it accesses. These objects define a list that has a last element $BASE\_TS[z]$. When p_i accesses that base object, we have both $pos_i = z$ and $\mathsf{last}(new\_name_i) = z$, from which it follows that p_i executes $\mathsf{return}(1)$ at line 07.

The proof that exactly one process that invokes $\mathsf{participate_m}()$ is a winner is decomposed in two steps.

- At most one process is a winner. The proof is by contradiction. Let us assume that 2 processes are winners. Let S be the set of the new names of these 2 processes. There is a set $SET\_LIST[y]$ such that $SET\_LIST[y] = S$. Due to the code of the construction, a process p_j that is a winner (with respect to the constructed object) has to be a winner in all the base 2-TS objects $BASE\_TS[x]$ such that $new\_name_j \in SET\_LIST[x]$. It follows from that observation that the two processes of S invoke $BASE\_TS[y].\mathsf{participate_2}()$ and obtain 1 from that base object, which is impossible.
- At least one process is a winner. Let $BASE\_TS[y]$ be the "last" 2-TS base object accessed during a run ("last" means here that this base object is the one with the largest index y that is accessed during a run). Due to the property of the base objects, there is at least one process p_j that is a winner with respect to the base object $BASE\_TS[y]$. As p_j does not access other 2-TS objects, it follows that it returns the value 1 as the result of its invocation of $\mathsf{participate_m}()$. $\square_{Lemma\ 2}$

Lemma 3. [7] 2-T&S $\simeq$ $(2,2)$-NA_renaming.

Proof. It is trivial to implement $(2,2)$-NA_renaming from 2-T&S: the processes invoke a 2-T&S object; if a process gets 0, decides 1, otherwise decides 2.

```
function participate() is
(01)  M[i] ← i    % Each entry of M is initialized to ⊥
(02)  j ← (i + 1) mod 2    % id of the other process
(03)  if M[j] = ⊥
(04)     then ℓ_i ← 𝒳.rename(1)
(05)     else if i < j
(06)             then ℓ_i ← 𝒳.rename(2)
(07)             else ℓ_i ← 𝒳.rename(3)
(08)          end if
(09)  end if
(10)  if ℓ_i = x_winner
(11)     then return(0)
(12)     else return(1)
(13)  end if
```

Fig. 4. Solving 2-T&S from $(2, 2)$-NA_renaming (code for P_i)

We now show how to implement 2-T&S from $(2, 2)$-NA_renaming. Let $\mathcal{X}$ be a $(2, 2)$-NA_renaming object. As explained in the proof on Theorem 1, there is a value $x_{winner} \in \{1, 2\}$ such that in every solo-execution of $\mathcal{X}$ with input 1, no matter the participating process, $\mathcal{X}$ outputs x_{winner}. Fig. 4 shows an implementation of 2-T&S from $\mathcal{X}$ and x_{winner}. This implementation is essentially the same as the in Fig. 1 and its correctness proofs is left to the reader. $\square_{Lemma\ 3}$

6 Conclusion

To conclude. Assuming one-shot multi-port objects where each process is attached to a specific port, this paper has shown that the consensus number of non-adaptive perfect renaming object depends on the number of distinct processes allowed to access it (number of ports of the object). It has shown that the consensus number of such an object is either 1 or 2. It is 2 if and only if the non-adaptive perfect renaming object can be accessed exactly by 2 processes (i.e., it has two ports). This result, shows that increasing the number of ports of an object does not necessarily increase its computability power in asynchronous systems where any number of process may crash. The paper has also shown that the $(2, 2)$-NA_renaming object is a member of the class of synchronization objects called *Common2*.

When the objects are multi-shot objects. The paper focused on a model in which each process can invoke an object at most once (one-shot object). Namely, every object has a given number of ports, one for each process (i.e., a porcess cna only acces the object through the port assigned to it). What does it happen when a process can access several ports? In that model of computation, (m, m)-NA_renaming can solve two-process consensus, for any $m \geq 2$. Given an (m, m)-NA_renaming object $\mathcal{X}$, each process p_i, $i = 0, 1$, invokes half of the ports of $\mathcal{X}$, in some order. If p_i gets (from some port) a

value $\mathcal{X}$ outputs in a solo-execution, then it decides its proposal, otherwise it decides the proposal of the other process. In this model of computation it would be interesting to explore the computability power of (m, m)-NA_renaming and (m', m')-NA_renaming, when $m \neq m'$.

Acknowledgments. The work of the first author has been partially supported at the Technion by an Aly Kaufman Fellowship. The work of the second author has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing. The authors want to thank the referees for their constructive comments.

References

1. Afek, Y., Gafni, E., Morisson, A.: Common2 Extended to Stacks and Unboudned Concurrency. Distributed Computing 20, 239–252 (2007)
2. Afek, Y., Weisberger, E., Weisman, H.: A Completeness Theorem for a Class of Synchronization Objects (Extended Abstract). In: Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC 1993), pp. 159–170 (1993)
3. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuck, R.: Renaming in an Asynchronous Environment. Journal of the ACM 37(3), 524–548 (1990)
4. Castañeda, A., Rajsbaum, S.: New Combinatorial Topology Upper and Lower Bounds for Renaming: The Lower Bound. Distributed Computing 22(5-6), 287–301 (2010)
5. Castañeda, A., Rajsbaum, S.: New Combinatorial Topology Upper and Lower Bounds for Renaming: The Upper Bound. Journal of the ACM 59(1), 3 (2012)
6. Castañeda, A., Rajsbaum, S., Raynal, M.: The Renaming Problem in Shared Memory Systems: an Introduction. Computer Science Review 5(3), 229–251 (2011)
7. Castañeda, A., Imbs, D., Rajsbaum, S., Raynal, M.: Renaming is Weaker than Set Agreement but for Perfect Renaming: A Map of Sub-Consensus Tasks. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 145–156. Springer, Heidelberg (2012)
8. Castañeda, A., Rajsbaum, S., Raynal, M.: There are plenty of tasks weaker than perfect renaming and stronger than set agreement (Brief announcement). In: Proc. 31st ACM Symposium on Principles of Distributed Computing (PODC 2012), pp. 97–98 (2012)
9. Chaudhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Information and Computation 105(1), 132–158 (1993)
10. Gafni, E., Raynal, M., Travers, C.: Test&set, Adaptive Renaming and Set Agreement: a Guided Visit to Asynchronous Computability. In: Proc. 26th IEEE Symposium on Reliable Distributed Systems (SRDS 2007), pp. 93–102. IEEE Computer Society Press (2007)
11. Herlihy, M.: Wait-Free Synchronization. ACM Transactions Programminf Languages and Systems 13(1), 124–149 (1991)
12. Herlihy, M.P., Shavit, N.: The Topological Structure of Asynchronous Computability. Journal of the ACM 46(6), 858–923 (1999)

Set Agreement and the Loneliness Failure Detector in Crash-Recovery Systems*

Sergio Arévalo[1], Ernesto Jiménez[1], and Jian Tang[2]

[1] Universidad Politécnica de Madrid, 28031 Madrid, Spain
{sergio.arevalo,ernes}@eui.upm.es
[2] Distributed System Laboratory (LSD), Universidad Politécnica de Madrid, 28031 Madrid, Spain
tjapply@gmail.com

Abstract. The *set agreement* problem states that from n proposed values at most $n-1$ can be decided. Traditionally, this problem is solved using a failure detector in asynchronous systems where processes may crash but not recover, where processes have different identities, and where all processes initially know the membership. In this paper we study the set agreement problem and the weakest failure detector $\mathcal{L}$ used to solve it in asynchronous message passing systems where processes may crash and recover, with homonyms (i.e., processes may have equal identities) and without a complete initial knowledge of the membership.

1 Introduction

The *k-set agreement* problem [9] guarantees from n proposed values at most k can be decided. Two cases of this problem have received special attention: *consensus* (when $k = 1$), and *set agreement* (when $k = n-1$). The k-set agreement problem that is trivial to solve when the maximum number of processes that may crash (denoted by t) is lesser than k, or the maximum number of different proposed values (denoted by d) is equal or lesser than k (i.e., $t < k$ or $d \leq k$), becomes impossible to solve in an asynchronous system where processes may crash when $t \geq k$ and $d > k$ ([6], [15], [21]). To circumvent this impossibility result, many works can be found in the literature where the asynchronous system is augmented with a failure detector [20] to achieve k-set agreement. A failure detector [7] is a distributed tool that each process can invoke to obtain some information about process failures. There are many classes of failures detectors depending on the quality and type of the returned information ($\diamond\mathcal{P}$, Σ, $\mathcal{FS}^*$, ψ, ...).

A very important issue to solve k-set agreement is to identify the information needed about processes failures. We say that a failure detector class X is the weakest [7] to achieve k-set agreement if the information returned by any failure detector D of this class X is necessary and sufficient to solve k-set agreement.

* This work has been partially funded by the Spanish Research Council (MICCIN) under project TIN2010-19077, by the Madrid Research Foundation (CAM) under project S2009/TIC-1692 (cofunded by ERDF & ESF).

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 13–27, 2013.

In other words, with the failure information output by any failure detector D' of any class Y that solves k-set agreement, a failure detector $D \in X$ can be built on any asynchronous system augmented with a failure detector $D' \in Y$. We say that a class X is strictly weaker than Y (denoted by $X \prec Y$) if a failure detector $D \in X$ can be obtained from a system augmented with any failure detector $D \in Y$, and the opposite is not possible.

In message passing systems, Ω is the weakest failure detector to solve consensus (i.e., 1-set agreement) when a majority of processes do not crash [8], and $\mathcal{L}$ [13] is the weakest failure detector to solve set agreement (i.e., (n-1)-set agreement). For all $2 \leq k \leq n-2$, to find the weakest failure detector to achieve k-set agreement is an open question.

New assumptions have been studied trying to solve k-set agreement in a more realistic message passing systems. In [1] consensus and failures detectors are presented in an extension of the crash-stop model where processes can crash and recover (called crash-recovery model, and by extension, the systems with this failure model are denoted by crash-recovery systems). It is easy to see that these systems are generalizations of systems where processes fail by crashing-stop. A typical definition of a system [7] defines links between processes as reliable (i.e., each sent message is delivered to all alive processes without errors and only once). For the sake of extending traditional system assumptions, consensus and failure detectors are studied when *fair-lossy* links are used [1] (i.e., messages can be lost, but if a process sends permanently a message m to a same alive process, message m is also received permanently).

Sometimes the assumption of knowing the membership in advance is not possible when a run starts (e.g., in a p2p network where servers working as seeds are unknown a priori, and they are possibly different in each run, or even in the same run). This assumption is relevant because, for instance, even though Ω is implementable when the membership is unknown, none of the original eight classes of failures detectors proposed in [7] ($\mathcal{P}$, $\diamond\mathcal{P}$, $\mathcal{S}$, ...) are implementable if each process does not know initially the identity of all processes [17]. Note that any failure detector implementation for a system S with the assumption of unknown membership initially also works in any system S' with the same assumptions except that the membership is known (we say that S is a generalization of S').

Finally, homonymy is a novel assumption included in current systems where privacy is an important issue [12]. Homonymy allows to assign the same identity to more than one process (all processes with the same identity are homonymous). Note that a classical system of n processes with a different identity per process is a particular case of an homonymous system (there are n sets of homomymous processes of size 1). Similarly, anonymity [5] can be considered as a particular case of homonymy (there is a unique set of homomymous processes of size n, or, in other words, all processes are homonymous).

Related Work. As we said previously, new assumptions have been studied trying to solve k-set agreement in a more realistic way. Consensus and failure detectors were presented in asynchronous systems where processes may crash and recover [1]. Besides processes that in a run do not crash (*permanently-up*)

and processes that crash and stop forever (*permanently-down*), new classes of processes may appear in a run of a crash-recovery system: processes that crash and recover several times but after a time are always up (*eventually-up*), processes that crash and recover several times but after a time are always down (*eventually-down*), and processes that are permanently crashing and recovering (*unstable*). In these crash-recovery systems a process is said to be *correct* in a run if it is either permanently-up or eventually-up. On the other hand, an *incorrect* process in a run is either a permanently-down, eventually-down or unstable process. In [1] is proven that consensus with the failure detector $\diamond\mathcal{P}$ [7] is impossible to solve if the number of permanently-up processes in a run can be lesser or equal to the number of incorrect processes. There are in the literature several implementations of consensus and Ω for crash-recovery message passing systems ([1], [16], [18]).

Even though the initial knowledge of the membership is not always possible, different grades of knowledge are also possible. For example, Ω is implementable if each process initially only knows its own identity [17], or if each process also knows n (i.e., the number of processes of the system) [3].

In [2] new classes of failure detectors are introduced to work in homonymous systems. In that paper consensus is also implemented with the counterparts of the weakest failure detectors in classical message passing systems with unique processes' identities: Ω [8] when a majority of processes are correct (its counterpart is called $H\Omega$), and $\langle \Omega, \Sigma \rangle$ [11] when a majority of processes can crash (its counterpart is called $\langle H\Omega, H\Sigma \rangle$).

Regarding set agreement in message passing systems, in the literature we find only two works using the weakest failure detector $\mathcal{L}$ in crash-stop asynchronous systems ([13], [4]). In [13] a total order of process' identifiers and the initial knowledge of the membership is necessary. In [4] set agreement is implemented in systems where the knowledge of n is required.

The failure detector $\mathcal{L}$ is defined and implemented for crash-stop message passing systems in [4] and [19]. $\mathcal{L}$ is a failure detector defined for crash-stop systems in such a way that it always returns the boolean value *false* in some process p_i, and, if there is only one correct process p_j, eventually p_j returns *true* permanently. Nevertheless, in both implementations the algorithms always output *false* in all processes in runs where all processes are correct (i.e., in fail-free runs), which are most frequent in practice. This behaviour is relevant because the complexity of all algorithms that implement set agreement with $\mathcal{L}$ (our algorithm presented in this paper included) is improved if the number of processes that return *true* increases.

Our Work. Trying to generalize the results to the maximum number of systems as possible, this paper is devoted to study set-agreement in message passing systems with the weakest failure detector $\mathcal{L}$ in crash-recovery asynchronous systems with homonyms and without a complete initial knowledge of the membership. In our crash-recovery system model the maximum number of different processes that may crash and recover is so weak ($t = n$) that set-agreement can be solved but consensus can not [1]. An algorithm that implements set-agreement

for crash-recovery systems using $\mathcal{L}$ with homonyms and without initial knowledge of membership is presented in this paper.

We also show in this paper that it is not possible to implement $\mathcal{L}$ even in synchronous crash-recovery systems when $t = n$, or in partially-synchronous crash-recovery systems when $t = n - 1$. We introduce an algorithm that implements $\mathcal{L}$ in synchronous crash-recovery systems when $t = n - 1$. This algorithm works when a subset of processes' identities are known by all processes.

Note that, to our knowledge, both algorithms presented in this paper are the first that implement set agreement and $\mathcal{L}$ in crash-recovery systems. These are also the first algorithms that work with homonyms and without initial knowledge of membership in crash-stop systems.

This paper is organized as follows. The crash-recovery model is presented in Section 2. Definitions of set agreement and failure detector $\mathcal{L}$ are included in Section 3. In Section 4 we have an implementation of set agreement. The implementability of $\mathcal{L}$ is studied in Section 5. An implementation of $\mathcal{L}$ is presented in Section 6. We finish our paper with the conclusions in Section 7.

2 System Model

Processes. The message passing system is formed by a set Π of processes, such that the size n of Π is greater than 1. We use $id(i)$ to denote the identity of the process $p_i \in \Pi$.

Homonymy. There could be homonymous processes [2], that is, different processes can have the same identity. More formally, let ID be the set of different identities of all processes in Π. Then, $1 \leq |ID| \leq n$. So, in this system, $id(i)$ can be equal to $id(j)$ and p_i be different of p_j (we say in this cases that p_i and p_j are homonymous). Note that anonymous processes [5] are a particular case of homonymy where all processes have the same identity, that is, $id(i) = id(j)$, for all p_i and p_j of Π (i.e., $|ID| = 1$).

Unknown Knowledge of Membership. Every process $p_i \in \Pi$ initially knows its own identity $id(i)$, but p_i does not know the identity of any subset of processes, or the size of any subset of Π, different of their trivial values. That is, process p_i only knows initially that $id(i) \in ID$ and $|\Pi| > 1$.

Time. Processes are asynchronous, and, for analysis, let us consider that time advances at discrete steps. We assume a global clock whose values are the positive natural numbers, but processes cannot access it.

Failures. Our system uses basically the failure model of crash-recovery proposed in [1]. In this model processes can fail by crashing (i.e., stop taking steps), but crashed processes may have a *recovery* if they restart their execution (i.e., they may recover). A process is *down* while it is crashed, otherwise it is *up*. Let us define a *run* as the sequence of steps taken by processes while they are up. So, in every run, each process $p_i \in \Pi$ belongs to one of these five classes:

- *Permanently-up*: Process p_i is always alive, i.e., p_i never crashes.

- *Eventually-up*: Process p_i crashes and recovers repeatedly a finite number of times (at least once), but eventually p_i, after a recovery, never crashes again, remaining alive forever.
- *Permanently-down*: Process p_i is alive until it crashes, and it never recovers again.
- *Eventually-down*: Process p_i crashes and recovers repeatedly a finite number of times (at least once), but eventually p_i, after a crash, never recovers again, remaining crashed forever.
- *Unstable*: Process p_i crashes and recovers repeatedly an infinite number of times.

In a run, a permanently-down, eventually-down or unstable process is said to be *incorrect*. On the other hand, a permanently-up or eventually-up process in a run is said to be *correct*. The set of incorrect processes in a run is denoted by $Incorrect \subseteq \Pi$. The set of correct processes in a run is denoted by $Correct \subseteq \Pi$. Hence, $Incorrect \cup Correct = \Pi$.

Unless otherwise is said, we will assume that there is no limitation in the number of correct (or incorrect) processes in each run, that is, $t = n$ (being t the maximum number of different processes that can crash and recover).

Features and Use of the Network. The processes can invoke the primitive $broadcast(m)$ to send a message m to all processes of the system (except itself). This communication primitive is modeled in the following way. The network is assumed to have a directed link from process p_i to process p_j for each pair of processes $p_i, p_j \in \Pi$ $(i \neq j)$. Then, $broadcast(m)$ invoked at process p_i sends one copy of message m along the link from p_i to p_j, for each $p_{j \neq i} \in \Pi$. If a process crashes while broadcasting a message, the message is received by an arbitrary subset of processes.

Unless otherwise is said, links are asynchronous and fair-lossy [1]. A link is fair-lossy if it can lose messages, but if a process p_i sends a message m permanently (i.e., an infinite number of times) to a correct process p_j, process p_j receives m permanently (i.e., infinitely often). A fair-lossy link [1] does not duplicate or corrupt messages permanently, nor generates spurious messages.

Process Status after Recovery. Following the same model of [1], when a process p_i recovers, it has lost all values stored in its variables previously to crash, and it has also lost all previous received messages. A special case are *stable storage variables*. All values stored in this type of variables will remain available after a crash and recovery. Note that stable storage variables have their cost (in terms of operations latencies), and the algorithms have to reduce their use as far as possible.

Unless otherwise is stated, we consider, like in [1], that when a process p_i crashes executing an algorithm $\mathcal{A}$, if process p_i recovers, it knows this fact, that is, p_i starts executing from a established line of $\mathcal{A}$ different of line 1.

Nomenclature. The asynchronous system with homonymy and with unknown membership previously defined in this section is notated by $HAS_f[\emptyset, \emptyset, n]$.

We denote by $HAS_f[X, Y, t]$ the system $HAS_f[\emptyset, \emptyset, n]$ augmented with the failure detector X ($\emptyset$ means no failure detector), and where all processes initially

know the identities of processes of Y ($\emptyset$ means unknown membership). The third parameter t indicates the maximum number of different processes that can crash and recover (n means that all processes can crash and recover). The sub-index f in the notation is used to denote that links are fair-lossy. For example, $HAS_f[\mathcal{L}, \Pi, n]$ denotes the asynchronous system with homonymous processes and fair-lossy links, enriched with the failure detector $\mathcal{L}$, where all processes initially know the identity of the members of Π, and where all processes can crash and recover. The classical definition of asynchronous systems found in the literature could be denoted by $AS_r[\emptyset, \Pi, t]$. That is, an asynchronous system without homonymy, with reliable links (i.e., where each sent message is delivered to all alive processes without errors and only once), where at most t processes can crash, and where all processes initially know the identity of the members of Π.

We will use HAS to denote a homonymous asynchronous system where the parameters are not relevant. Similarly, we use AS instead of HAS to indicate that it is a classical system where each process has a different identity.

3 Definitions

First, we will formalize here the set agreement problem [9].

Definition 1. *(Set agreement). In each run, every process of the system proposes a value, and has to decide a value satisfying the following three properties:*
1. Validity: Every decided value has to be proposed by some process of the system.
2. Termination: Every correct process of the system eventually has to decide some value.
3. Agreement: The number of different decided values can be at most $n - 1$.

It is easy to see that if $t = n$ and there is not any stable storage variable, if all processes crash jointly previously to decide, and after that they recover, all proposed values will be lost forever. Then, the Validity Property can not be preserved, and, hence, set agreement can not be solved. Thus, any algorithm that implements set agreement needs to use stable storage variables.

Like in [1], we consider that a process p_i proposes a value v when process p_i writes v into a predetermined stable storage variable. Similarly, a process p_i decides a value v when process p_i writes v into another predetermined stable storage variable. Hence, after a recovery, a process p_i, reading these variables, can know easily if a value has already been proposed and/or decided.

The set agreement problem can not be solved in asynchronous systems where any number of processes can crash and not recover ([6], [15], [21]). To circumvent this impossibility result, we use a failure detector [7].

The failure detector $\mathcal{L}$ [13] was defined for asynchronous systems with the crash-stop failure model. We adapt here this definition of $\mathcal{L}$ to asynchronous systems where processes can crash and recover. Let us consider that each process p_i has a local boolean variable $output_i$. We denote by $output_i^\tau$ this variable at

time τ. Let us assume that the value in $output_i$ is *false* while process p_i is crashed (i.e, $output_i^\tau = false$, for all time τ while p_i is down). In each run, a failure detector of class $\mathcal{L}$ satisfies the following two properties:
1. Some process p_i always returns in its variable $output_i$ the value *false*, and
2. If p_i is the unique correct process, then there is a time after which p_i always returns in its variable $output_i$ the value *true*.

More formally, the definition of $\mathcal{L}$ for crash-recovery systems is the following.

Definition 2. *(Failure detector $\mathcal{L}$). For all process $p_i \in \Pi$ and run R, $output_i^\tau = false$ if process p_i is down at time τ in run R. Furthermore, the variable $output_i$ of every process $p_i \in \Pi$ must satisfy in each run R:*
1. $\exists p_i : \forall \tau$, $output_i^\tau = false$, *and*
2. $(Correct = \{p_i\}) \implies \exists \tau : \forall \tau' \geq \tau, output_i^{\tau'} = true$

To solve set agreement, we augment our asynchronous system $HAS_f[\emptyset, \emptyset, n]$ with the loneliness failure detector $\mathcal{L}$, which is the weakest failure detector to achieve set agreement in classical asynchronous message passing systems AS with the crash-stop failure model [13]. As we said previously, we denote this system enhanced with $\mathcal{L}$ as $HAS_f[\mathcal{L}, \emptyset, n]$.

4 Implementing Set Agreement in the Crash-Recovery Model

In this section we present the algorithm $\mathcal{A}_{set}$ (see Figure 1) that implements set agreement in homonymous asynchronous systems with unknown membership and with the failure detector $\mathcal{L}$, that is, in $HAS_f[\mathcal{L}, \emptyset, n]$.

Differently from $\mathcal{A}_{set}$, all algorithms presented in the literature to solve set agreement with $\mathcal{L}$ ([4] and [13]), besides working in crash-stop asynchronous systems, they need to know the system membership to work.

4.1 Explanation of $\mathcal{A}_{set}$

$\mathcal{A}_{set}$ is the algorithm of Figure 1 executed in $HAS_f[\mathcal{L}, \emptyset, n]$ to solve set agreement. Let $id(i)$ be the identifier of process p_i. Note that the values of these process identifiers could be whatever that imposes an order that allows to compare them. Also note that several identifiers can be the same (homonymous processes).

Like in [1], we consider that a process p_i proposes a value v (that is, $propose_i(v)$ is invoked) by writing v into a stable storage variable $PROP_i$. Similarly, a process p_i decides a value v (that is, $decide_i(v)$ is invoked) by writing v into another stable storage variable DEC_i. Let us suppose that both variables have the value $\perp$ previously to any invocation. If a process p_i recovers, it can see easily if it has already proposed or decided a value (that is, if $propose_i(v)$ or $decide_i(v)$ were invoked) reading these stable storage variables and checking if their values are different of $\perp$.

```
propose_i(v): % by writing v into PROP_i
(1)  v_i ← v;
(2)  start task 1

task 1:
(3)  end_i ← false;
(4)  repeat each η time
(5)     % Phase 0
(6)     broadcast (PH0, id(i), v_i);
(7)     if (PH0, id(k), v_k) is received then
(8)        if (⟨id(k), v_k⟩ ≤ ⟨id(i), v_i⟩) then
(9)           v_i ← v_k;
(10)          decide_i(v_k); % by writing v_k into DEC_i
(11)          end_i ← true
(12)       end if
(13)    else
(14)       % Phase 1
(15)       if (PH1, v_k) is received then
(16)          v_i ← v_k;
(17)          decide_i(v_k); % by writing v_k into DEC_i
(18)          end_i ← true
(19)       else
(20)          if (L.output_i=true) then % returned by L
(21)             decide_i(v_i); % by writing v_i into DEC_i
(22)             end_i ← true
(23)          end if
(24)       end if
(25)    end if
(26) until end_i;
(27) start task 2

task 2:
(28) repeat forever each η time
(29)       broadcast (PH1, v_i);
(30) end repeat

when process p_i recovers:
     % by checking PROP_i
(31) if (propose_i() was invoked) then
          % by checking DEC_i
(32)      if (decide_i() was invoked) then
(33)           v_i ← DEC_i;
(34)           start task 2
(35)      else
(36)           v_i ← PROP_i;
(37)           start task 1
(38)      end if
(39) end if
```

Fig. 1. The algorithm $\mathcal{A}_{set}$ for set agreement in $HAS_f[\mathcal{L}, \emptyset, n]$

The variable v_i is used by process p_i to keep the current estimate of its decision value (lines 9 and 16). This variable v_i contains initially the value v proposed by process p_i when it invokes $propose_i(v)$ (line 1). In order to remember, in case of recovering, the changes in v_i before crashing, a process p_i uses the stable storage variables $PROP_i$ and DEC_i (lines 33 and 36).

$propose_i(v)$ starts task 1. This task is a loop that executes lines 6-25 each η time until a decision is taken (and, hence, variable $end_i = true$).

Each process p_i in phase 0 broadcasts a $(PH0, id(i), v_i)$ message with a proposal v_i (initially v_i is p_i's proposal v, line 1) to the rest of processes of the system. After that, process p_i can decide a proposed value if a $(PH0, id(k), v_k)$ message is received. This value v_k is only decided if the condition $\langle id(k), v_k\rangle \leq \langle id(i), v_i\rangle$ happens. This condition is a shortcut for $(id(k) < id(i)) \vee [(id(k) = id(i)) \wedge (v_k \leq v_i)]$. That is, process p_i decides v_k if process p_k has a lesser identifier or, if they have the same identifier, v_k is lesser or equal than v_i. When a process decides, it moves to phase 1. If process p_i has not decided in phase 0, it can decide a value already decided by another process if a $(PH1, v_k)$ message is received. If after that phase 1 process p_i has not decided yet, it can decide its value v_i if the failure detector $\mathcal{L}$ returns *true* (i.e., $\mathcal{L}.output_i = true$). Note that at most $n-1$ processes can get *true* in this variable $output_i$ (from Condition 1 of Definition 2).

Finally, if process p_i decided in phase 0, phase 1, or locally because $\mathcal{L}.output_i = true$, the loop of lines 6-25 finishes, and task 2 starts. As links are not reliable (but

fair-lossy) and processes may crash and recover, with task 2 process p_i guarantees the propagation of its decided value v_i to the rest of processes. This value is broadcast in a $(PH1, v_i)$ message. The propagation is preserved repeating forever this broadcast invocation (lines 28-30).

If a process p_i crashes and recovers while running the algorithm, it always executes, after the recovery, lines 31-39. If process p_i proposed a value v but it crashed before writing any decision value in DEC_i, then p_i will get the proposed value from the stable storage variable $PROP_i$ (line 36). In other case, v_i will obtain its decided value from stable storage variable DEC_i (line 33). If it has already proposed and decided a value, process p_i starts task 2 to propagate this decided value (line 34). If process p_i has proposed a value but it has not decided yet, it starts task 1 to look for a value to decide (line 37).

4.2 Proofs of $\mathcal{A}_{set}$ in $HAS_f[\mathcal{L}, \emptyset, n]$

Lemma 1. *(Validity) For each run, if a process p_i of the system $HAS_f[\mathcal{L}, \emptyset, n]$ decides a value v', then v' has to be proposed by some process of the system $HAS_f[\mathcal{L}, \emptyset, n]$.*

Proof. The variable v_i has initially, when p_i starts for the first time, the value v proposed by process p_i when it invokes $propose_i(v)$ (line 1). Note that if process p_i recovers after proposing a value v but before writing any value in DEC_i, then $v_i = v$ (line 36). Thus, $v_i = v$ is broadcast in $(PH0, v_i)$ messages permanently (line 6 of p_i). So, this value $v_i = v$ only changes if:

Case 1: $(PH0, id(k), v')$ is received from some process p_k such that $\langle id(k), v' \rangle \leq \langle id(i), v \rangle$ (lines 7-12 of p_i). Then, $v_i = v'$ and $DEC_i = v'$, being v' the initial value proposed by process p_k.

Case 2: $(PH1, v')$ is received (lines 15-18 of p_i). We have three subcases:

Case 2.1: $(PH1, v')$ was broadcast by some process p_j after receiving $(PH0, id(k), v')$ of p_k ($p_k \neq p_j$) such that $\langle id(k), v' \rangle \leq \langle id(j), v \rangle$ (lines 7-12 and task 2 of p_j). Then, $v_i = v'$ and $DEC_i = v'$, being v' the initial value proposed by process p_k.

Case 2.2: $(PH1, v')$ was broadcast by some process p_j after receiving $(PH1, v')$ of other process p_x (lines 15-18 and task 2 of p_j). Note that this $(PH1, v')$ is broadcast, like in Case 2.1, when process p_x receives $(PH0, id(k), v')$ of some process p_k such that $\langle id(k), v' \rangle \leq \langle id(x), v \rangle$. Then, $v_i = v'$ and $DEC_i = v'$, being v' the initial value proposed by process p_k.

Case 2.3: $(PH1, v')$ was broadcast by process p_k when $output_k = true$ (lines 20-23 and task 2 of p_k). Then, $v_i = v'$ and $DEC_i = v'$, being v' the initial value proposed by process p_k.

Therefore, for each run, if a process p_i of the system decides a value v', then v' has to be proposed by some process of the system.

Lemma 2. *(Agreement) For each run, the number of different decided values in the system $HAS_f[\mathcal{L}, \emptyset, n]$ is at most $n - 1$.*

Proof. Let us suppose, by the way of contradiction, that there is a run R such that the number of different decided values is n. From Lemma 1, each decided value in R has to be one of the proposed values. Hence, if we find in this run R a proposed value which is not decided, we reach a contradiction.

Note that if in run R there are two processes p_i and p_j such that p_i proposes v_i, and p_i proposes v_j being $v_i = v_j$, then the statement of this lemma is trivial. So, we consider that $v_i \neq v_j$, for all p_i and p_j of the system.

Let us denote by G the set of processes that decide in this run R not executing lines 20-23. Note that $G \neq \emptyset$ from Condition 1 of Definition 2. Also note that this implies that every process $p_j \notin G$ decides its own proposed value.

Let us assume that $p_i \in G$ is the process with the greatest pair $\langle id(i), v\rangle$ among processes in G. Let us also assume that p_i proposes v_i. So, if contradiction holds, v_i has to be decided by p_i or by another different process p_j. We now analyze both cases and we will see that it is impossible that some process decides this value v_i in run R. Hence, we reach a contradiction.

Case 1: Process p_i decides v_i. As p_i, by definition, has the greatest pair $\langle id(i), v\rangle$ among processes in G, it did not receive any $(PH1, v_i)$ message from any process in G. Due to the fact that every process $p_j \notin G$ decides its own proposed value v_j (being $v_j \neq v_i$), process p_i did not receive any $(PH1, v_i)$ message from any process p_j. Then, it is impossible that process p_i decides its own proposed value v_i.

Case 2: Process p_j decides v_i, being $j \neq i$. As every process $p_k \notin G$ decides its own proposed value v_k (being $v_k \neq v_i$), then process $p_j \in G$. Hence, if process p_j decides v_i, which is a different value of its own proposed value v_j, it is because p_j receives a $(PH0, id(l), v_i)$ or $(PH1, v_i)$ message from some process $p_l \in G$. This is impossible because, by definition, p_i has the greatest pair $\langle id(i), v_i\rangle$ among processes in G, and $\langle id(i), v_i\rangle \leq \langle id(x), v_x\rangle$ is always false for all $p_x \in G$ (line 8).

Therefore, we reach a contradiction, and, for each run, the number of different decided values is at most $n - 1$.

Lemma 3. *(Termination) For each run, every process $p_i \in Correct$ of the system $HAS_f[\mathcal{L}, \emptyset, n]$ eventually decides some value.*

Proof. Let us suppose, by the way of contradiction, that there is a run R such that a correct process p_i never decides. Hence, if process $p_i \in Correct$ never decides in run R it is because lines 10, 17 and 21 are never executed.

Let us prove that this situation is impossible. If line 21 is never executed, then $\mathcal{L}.output_i = false$ permanently. If this is so, it is because there is at least another process p_k that is correct (from Condition 2 of Definition 2). Note that p_i, after its last recovery (if any), will be permanently broadcasting $(PH0, id(i), v_i)$ messages, being v_i the proposed value of p_i (line 6 of p_i). Hence, if process p_i never receives $(PH1, -)$ messages (lines 15-18 of p_i) it is because all processes p_l (included p_k) that receive the messages of p_i have a lesser pair $\langle id(l), v_l\rangle$ than $\langle id(i), v_i\rangle$ (line 8 of p_l). Nevertheless, process p_i will receive $(PH0, id(k), v_k)$ messages of p_k because links are fair-lossy, and correct process p_k also broadcasts $(PH0, id(k), v_k)$ messages permanently (line 6 of p_k). Then, p_i will execute line 10 because $\langle id(k), v_k\rangle < \langle id(i), v_i\rangle$. Hence, process p_i will decide v_k in run R.

Therefore, we reach a contradiction, and, for each run, every process $p_i \in Correct$ eventually decides some value.

Theorem 1. *The algorithm of Figure 1 implements set agreement in the system* $HAS_f[\mathcal{L}, \emptyset, n]$.

Proof. From Lemma 1, Lemma 2 and Lemma 3, the validity, agreement and termination properties (respectively) are satisfied in every run. Hence, the algorithm of Figure 1 solves set agreement in the system $HAS_f[\mathcal{L}, \emptyset, n]$.

5 On the Implementability of $\mathcal{L}$ in the Crash-Recovery Model

In this section we prove that the failure detector $\mathcal{L}$ can not be implemented, even in a synchronous system where the membership is known, if up to n different processes can crash and recover, that is, $\mathcal{L}$ is not realistic [10]. We also prove in this section that the failure detector $\mathcal{L}$ can not be implemented in a partially synchronous system even if the membership is known and up to $n-1$ different processes can crash and recover.

Let $SS_r[\emptyset, \Pi, n]$ be a system like $AS_r[\emptyset, \Pi, n]$ but synchronous, that is, the maximum time to execute a step is bounded and known by every process, and the time to deliver a message is also known by all processes. Hence, $SS_r[\emptyset, \Pi, n]$ is a synchronous system where all processes have different identities, links are reliable, the membership is known, and the maximum number of processes that can crash and recover is $t = n$. Similarly, let $PSS_r[\emptyset, \Pi, n]$ be a system like $SS_r[\emptyset, \Pi, n]$ but partially synchronous [14], that is, the maximum time to execute a step by each process p_i is bounded, but unknown by every process different of p_i, and the time to deliver a message is bounded but unknown.

Lemma 4. *For every run, if in* $SS_r[\emptyset, \Pi, t]$ *or* $PSS_r[\emptyset, \Pi, t]$ *when* $t \geq n-1$ *a process* $p_i \in Correct$ *stops receiving messages from the rest of processes at some time* τ*, there is a time* $\tau' \geq \tau$ *where* $output_i^{\tau'} = true$.

Proof. Let us assume, by the way of contradiction, that there is a run R where some correct process p_i stops receiving messages from the rest of processes at some time τ, but for all time $\tau' \geq \tau$ it has $output_i^{\tau'} = false$.

Let us consider another run R' behaving exactly like R until time τ, and at this time τ all alive processes crash permanently except p_i. From Condition 2 of Definition 2 of $\mathcal{L}$, there is a time τ' where $output_i = true$. Note that each process only knows that in a run the rest of processes can crash, but it does not know a priori how many processes will crash or who they will be. Then, R and R' are indistinguishable until time τ' for p_i, and, hence, there is a time τ' where $output_i = true$ in R, which is a contradiction.

The following theorem shows that failure detector $\mathcal{L}$ can not be implemented in $SS_r[\emptyset, \Pi, n]$.

Theorem 2. *There is no algorithm $\mathcal{A}$ that implements the failure detector $\mathcal{L}$ in every run of a system $SS_r[\emptyset, \Pi, n]$, even if there is not any unstable process.*

Proof. Let us assume, by the way of contradiction, that there is an algorithm $\mathcal{A}$ that implements the failure detector $\mathcal{L}$ in every run of a system $SS_r[\emptyset, \Pi, n]$, even if there is not any unstable process.

For simplicity, let us consider that $\Pi = \{p_1, p_2, \ldots, p_n\}$, and that all these n processes of Π are eventually-up (hence, correct). Let us construct a valid run R of $\mathcal{A}$ as follows. For each process p_i, at time τ_i all processes crash except process p_i. From lemma 4, there is a time $\tau_i' \geq \tau_i$ where $output_i = true$. Now, all crashed processes recover at this time τ_i'. Let τ_1=0, and $\tau_i' < \tau_{i+1}$, $i = 1, \ldots, n$. Finally, after time τ_n' all processes keep alive in R (i.e., there is no unstable processes). Then, at time τ_n' all processes have had $output = true$ at some time, which violates Condition 1 of Definition 2. Hence, we reach a contradiction.

Therefore, there is no algorithm $\mathcal{A}$ that implements the failure detector $\mathcal{L}$ in every run of a system $SS_r[\emptyset, \Pi, n]$, even if there is not any unstable process.

The following theorem shows that failure detector $\mathcal{L}$ can not be implemented in $PSS_r[\emptyset, \Pi, n-1]$.

Theorem 3. *There is no algorithm $\mathcal{A}$ that implements the failure detector $\mathcal{L}$ in every run of a system $PSS_r[\emptyset, \Pi, n-1]$, even if there is not any unstable process.*

Proof. From Lemma 4, there is a time τ_i after which each process $p_i \in Correct$ sets $output_i = true$ if it stops receiving messages from the rest of processes. Let us consider that every process p_i in a run R is permanenly-up (hence, correct) and takes an step after a time τ which is greater than the maximum time τ_i, for all process $p_i \in \Pi$. Note that processes do not know a piori the time needed by other processes to take a step in run R, nor the number of other processes that are correct in R. Hence, there is a time $\tau' \geq \tau$ after which every process p_i has $output_i = true$, which violates the Condition 1 of Definition 2 of $\mathcal{L}$. Therefore, there is no algorithm $\mathcal{A}$ that implements the failure detector $\mathcal{L}$ in every run of a system $PSS_r[\emptyset, \Pi, n-1]$, even if there is not any unstable process.

6 Implementing $\mathcal{L}$ in the Crash-Recovery Model

From Section 5, we know that the failure detector $\mathcal{L}$ neither can be implemented in a synchronous system when until $t = n$ processes can crash and recover, that is, $\mathcal{L}$ is not realistic [10], nor in a partially synchronous system where $t = n - 1$. Now, we enrich here the system with a property such that we can circumvent this impossibility result. This property reduces to $t = n - 1$ the number of processes that can crash and recover in a synchronous system. Note that all algorithms found in the literature that implement the loneliness failure detector $\mathcal{L}$ ([4], [19]) work in systems where processes can crash but not recover and where up to $t = n - 1$ processes can crash and where the membership is totally known. Therefore, we present in this section an implementation of $\mathcal{L}$ (denote it by $\mathcal{A}_{\mathcal{L}}$)

for a synchronous system with homonymous processes, a partial knowledge of the membership, and where until $t = n - 1$ different processes can crash and recover.

6.1 Model

Let HSS be a system like HAS but synchronous. By synchronous we mean that processes start their execution at the same time, the time to execute a step is bounded and known by every process, the time to deliver a message sent through a link is at most Δ units of time, and this time is also known by all processes. For simplicity, we consider that the local execution time is negligible with respect to Δ (i.e., the execution time of a line of the algorithm is zero).

6.2 Algorithm $\mathcal{A}_{\mathcal{L}}$

We show in this section that the algorithm $\mathcal{A}_{\mathcal{L}}$ of Figure 2 implements the failure detector $\mathcal{L}$ in $HSS_r[\emptyset, Y, n-1]$ when $|Y| \geq 2$ and two processes of Y have different and known identities.

For each process p_i, $output_i$ is initially *false* (line 3). Process p_i uses the boolean value of the stable storage variable $restarted_i$ to communicate to the other processes if it has ever crashed (initially is *false*, line 1). If process p_i recovers, it will execute lines 19-20, and $restarted_i$ will be *true* (line 19). By definition of the system HSS used to execute $\mathcal{A}_{\mathcal{L}}$, process p_i knows at least two processes' identifiers with different values. These two known identifiers of Y with different value are $IDENT_1$ and $IDENT_2$ in Figure 2. Then, each process p_i whose identifier is neither $IDENT_1$ nor $IDENT_2$ changes $output_i$ to *true* (lines 4-6). Every η time, $\eta > \Delta$, each process p_i broadcasts heartbeats with $(alive, restarted_i)$ messages that arrive synchronously (at most Δ units of time later) to the rest of processes of the system (line 8). Note that we select a value η greater than Δ to allow that messages broadcast in line 8 arrive to processes on time in each iteration of line 9.

After Δ units of time, process p_i analyzes the messages received (rec_i) to see if it has to set $output_i$ to true (lines 11-17). Note that once $output_i = true$, process p_i never changes it to *false* again while it is running. Only if process p_i crashes and recovers, line 3 is executed again and $output_i$ is *false* again, but $restarted_i$ will be *true* in this case. The variable $count_i$ counts the number of heartbeats received by p_i from processes that are up, and that have never crashed (lines 12-14). If this number of messages is 0, then p_i sets $output_i = true$ (lines 15-17).

Note that in all algorithms in the literature that implement set agreement with $\mathcal{L}$ (our algorithm $\mathcal{A}_{set}$ included), the performance is improved if processes obtain *true* from $\mathcal{L}$ as soon as possible. This happens because a process of set agreement can decide locally (without waiting to receive any message) if *true* is returned by $\mathcal{L}$. For that reason, our algorithm $\mathcal{A}_{\mathcal{L}}$ with a partial knowledge of the membership immediately sets $output = true$ permanently in $n - 2$ processes (lines 4-6 of Figure 2).

```
init:
(1)  restarted_i ← false; % stable storage variable
(2)  start task 1

task 1:
(3)  output_i ← false;
       % IDENT_1 and IDENT_2 are two
       % identifiers known by all processes
(4)  if ((id(i) ≠ IDENT_1) ∧ (id(i) ≠ IDENT_2)) then
(5)          output_i ← true
(6)  end if
(7)  repeat forever each η time
(8)      broadcast (alive, restarted_i);
(9)      wait Δ time;
(10)     let rec_i be the set of (alive, restarted) messages received;
(11)     count_i ← 0;
(12)     for_each ((alive, restarted) ∈ rec_i such that restarted = false) do
(13)         count_i ← count_i + 1
(14)     end for_each
(15)     if (count_i = 0) then
(16)         output_i ← true
(17)     end if
(18) end repeat

when process p_i recovers:
(19)  restarted_i ← true; % stable storage variable
(20)  start task 1
```

Fig. 2. Algorithm $\mathcal{A}_{\mathcal{L}}$ for process p_i to implement $\mathcal{L}$

Theorem 4. *The algorithm $\mathcal{A}_{\mathcal{L}}$ implements the failure detector $\mathcal{L}$ in a system $HSS_r[\emptyset, Y, n-1]$ when $|Y| \geq 2$ and two processes of Y have different identities.*

The proof of this theorem is omitted due to space limitations.

7 Conclusions

We study the *set agreement* problem in message passing systems with the weakest failure detectors $\mathcal{L}$ in crash-recovery asynchronous systems with homonymous processes and without a complete initial knowledge of the membership.

References

1. Aguilera, M.K., Chen, W., Toueg, S.: Failure Detection and Consensus in the Crash-Recovery Model. Distributed Computing 13(2), 99–125 (2000)
2. Arévalo, S., Fernández Anta, A., Imbs, D., Jiménez, E., Raynal, M.: Failure Detectors in Homonymous Distributed Systems (with an Application to Consensus). In: Proc. IEEE 32nd IEEE Int. Conf. on Distributed Computing Systems (ICDCS), pp. 275–284 (2012)
3. Arévalo, S., Jiménez, E., Larrea, M., Mengual, L.: Communication-efficient and crash-quiescent Omega with unknown membership. Information Processing Letters 111(4), 194–199 (2011)
4. Biely, M., Robinson, P., Schmid, U.: Weak Synchrony Models and Failure Detectors for Message Passing (k-)Set Agreement. In: Abdelzaher, T., Raynal, M., Santoro, N. (eds.) OPODIS 2009. LNCS, vol. 5923, pp. 285–299. Springer, Heidelberg (2009)

5. Bonnet, F., Raynal, M.: Anonymous Asynchronous Systems: The Case of Failure Detectors. Distributed Computing (in press 2013), doi:10.1007/s00446-012-0169-5
6. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 91–100. ACM, New York (1993)
7. Chandra, T., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM 43(2), 225–267 (1996)
8. Chandra, T., Hadzilacos, V., Toueg, S.: The Weakest Failure Detector for Solving Consensus. Journal of the ACM 43(4), 685–722 (1996)
9. Chaudhuri, S.: More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. Information and Computation 105, 132–158 (1993)
10. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: A Realistic Look At Failure Detectors. In: Proc. 42th International IEEE Conference on Dependable Systems and Networks, DSN 2002, pp. 345–353 (2002)
11. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Hadzilacos, V., Kuznetsov, P., Toueg, S.: The Weakest Failure Detectors to Solve Certain Fundamental Problems in Distributed Computing. In: Proceedings of 23th ACM Symp. on Principles of Distrib. Comp., PODC, pp. 338–346 (2004)
12. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kermarrec, A.M., Ruppert, E., Tran, H.: The Byzantine agreement with homonymous. In: Proceedings of 30th ACM Symp. on Principles of Distrib. Comp., PODC, pp. 21–30 (2011)
13. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Tielmann, A.: The Weakest Failure Detector for Message Passing Set-Agreement. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 109–120. Springer, Heidelberg (2008)
14. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchronism needed for distributed systems. Journal of the ACM 34(1), 77–97 (1987)
15. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. Journal of the ACM 46(6), 858–923 (1999)
16. Hurfin, M., Mostefaoui, A., Raynal, M.: Consensus in asynchronous systems where processes can crash and recover. In: Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, SRDS 1998, pp. 280–286 (1998)
17. Jiménez, E., Arévalo, S., Fernández, A.: Implementing unreliable failure detectors with unknown membership. Information Processing Letters 100(2), 60–63 (2006)
18. Martín, C., Larrea, M., Jiménez, E.: Implementing the Omega Failure Detector in the Crash-recovery Failure Model. Journal of Computer and System Sciences 75(3), 178–189 (2009)
19. Mostéfaoui, A., Raynal, M., Stainer, J.: Relations Linking Failure Detectors Associated with k-Set Agreement in Message-Passing Systems. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 341–355. Springer, Heidelberg (2011)
20. Raynal, M.: Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems, 250 pages. Morgan & Claypool Publishers (2010)
21. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM Journal on Computing 29(5), 1449–1483 (2000)

Black Art: Obstruction-Free k-set Agreement with $|MWMR\ registers| < |proccesses|$*

Carole Delporte-Gallet[1], Hugues Fauconnier[1], Eli Gafni[2], and Sergio Rajsbaum[3]

[1] U. Paris Diderot, France
{cd,hf}@liafa.univ-paris-diderot.fr
[2] Computer Science Department, UCLA, USA
eli@ucla.edu
[3] Instituto de Matemáticas, UNAM, Mexico
rajsbaum@math.unam.mx

Abstract. When n processes communicate by writing to and reading from $k < n$ MWMR registers the "communication bandwidth" precludes emulation of SWMR system, even non-blocking.

Nevertheless, recently a positive result was shown that such a system either wait-free or obstruction-free can solve an interesting one-shot task. This paper demonstrates another such result. It shows that $(n-1)$-set agreement can be solved obstruction-free with merely 2 MWMR registers. Achieving k-set agreement with $n-k+1$ registers is a challenge. We make the first step toward it by showing k-set agreement with $2(n-k)$ registers.

Keywords: Shared memory, MWMR registers, k-set agreement, Consensus with bounded memory.

1 Introduction

Designing not to mention proving algorithms for the multi-writer multi-reader shared memory (MWMR SM) system is still a "Black-Art" (Dictionary: a mysterious skill that is difficult to master or describe). Every area of research, at least at its infancy is a Black-Art. Why do we contend that MWMR, though chronologically is quite old, is at the Black-Art phase, while, say, designing and proving well designed algorithms in a single-writer multi-reader shared memory (SWMR SM) system is not Black-Art any more?

An area of investigation has matured when the Ph.D-worthy results of yesteryear can be given as an homework now. Consider the problem of solving Renaming in the SWMR SM system [3]. At the time, the solution was considered ingenious, its proof called for the hiring of combinatorial guns as coauthors, and its depth worthy of JACM publication. Today, solving this problem not necessarily in the original way it was solved, and proving the solution is considered a homework-problem of relatively modest difficulty.

* Supported by ANR DISPLEXITY, ECOS/ANUIES and UNAM-PAPIIT.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 28–41, 2013.

What has changed? Few years after the appearance of the Renaming paper it was realized that the SWMR model is equivalent to one in which read is done in an Atomic Snapshot [1]. Few years later than that, it was furthermore realized that one can replace Atomic Snapshots with Immediate Snapshots [8], and furthermore, that one can consider Iterated Models [9]. With all this high-level techniques and constructs in hand, designing the algorithm can be compared conceiving an algorithm in JAVA or C++ as opposed to starting and thinking in BASIC.

On the impossibility side, one later learned of the proof of the impossibility of set consensus [7,23,25] and reducing Adaptive Renaming to set consensus is again straightforward [19]. Furthermore, the characterization of wait-free[1] solvability through its connection to Topology in [23], has allowed to infer the implicit existence of an algorithm, rather than explicit.

In short, with SWMR we acquired many tools and deep understanding.

We contend that our understanding of the MWMR system is still about where it was 30 years ago. The proof that Mutual-Exclusion in a system of n processes, requires n MWMR registers [10] is as ingenious as Renaming if not more, yet essentially that's where we stand today. It uses a "covering argument" which time and again has to be reinvented from scratch for the particular problem at hand, by experienced hands who have acquired the intuition for it in for example [15]. MWMR algorithms in the "wait" model[2] have been dealt with in [27] and algorithms there are beyond Black-Art, concise, and astonishingly, work without revealing a hint of how the authors conceived them.

In fact, the simple basic fundamental question, asked almost 3 decades ago, at the time of the appearance of the Mutual Exclusion lower bound, of whether to solve (obstruction-free[3]) n processes consensus require n MWMR registers is still wide open. All we know is a lower bound of $\sqrt{n}$ [16], an order of magnitude away from the conjectured result.

On the positive side, the question of "time-stamps" was investigated and shown to be wait-free solvable with less than n registers [15,20].

The first somewhat systematic attempt of dealing with MWMR systems by the current authors recently appeared in [13]. It shows that like Mutual-Exclusion, SWMR system emulation even non-blocking requires at least n MWMR registers, while for wait-free implementation it requires $2n - 1$ registers [14].

Our understanding of this Black-Art gives us the intuition that most tasks which are wait-free solvable are not solvable with k, $k < n$, MWMR registers. But there are "few" tasks which are wait-free solvable. In contrast, "most" tasks are obstruction-free solvable. Yet, we cannot conceive of Adaptive Renaming [4] being obstruction-free solvable with $k < n$, let alone Symmetric Renaming [3]. Since we have seen already a task that is solvable with $k < n$ [20] it is valid to wonder what are necessary conditions on a task that would make it amendable

[1] In the wait free model, each process that takes step terminates its operation.

[2] In the "wait" model, a process can wait another process if it saw this process taking steps.

[3] In this model, if a process takes step alone it terminates its operation.

to obstruction-free solvability with $k < n$? The first step in this investigation should collect more examples of tasks obstruction-free solvable with $k < n$. That's what started this paper. Is there a simple yet non-contrived task that is obstruction-free solvable with $k < n$ MWMR registers? Once this question is asked, we exhibit a simple, almost an obvious, algorithm for the $(n-1)$-set agreement that uses $k = 2$ registers. Thus, the contribution of this paper is in raising this question and as shown below, in explaining, that extending this result so that with $k+1$ registers we can achieve $(n-k)$-set agreement is a non-trivial intriguing challenge.

What is the idea behind our algorithm? In [13] it was observed that k processes with k MWMR registers can non-blocking emulate a SWMR system. If we have $n > k$ processes, assume processes depart with their own name until there are just $k+1$ processes in the system. Then in an implicit way, unknown to the processes, they can communicate read-write and therefore emulate say commit-adopt [18,28,2] to obstruction-free solve consensus. If the "last" $k+1$ processes output a single value we obtain $(n-k)$-set agreement. Thus, the rough idea would be to ask processes to follow the read-write algorithm in [13] and never depart only before commit-adopt of a value.

The complication occurs when we down to k processes, is that commit-adopt assumes some "initial state" of processes. Now, process's state is remnants of their interaction with other processes before the number of processes dropped to k.

Thus, in a way, we need a "self-stabilizing" commit-adopt. We conjecture that such a self stabilizing algorithm exists, but we haven't found it yet.

We make the first step toward the result by showing (obstruction-free) k-set agreement with $2(n-k)$ registers.

2 Model

We assume a standard asynchronous shared-memory model of computation with n processes communicating by reading and writing to a fixed set of shared registers [6,24]. The processes have unique ids in $\{1, \ldots, N\}$, with $N >> n$. Processes know n but don't know the identities of other processes. Processes may take a finite or an infinite number of steps but we assume that at least one process takes an infinite number of steps.

2.1 Progress Condition

Three progress conditions that have received much attention are obstruction-free, non-blocking and wait-free. The *obstruction-free* [22] progress condition states that when a process takes steps alone it terminates its operations. The *non-blocking* progress condition states that when there are concurrent operations at least one process terminates its operations. The *wait-free* [21] progress condition states that each process terminates its operations in a bounded number of its own steps.

2.2 Shared Memory

The shared memory consists of a set of atomic Multi-Writer Multi-Readers (MWMR) registers. We assume that processes can read and write any MWMR register and these operations are atomic [21]. For short, we usually omit the term atomic. A process executes its code by taking three types of atomic steps: the *read* of a register, the *write* of a register, and the modification of its local state.

We also consider two more powerful operations: *update* and *scan*. The update operation takes a register and data value as arguments and does not return a value. It writes the data value in the register. The scan operation has no arguments and returns a vector of n elements from an arbitrary set of data values. The vector returned is a snapshot, an instantaneous view of the registers. In [1], there are non blocking and wait free linearizable implementations of scan and update for single-writer algorithm (each register may be written by only one process) and multi-writer algorithm (each register may be written by all processes).

In [1], it is observed that if every write leaves a unique indelible mark whenever it is executed in the shared memory, then if two consecutive reads of the entire shared memory return identical values then the values returned constitute a snapshot. Using this idea, the paper designed a non-blocking linearizable implementation of scan and update for single-writer algorithm. To leave a unique indelible mark in a write operation, a process updates a register by writing not only the data but also the identity of the process and a sequence number (the number of writes already made by the process). The value in a shared register may be unbounded. This implementation works in the same way in our case and we get a non blocking linearizable implementations of *scan* and *update*.

Proposition 1 ([1]). *There is a non blocking linearizable implementation of scan/update of n MWMR registers that uses n MWMR registers (and no other shared register).*

2.3 k-set Agreement

A decision task T is described by a triple (I, O, Δ) where I is the set of inputs, O is the set of outputs, and Δ is the specification of the task mapping every input to a set of possible outputs.

We consider the classical *k-set agreement* decision task [11] in which each process proposes its input value, and after communicating with the others, it has to decide on one of the proposed inputs, such that there are at most k decided values. We assume that the input values come from a finite set of values *Values*.

There are three requirements:

- at most k values are decided (*agreement*),
- if a process decides v, this value has been proposed by some process (*validity*),
- if a process takes an infinite number of steps then it decides (*termination*).

When $k = n - 1$, k-set agreement is also known as set agreement.

The well-known *consensus* task [17] is nothing else than 1-set agreement.

For decision tasks, wait-free solvability and non-blocking solvability are equivalent, see e.g. [6]. It is well known that k-set agreement is not wait-free solvable [7,23,26] in shared memory, so we study here the obstruction-free solvability of k-set agreement. It is also know that there exits an obstruction-free k-set agreement implementation (for example [5]). We are interested here on the space complexity of an implementation.

3 Algorithm

Our algorithm solves obstruction-free k-set agreement with $2(n-k)$ registers. The main idea of our algorithm is to ensure that the $n-k+1$ processes that decide last, decide using consensus. In this way, the $k-1$ processes that decide first may decide any value. Thus, the $(n-k+1)$ last processes decide on one value, and at most k values are decided in total.

The algorithm is in Figure 1. Processes share an array R of $2(n-k)$ cells, each cell is a MWMR-registers. Each process maintains a proposal given by its local variable *prop*, initially, this variable contains the proposed value of the process for the k-set agreement. Each process tries to write *prop* in all cells of R (using an update). When a process sees (using a scan of R) that all cells contain the same value v, it decides v.

More precisely, each process updates *prop* in a cyclic order on $2(n-k)$ in cells of R, and then reads all the cells with a scan. From this scan, if all the values in R are the same value v, then the process decides this value v. If at least $(n-k)$ cells among all the cells except the one that the process has just written contain the same value v, then v will be the value proposed for the next update. When the process does not change its proposal then it will update with this proposal the next (in cyclic order) cell, else it will write the same cell.

As noted before the proposed algorithm achieves consensus among the last $(n-k+1)$ processes with $2(n-k)$ registers. Roughly speaking, with less than $(n-k)$ processes, at least $(n-k)$ registers are not "covered" by processes (a register is covered if there is a process whose the next step will write it) and the algorithm ensures that if at some time all cells contain the same value, this value will remain forever in at least $(n-k)$ cells, avoiding any other decision. Moreover, if some process eventually takes all its steps alone (the obstruction-free assumption), it is easy to verify that it will succeed to write the same value in all cells and then decide.

Now we proceed with the correctness proof of the algorithm.

Each process alternates update and scan of the shared memory R. By definition scan and update are linearizable and when we say that some scan or update operation *op* occurs at some time τ^{op}, time τ^{op} is the linearization time of this operation.

For any local variables or shared registers v (including the shared memory R), we denote by v^τ the value of the variable v at time τ.

We say that at time τ process p *covers* cell i of R if the next operation on R for p is an update of cell i of R.

Shared variables:

R :`array of` $2(n-k)$ `MWMR-register`

initialization: $\forall i(0 \leq i < 2(n-k)) : R[i] = \perp$

CODE FOR PROCESS p

```
1  prop = v_p                                                    /*p proposal */
2  View :array of 2(n − k) Values ∪ {⊥}
3  i = 0
4  forever do
5      update(R[i], prop)
6      View = scan(R)
7      if | {View[j]|(0 ≤ j < 2(n − k))} |> 1
8      then                                                      /* more than one value in R */
9          if (∃v(v ≠ ⊥) :| {j | j ≠ i ∧ (0 ≤ j < 2(n − k)) ∧ v = View[j]} |≥ (n − k))
10             then
11                 let v (v ≠ ⊥) such that
                       (| {j | j ≠ i ∧ (0 ≤ j < 2(n − k)) ∧ v = View[j]} |≥ (n − k))
12                 if prop = v
13                     then i := (i + 1) mod 2(n − k)
14                     else prop := v
15             else
16                 i := (i + 1) mod 2(n − k)
17     else                                                      /* one value in R */
18         let v such that ∀j(0 ≤ j < 2(n − k)) : v = View[j]
19         decide v ; exit
```

Fig. 1. k-set agreement with $2(n-k)$ MWMR registers

In the following we say that v *is* i*-critical* at time τ if and only if $v \neq \perp$ and $|\{j | j \neq i \wedge (0 \leq j < 2(n-k)) \wedge R^{\tau}[j] = v\}| \geq n-k$.

We remark that it may happen that at the same time v is i-critical and $v' \neq v$ is j-critical for some $j \neq i$. However, if v is i-critical then v is the value of a majority of cells in array R without the cell $R[i]$. Then:

Lemma 1. *At any time at most one v is i-critical.*

Directly from the algorithm we have:

Lemma 2. *After an update of cell i by p if v is i-critical at the time of the next scan of p, the proposal value of p for the next update is v.*

Consider any process p that decides. Time τ_d^p will denote the (linearization) time of the *last* scan made by p. As the decision for p is entirely determined by this scan, by definition, τ_d^p will be considered as the *decision time* for p.

We prove now the safety property: no more than k values are decided.

By contradiction assume that strictly more than k values have been decided, hence strictly more than k processes decide. Let p_0 be the process that decides

the k^{th} value (ordered by decision times). Hence at most $k-1$ processes decide before p_0 and at most $n-k$ processes may decide after p_0. Let v_0 be the value decided by p_0. Notice that at time $\tau_d^{p_0}$ all values written in R are equal to v_0.

We introduce some notations:

- Let $Proc(v_0)^\tau$ be the processes for which (i) $prop \neq v_0$ at time τ and (ii) the next operation on R is an update.
- For each process p in $Proc(v_0)^{\tau_d^{p_0}}$, i_p is the index of the cell of the first update after time $\tau_d^{p_0}$.
- $L1(\tau) = \{i | R[i]^\tau \neq v_0\}$.
- $L2(\tau) = \{i|$ there exists $p \in Proc(v_0)^{\tau_d^{p_0}}$ such that, between $\tau_d^{p_0}$ and τ, p has made its first update on cell i and p has not made its second update$\}$.

By construction of $\tau_d^{p_0}$, and the definition of $L2(\tau)$:

Lemma 3. $|L2(\tau)| \leq n-k$

We have:

Lemma 4. *For every time $\tau \geq \tau_d^{p_0}$,*

(1) $L1(\tau) \subseteq L2(\tau)$,

(2) for every p in $Proc(v_0)^{\tau_d^{p_0}}$, in every update made by p between $\tau_d^{p_0}$ and τ except the first one, p updates by v_0, and

(3) for every p not in $Proc(v_0)^{\tau_d^{p_0}}$, in every update made by p between $\tau_d^{p_0}$ and time τ, p updates by v_0.

Proof. Initially, at time $\tau_d^{p_0}$ there is no update before $\tau_d^{p_0}$ then $L2(\tau_d^{p_0})$ is empty and (2) and (3) are true. At time $\tau_d^{p_0}$, as p_0 decides, all values written in R are equal to v_0, and so $L1(\tau_d^{p_0}) = \emptyset$. Then we get (1).

We proceed by induction. Assume that (1), (2), and (3) hold for any time between $\tau_d^{p_0}$ and τ. A scan operation doesn't change (1), (2), and (3). Let τ' be the time of the next update on R made by some process say p. We'll show that (1), (2) and (3) hold at τ'.

- If p is in $Proc(v_0)^{\tau_d^{p_0}}$, by induction hypothesis, (3) holds at τ'. There are three cases:
  - This update is the first update of process p. i_p is added to $L1(\tau)$ and $L2(\tau)$, the other elements of $L1(\tau)$ and $L2(\tau)$ are unchanged: $L1(\tau') = L1(\tau) \cup \{i_p\}$, $L2(\tau') = L2(\tau) \cup \{i_p\}$. Thus by induction hypothesis, (1) holds at τ'. (2) trivially holds.
  - This update is the second update of process p. Between the two updates, following the algorithm, p executes a scan at some time, say ϕ, between its first and its second update.
    We claim that at the time of this scan, v_0 is i_p-critical: i_p is in $L2(\phi)$. As $L1(\phi) \subseteq L2(\phi)$ and, by Lemma 3 $|L2(\phi)| \leq (n-k)$, then there is at least $(n-k)$ cells excluding the cell i_p that contains v_0. Then v_0 is i_p-critical at time ϕ.
    As v_0 is i_p-critical at the time of the scan after the first update on cell i_p, by Lemma 2, p changes its *prop* to v_0. By the algorithm, the next

update of p remains on cell i_p. Then at the next update at time τ', p updates cell i_p by v_0. Then (2) holds.
$L2(\tau') = L2(\tau) - \{i_p\}$ or $L2(\tau') = L2(\tau)$ (it is possible that i_p remains in $L2$ in case of one process q in $Proc(v_0)^{\tau_d^{p_0}}$ has already made its first update at cell $i_q = i_p$ but not its second update). After the update v_0 is written in cell i_p then $L1(\tau') = L1(\tau) - \{i_p\}$. Thus by induction hypothesis, (1) holds at τ'.

- If this operation is an update of process p different from its first and second update. At the time ϕ of the previous update, p is not in $L2(\phi)$ and will be never inserted in $L2$, then (α) $L2(\tau') = L2(\tau)$.
Between the update at time ϕ and the next update at time τ', following the algorithm, p executes a scan. At the time ψ of this scan p is not in $L2(\psi)$, then $|L2(\psi)| < n - k$. By induction hypothesis (1), $L1(\psi) \subseteq L2(\psi)$, then there is at least $n - k + 1$ values equal to v_0 in R and for every j, v_0 is j-critical. By Lemma 2, the value of $prop_p$ after this scan is v_0. Thus, (2) holds.
As p updates R by v_0, then $L1$ may decreased. With (α) and the induction hypothesis, we get (1) at τ'.

– if this operation is an update of a process that is not in $Proc(v_0)^{\tau_d^{p_0}}$. In this case (2) holds and we have (a) $L2(\tau') = L2(\tau)$. Furthermore, as $L2$ is a subset of $Proc(v_0)^{\tau_d^{p_0}}$, (b) for all time $\phi \geq \tau_d^{p_0}$, $|L2(\phi)| < n - k$.
If it is the first update of p and there is no scan of p between $\tau_d^{p_0}$ and τ' then by definition of $Proc(v_0)^{\tau_d^{p_0}}$, p updates the cell with v_0. (3) holds. $L1$ may decreased, with (a) and the induction hypothesis, we get (1) at τ'.
If it is not its first update or there is a scan of p between $\tau_d^{p_0}$ and τ'. By induction hypothesis for every time ϕ, $\tau \geq \phi \geq \tau_d^{p_0}$, $L1(\phi) \subseteq L2(\phi)$, then with (b) for every time ϕ, $\tau \geq \phi \geq \tau_d^{p_0}$, there is at least $n-k+1$ values equal to v_0 in R. Then for all j, v_0 is j-critical at every time ϕ $\tau \geq \phi \geq \tau_d^{p_0}$. By Lemma 2, in the scan before the update at time τ', p keeps v_0 as proposal, then (3) holds. $L1$ may decreased, with (a) we get (1) at τ'.

From Lemma 4, we deduce the safety property of the algorithm:

Proposition 2. *No more than k values can be decided by the processes.*

Proof. Let p_0, $\tau_d^{p_0}$ and v_0 defined as before. At most $k - 1$ processes decide by times $\tau_d^{p_0}$. As there is $2(n-k)$ MWMR registers, by Lemma 4 (1) and Lemma 3, v_0 will always be the value of at least one cell in R then no process may decide a value different from v_0. Then no more than k values may be decided.

Now we prove the liveness property of the algorithm. Assuming obstruction freedom, there is a process, say p_1, that eventually is the only process taking steps. Then there is a time τ after which no other process takes steps. Notice that in particular, after time τ, only p_1 may modify cells in R. Let τ_1 be the time after τ, at which p_1 makes its first update of some cell, say i, in R with value r_1.

Lemma 5. *If after the update of $R[i]$, v is i-critical at time τ_1 then p_1 eventually decides v.*

Proof. Assume that after the update of $R[i]$ v is i-critical at time τ_1. By Lemma 1, v is the only value i-critical. Consider the following two cases:

- (a) the proposal value of p_1 is v, v being i-critical and p_1 has just written v in cell i, then $SM(v)^\tau \geq (n-k)+1$. An easy induction proves that v will be forever j-critical for any j. Hence p_1 never changes its proposal, and p_1 will write all cells in R until they all contain v, and then p_1 decides.
- (b) if the proposal value for p_1 is different from v then p_1 changes *prop* to v and updates again $R[i]$ but with v, then we are brought back to case (a).

Lemma 6. *If no value v is i-critical, eventually a value will be j-critical for all j.*

Proof. Let v be the proposal value of p_1. While there is no i-critical value, p_1 does not change *prop*, then it updates $R[i]$ with v and changes i to $(i+1) \bmod 2(n-k)$(Lines 13 or 16). Eventually $SM(v)$ will be greater than $n-k$ and v is j-critical for all j.

Then we have:

Proposition 3. *If some process p is eventually the only process to take steps, then p decides.*

Together with Proposition 1, we get:

Theorem 1. *The algorithm of Figure 1 solves obstruction-free k-set agreement with $2(n-k)$ MWMR registers.*

At one extreme, we deduce from this theorem that 2 MWMR-registers are enough to solve set agreement

Corollary 1. *There is an algorithm that solves obstruction-free set agreement with 2 MWMR registers.*

4 Remarks

4.1 Processes with a Known Small Set of Identities

Assume that each process $p \in \Pi = \{p_1, \ldots, p_n\}$ knows its identity p_i and the set Π, the number of registers may be reduced. For this:

- Processes $p_1, \cdots, p_{k-1}$ decide their own value.
- Processes $p_k, \cdots, p_n$ share $(n-k+1)$ MWMR-registers. As the number of registers is the same as the number of processes writing in these registers, each process in $\{p_k, \cdots, p_n\}$ may have its own SWMR register, then any obstruction-free algorithm solving consensus may be applied and at most one value is decided.

Hence at most k values are decided. In this way with only $(n - k + 1)$ registers we solve (obstruction-free) k-set agreement.

The point is the fact that here processes have identities and that the identities of participants of the consensus is known. If this set of identities is not known, only the set of possible identities that may be very large can be considered. For example each process may have an IP address as identity and the set of possible identities is very large even if, n, the number of processes participating to the k-agreement task is very small.

4.2 Lower Bound on the Number of Registers

In this section we give some partial results concerning the number of MWMR registers needed.

When $k < n/2$. When $k < n/2$, the proposed algorithm uses more than n MWMR-registers. In this case from [13] it is possible to simulate n $SWMR$ registers in such a way that each process is the single writer of one register. Then any obstruction-free consensus algorithm may be applied (e.g. [5,12]) to get consensus. We can deduce from this that at least when $k < n/2$, the proposed algorithm is not optimal concerning the number of registers needed.

Impossibility of Non Trivial Decision Task with One Shared Register. It is impossible to achieve $(n - 1)$-set agreement with one register. Moreover with one register the only decision task that can be solved obstruction-free is a trivial task in the following sense: the decision of a process is a function of its own input.

Proposition 4. *With one shared register, the only decision task that can be solved obstruction-free is a task where the output of a process is a function of its input.*

Proof. Consider a distributed algorithm that solves a decision task and assume that processes share only one register. Construct an execution e that is indistinguishable for each process to an execution where this process takes steps alone. The code of a process is a sequence of scan and update. Assume that the shared register is initialized to some value v. In e, each process executes its code and stops just before its first update, if exists. Consider the processes that run, decide and never update the register. These processes see only the initial value v in the register. Let p be one of these processes, e is indistinguishable from an execution where p is the only process. So the decision of p is a function of its input. Consider now processes that are just before their update. Let p be one of these processes. Now p runs alone. Its first step, an update, covers the value possibly written by some process. And p reads in the shared register the value that it has written. Then e is indistinguishable for p from an execution where p is the only process. So the decision of p is a function of its input.

From Proposition 4 and Corollary 1, we get a tight result concerning set agreement:

Theorem 2. *Set agreement is obstruction-free solvable if and only if there are at least two MWMR registers.*

Impossibility of $(n-2)$-Set Agreement with 2 Registers and 3 Processes

Proposition 5. *With 2 shared registers and 3 processes, it is impossible to implement $(n-2)$-set agreement.*

Proof. If a process never updates a register then we can construct an execution where it takes steps before any update and it can only decide it own initial value (due to the validity property of $(n-2)$-set agreement). So if we have 3 processes that never updates a register we cannot achieve $(n-2)$-set agreement. Consider that it is not the case. As $N >> n$ (in particular $N > 6$), there are at least 3 identities for which the first update is on $R[1]$ (else swap the two registers). Let a, b and c be these 3 processes. We construct now an execution. First processes a, b and c run until they are just before their first update. Then process b runs until it decides or it is ready to update $R[2]$, the second register. There are two cases:
Case 1: If b decides, this execution is indistinguishable for b from an execution where it runs alone, then it decides its own input value v_b. Now c runs alone (its update $R[1]$ covering the value written by b) and decides. This execution is indistinguishable for c from an execution where it runs alone, then it decides its own input value v_c contradicting the the agreement property of $(n-2)$-set agreement.
Case 2: b is ready to update $R[2]$. Now c runs alone, its update $R[1]$ covering the value written by b it runs until it decides. This execution is indistinguishable for c from an execution where it runs alone, then it decides its own input value v_c. Then process a takes a step: it updates $R[1]$. And b takes a step : it updates $R[2]$. Now in the shared registers there are no traces of c. Then b runs alone and decides. This execution is indistinguishable for b from an execution where only processes a and b take steps, then it decides the input value v_b or v_a, contradicting the agreement property of $(n-2)$-set agreement.

5 Conclusion

We have solved obstruction free k-set agreement with $2(n-k)$ MWMR registers. We have shown that this result is optimal for set agreement ($k = n-1$). But we think that for the other values of k, this result is not optimal. We do not know if it is possible or not to achieve $(n-2)$-set agreement with 2 registers when $n > 3$. We may think that in fact the algorithm in Figure 1 with 2 registers allows us to solve better than $(n-1)$-set agreement. Unfortunately, this is not the case. With our algorithm in Figure 1, there exists an execution with n processes that decides $n-1$ values. We give in Figure 2 one such execution with 4 processes. Our example can be easily generalized for n processes.

	a	b	c	d	R
initially	A	B	C	D	$(\perp, \perp)$
a:update(R[1],A)	A	B	C	D	(A,$\perp$)
a:scan()	A	B	C	D	(A,$\perp$)
a:update(R[2],A)	A	B	C	D	(A,A)
b:update(R[1],B)	A	B	C	D	(B,A)
a:scan()	B	B	C	D	(B,A)
b:scan()	B	A	C	D	(B,A)
b:update(R[1],A)	B	A	C	D	(A,A)
b:scan()	B	A	C	D	(A,A)
b: **decide A**					
c:update(R[1],C)	B	-	C	D	(C,A)
a:update(R[2],B)	B	-	C	D	(C,B)
c:scan()	B	-	B	D	(C,B)
a:scan()	C	-	B	D	(C,B)
c:update(R[1],B)	C	-	B	D	(B,B)
c:scan()	C	-	B	D	(B,B)
c: **decide B**					
d:update(R[1],D)	B	-	-	D	(D,B)
a:update(R[2],B)	B	-	-	D	(D,B)
a:scan()	D	-	-	D	(D,B)
a:update(R[2],D)	D	-	-	D	(D,D)
d:scan()	D	-	-	D	(D,D)
d: **decide D**					
a: scan()	D	-	-	-	(D,D)
a: **decide D**					

Fig. 2. Execution of $(n-1)$-set agreement algorithm with 4 processes

References

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. Journal of the ACM 40(4), 873–890 (1993)
2. Aspnes, J., Ellen, F.: Tight bounds for anonymous adopt-commit objects. In: Rajaraman, R., auf der Heide, F.M. (eds.) Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2011(Co-located with FCRC 2011), San Jose, CA, USA, June 4-6, pp. 317–324. ACM (2011)
3. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. Journal of the ACM 37(3), 524–548 (1990)
4. Attiya, H., Fouren, A.: Adaptive and efficient algorithms for lattice agreement and renaming. SIAM J. Comput. 31(2), 642–664 (2002)
5. Attiya, H., Guerraoui, R., Hendler, D., Kuznetsov, P.: The complexity of obstruction-free implementations. J. ACM 56(4) (2009)
6. Attiya, H., Welch, J.: Distributed Computing. Fundamentals, Simulations, and Advanced Topics. John Wiley & Sons (2004)
7. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: STOC, pp. 91–100. ACM Press (1993)

8. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: PODC, pp. 41–51. ACM Press (1993)
9. Borowsky, E., Gafni, E.: A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In: Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, PODC 1997, pp. 189–198. ACM Press, New York (1997)
10. Burns, J.E., Lynch, N.A.: Bounds on shared memory for mutual exclusion. Inf. Comput. 107(2), 171–184 (1993)
11. Chaudhuri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Information and Computation 105(1), 132–158 (1993)
12. Delporte-Gallet, C., Fauconnier, H.: Two Consensus Algorithms with Atomic Registers and Failure Detector Ω. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, pp. 251–262. Springer, Heidelberg (2008)
13. Delporte-Gallet, C., Fauconnier, H., Gafni, E., Rajsbaum, S.: Linear space bootstrap communication schemes. In: Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R.K., Sinha, P. (eds.) ICDCN 2013. LNCS, vol. 7730, pp. 363–377. Springer, Heidelberg (2013)
14. Delporte-Gallet, C., Fauconnier, H., Gafni, E., Rajsbaum, S.: Linear space bootstrap communication scheme. Technical report (2013)
15. Ellen, F., Fatourou, P., Ruppert, E.: The space complexity of unbounded timestamps. Distributed Computing 21(2), 103–115 (2008)
16. Fich, F.E., Herlihy, M., Shavit, N.: On the space complexity of randomized synchronization. J. ACM 45(5), 843–862 (1998)
17. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (1985)
18. Gafni, E.: Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony. In: Proceedings of the 17th Symposium on Principles of Distributed Computing (1998)
19. Gafni, E., Mostéfaoui, A., Raynal, M., Travers, C.: From adaptive renaming to set agreement. Theor. Comput. Sci. 410(14), 1328–1335 (2009)
20. Helmi, M., Higham, L., Pacheco, E., Woelfel, P.: The space complexity of long-lived and one-shot timestamp implementations. In: Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2011, pp. 139–148. ACM, New York (2011)
21. Herlihy, M.: Wait-free synchronization. ACM Transactions on Programming Languages and Systems 13(1), 123–149 (1991)
22. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: 23rd International Conference on Distributed Computing Systems, ICDCS 2003, Providence, RI, USA, May 19-22, pp. 522–529. IEEE Computer Society (2003)
23. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. Journal of the ACM 46(2), 858–923 (1999)
24. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann (2008)
25. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. In: Proceedings of the 25th ACM Symposium on Theory of Computing, pp. 101–110. ACM Press (May 1993)

26. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. on Computing 29, 1449–1483 (2000)
27. Styer, E., Peterson, G.L.: Tight bounds for shared memory symmetric mutual exclusion problems. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, PODC 1989, pp. 177–191. ACM, New York (1989)
28. Yang, J., Neiger, G., Gafni, E.: Structured derivations of consensus algorithms for failure detectors. In: Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, pp. 297–306 (1998)

Self-stabilizing Byzantine Resilient Topology Discovery and Message Delivery
(Extended Abstract)

Shlomi Dolev[1,⋆], Omri Liba[1], and Elad M. Schiller[2,⋆⋆]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
{dolev,liba}@cs.bgu.ac.il
[2] Department of Computer Science and Engineering,
Chalmers University of Technology, Goeteborg, Sweden
elad@chalmers.se

Abstract. Traditional Byzantine resilient algorithms use $2f + 1$ vertex-disjoint paths to ensure message delivery in the presence of up to f Byzantine nodes. The question of how these paths are identified is related to the fundamental problem of topology discovery. Distributed algorithms for topology discovery cope with a never ending task: dealing with frequent changes in the network topology and unpredictable transient faults. Therefore, algorithms for topology discovery should be self-stabilizing to ensure convergence of the topology information following any such unpredictable sequence of events. We present the first such algorithm that can cope with Byzantine nodes. Starting in an arbitrary global state, and in the presence of f Byzantine nodes, each node is eventually aware of all the other non-Byzantine nodes and their connecting communication links. Using the topology information, nodes can, for example, route messages across the network and deliver messages from one end user to another. We present the first deterministic, cryptographic-assumptions-free, self-stabilizing, Byzantine-resilient algorithms for network topology discovery and end-to-end message delivery. We also consider the task of r-neighborhood discovery for the case in which r and the degree of nodes are bounded by constants. The use of r-neighborhood discovery facilitates polynomial time, communication and space solutions for the above tasks. The obtained algorithms can be used to authenticate parties, in particular during the establishment of private secrets, thus forming public key schemes that are resistant to man-in-the-middle attacks of the compromised Byzantine nodes. A polynomial and efficient end-to-end algorithm that is based on the established private secrets can be employed in between periodical secret re-establishments.

⋆ Partially supported by Deutsche Telekom, Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant number 428/11), Cabarnit Cyber Security MAGNET Consortium, Grant from the Institute for Future Defense Technologies Research named for the Medvedi of the Technion, and Israeli Internet Association.

⋆⋆ Partially supported by the EC, through project FP7-STREP-288195, KARYON (Kernel-based ARchitecture for bsafetY-critical cONtrol), the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement 257007 and through the FP7-SEC-285477-CRISALIS project.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 42–57, 2013.

1 Introduction

Self-stabilizing Byzantine resilient topology discovery is a fundamental distributed task that enables communication among parties in the network even if some of the components are compromised by an adversary. Currently, such topology discovery is becoming extremely important where countries' main infrastructures, such as the electrical smart-grid, water supply networks and intelligent transportation systems are subject to cyber-attacks. Self-stabilizing Byzantine resilient algorithms naturally cope with mobile attacks [e.g., 1]. Whenever the set of compromised components is fixed (or dynamic, but small) during a period that suffices for convergence of the algorithm, the system starts demonstrating useful behavior following the convergence. For example, consider the case in which nodes of the smart-grid are constantly compromised by an adversary while local recovery techniques, such as local node reset and/or refresh, ensure the recovery of a compromised node after a bounded time. Once the current compromised set does not imply a partition of the communication graph, the distributed control of the smart grid automatically recovers. Self-stabilizing Byzantine resilient algorithms for topology discovery and message delivery are important for systems that have to cope with unanticipated transient violations of the assumptions that the algorithms are based upon, such as unanticipated violation of the upper number of compromised nodes and unanticipated transmission interferences that is beyond the error correction code capabilities.

The dynamic and difficult-to-predict nature of electrical smart-grid and intelligent transportation systems give rise to many fault-tolerance issues and require efficient solutions. Such networks are subject to transient faults due to hardware/software temporal malfunctions or short-lived violations of the assumed settings for the location and state of their nodes. Fault-tolerant systems that are *self-stabilizing* [2] can recover after the occurrence of transient faults, which can drive the system to an arbitrary system state. The system designers consider *all* configurations as possible configurations from which the system is started. The self-stabilization design criteria liberate the system designer from dealing with specific fault scenarios, risking neglecting some scenarios, and having to address each fault scenario separately.

We also consider Byzantine faults that address the possibility of a node to be compromised by an adversary and/or to run a corrupted program, rather than merely assuming that they start in an arbitrary local state. Byzantine components may behave arbitrarily (selfishly, or even maliciously) as message senders and as relaying nodes. E.g., Byzantine nodes may block messages, selectively omit messages, redirect message routes, playback messages, or modify messages. Any system behavior is possible, when all (or one third or more of) the nodes are Byzantine nodes. Thus, the number of Byzantine nodes, f, is usually restricted to be less than one third of the nodes [2, 3].

The task of *r-neighborhood network discovery* allows each node to know the set of nodes that are at most r hops away from it in the communication network. Moreover, the task provides information about the communication links attached to these nodes. The task *topology discovery* considers knowledge regarding the node's entire connected component. The r-neighborhood network discovery and network topology discovery tasks are identical when r is the communication graph radius.

This work presents the first deterministic self-stabilizing algorithms for r-neighborhood discovery in the presence of Byzantine nodes. We assume that every r-neighborhood cannot be partitioned by the Byzantine nodes. In particular, we assume the existence of at least $2f + 1$ vertex-disjoint paths in the r-neighborhood, between any two non-Byzantine nodes, where at most f Byzantine nodes are present in the r-neighborhood, rather than in the entire network. [1] Note that by the self-stabilizing nature of our algorithms, recovery is guaranteed after a temporal violation of the above assumption. When r is defined to be the communication graph radius, our assumptions are equivalent to the standard assumption for Byzantine agreement in general (rather than only complete) communication graphs. In particular the standard assumption is that $2f + 1$ vertex disjoint paths exist and *are known* (see e.g., [3]) while we present distributed algorithms to find these paths starting in an arbitrary state.

Related Work. Self-stabilizing algorithms for finding vertex-disjoint paths for at most two paths between any pair of nodes, and for all vertex-disjoint paths in anonymous mesh networks appear in [4] and in [5], respectively. We propose self-stabilizing Byzantine resilient procedures for finding $f + 1$ vertex-disjoint paths in $2f + 1$-connected graphs. In [6], the authors study the problem of spanning tree construction in the presence of Byzantine nodes. Nesterenko and Tixeuil [7] presented preliminary ideas for a *non-stabilizing* algorithm for topology discovery in the presence of Byzantine nodes. Awerbuch and Sipser [8] consider algorithms that were designed for synchronous static network and give topology update as an example. They show how to use such algorithms in asynchronous dynamic networks. Unfortunately, their scheme starts from a consistent state and cannot cope with transient faults or Byzantine nodes.

The problems of *Byzantine gossip* [9–14] and *Byzantine Broadcast* [15, 16] consider the dissemination of information in the presence of Byzantine nodes rather than self-stabilizing topology discovery. Non-self-stabilizing Byzantine resilient gossip in the presence of one selfish node is considered in [10, 12]. In [11] the authors study oblivious deterministic gossip algorithms for multi-channel radio networks with a malicious adversary. They assume that the adversary can disrupt one channel per round, preventing communication on that channel. In [13] the authors consider probabilistic gossip mechanisms for reducing the redundant transmissions of flooding algorithms. They present several protocols that exploit local connectivity to adaptively correct propagation failures and protect against Byzantine attacks. Probabilistic gossip mechanisms in the context of recommendations and social networks are considered in [14]. In [9] the authors consider rules for avoiding a combinatorial explosion in (non-self-stabilizing) gossip protocol. Note that deterministic and self-stabilizing solutions are not presented in [9–14]. Drabkin et al. [15] consider non-self-stabilizing broadcast protocols that overcome Byzantine failures by using digital signatures, message signature gossiping, and failure detectors. Our deterministic self-stabilizing algorithm merely use the topological properties of the communication graph to ensure correct message delivery to the application layer in the presence of message omission, modifications and Byzantine nodes.

[1] Section 4 considers cases in which r and an upper bound on the node degree, Δ, are constants. For these cases, we have $\mathcal{O}(n)$ disjoint r-neighborhoods. Each of these (disjoint) r-neighborhoods may have up to f Byzantine nodes, and yet the above assumptions about at least $2f + 1$ vertex-disjoint paths in the r-neighborhood, hold.

A non-self-stabilizing broadcasting algorithm is considered in [16]. The authors assume the restricted case in which links and nodes of a communication network are subject to Byzantine failures, and that faults are distributed randomly and independently. We note that our result can serve as a base for a compiler that convert non-stabilizing algorithm to a stabilizing algorithm. We facilitate communication among participants that enables (repeatedly) run of a non-stabilizing algorithm that copes with Byzantine processors, using the standard re-synchronization technique that is based on self-stabilizing Byzantine clock synchronization [2, 17].

Our Contribution. We present two cryptographic-assumptions-free yet secure algorithms that are deterministic, self-stabilizing and Byzantine resilient.

We start by showing the existence of deterministic, self-stabilizing, Byzantine resilient algorithms for network topology discovery and end-to-end message delivery. The algorithms convergence time is in $\mathcal{O}(n)$. They take in to account every possible path and requiring bounded (yet exponential) memory and bounded (yet exponential) communication costs. Therefore, we also consider the task of r-neighborhood discovery, where r is a constant. We assume that if the r-neighborhood of a node has f Byzantine nodes, there are $2f+1$ vertex independent paths between the node and any non-Byzantine node in its r-neighborhood. The obtained r-neighborhood discovery algorithm requires polynomial memory and communication costs and supports deterministic, self-stabilizing, Byzantine-resilient algorithm for end-to-end message delivery across the network. Unlike topology update, the proposed end-to-end message delivery algorithm establishes message exchange synchronization between end-users that is based on message reception acknowledgments. Detailed proofs appear in [18].

2 Preliminaries

We consider settings of a standard asynchronous system [cf. 2]. The system consists of a set, $N = \{p_i\}$, of communicating entities, chosen from a set, P, which we call *nodes*. The upper bound on the number of nodes in the system is $n = |P|$. Each node has a unique identifier. Sometime we refer to a set, $P \setminus N$, of nonexisting nodes that a false indication on their existence can be recorded in the system. A node p_i can directly communicate with its *neighbors*, $N_i \subseteq N$. The system can be represented by an undirected network of directly communicating nodes, $G = (N, E)$, named the *communication graph*, where $E = \{(p_i, p_j) \in N \times N : p_j \in N_i\}$. We denote N_k's set of indices by $indices(N_k) = \{m : p_m \in N_k\}$ and N_k's set of edges by $edges(N_j) = \{p_j\} \times N_j$.

The r-neighborhood of a node $p_i \in N$ is the connected component that includes p_i and all nodes that can be reached from p_i by a path of length r or less. The problem of r-neighborhood topology discovery considers communication graphs in which p_i's degree, δ_i, is bounded by a constant Δ. Hence, when the neighborhood radius, r, and the node degree, Δ, are both constants the number of nodes in the r-neighborhood is also bounded by a constant, namely by $\mathcal{O}(\Delta^{r+1})$.

We model the communication channel, $queue_{i,j}$, from node p_i to node $p_j \in N_i$ as a FIFO queuing list of the messages that p_i has sent to p_j and p_j is about to receive. When p_i sends message m, the operation send() inserts a copy of m to the queue $queue_{i,j}$ of the one destination p_j, such that $p_j \in N_i$. We assume that the number of messages in

transit, i.e., stored in $queue_{i,j}$, is at most $capacity$. Once m arrives, p_j executes receive and m is dequeued.

We assume that p_i is completely aware of N_i, as in [7]. In particular, we assume that the identity of the sending node is known to the receiving one. In the context of the studied problem, we say that node $p_i \in N$ is *correct* if it reports on its genuine neighborhood, N_i. A *Byzantine* node, $p_b \in N$, is a node that can send arbitrarily corrupted messages. Byzantine nodes can introduce new messages and modify or omit messages that pass through them. This way they can, e.g., disinform correct nodes about their neighborhoods, about the neighborhood of other correct nodes, or the path through which messages travel, to name a very few specific misleading actions that Byzantine nodes may exhibit. Note that our assumptions do not restrict system settings in which a *duplicitous Byzantine* node, p_b, reports about N_b differently to its correct neighbors. In particular, p_b can have $\{N_{b_1}, \ldots N_{b_{\delta_b}}\}$ reports, such that p_b's identity in N_{b_i} is different than the one in N_{b_j}, where δ_x is the degree of node p_x. One may use a set of non-duplicitous Byzantine nodes, $\{p_{b_1}, \ldots p_{b_\delta}\}$, to model each of p_b's reports. Thus, for a $2k+1$ connected graph, the system tolerates no more than $\lfloor k/\Delta \rfloor$ duplicitous Byzantine nodes, where Δ is an upper bound on the node degree.

We denote C and B to be, respectively, the set of correct and Byzantine nodes. We assume that $|B| = f$, the identity of B's nodes is unknown to the ones in C, and B is fixed throughout the considered execution segment. These execution segments are long enough for convergence and then for obtaining sufficient useful work. We assume that between any pair of correct nodes there are at least $2f+1$ vertex-disjoints paths. We denote by $G_c = (C, E \cap C \times C)$ the *correct graph* induced by the set of correct nodes.

Self-stabilizing algorithms never terminate [2]. The non-termination property can be easily identified in the code of a self-stabilizing algorithm: the code is usually a do forever loop that contains communication operations with the neighbors. An iteration is said to be complete if it starts in the loop's first line and ends at the last (regardless of whether it enters branches).

Every node, p_i, executes a program that is a sequence of *(atomic) steps*. For ease of description, we assume the interleaving model with atomic step execution; a single step at any given time. An input event can either be the receipt of a message or a periodic timer going off triggering p_i to send. Note that the system is totally asynchronous and the (non-fixed) node processing rates are irrelevant to the correctness proof.

The *state* s_i of a node p_i consists of the value of all the variables of the node (including the set of all incoming communication channels, $\{queue_{j,i} | p_j \in N_i\}$. The execution of a step in the algorithm can change the state of a node. The term *(system) configuration* is used for a tuple of the form $(s_1, s_2, \cdots, s_n)$, where each s_i is the state of node p_i (including messages in transit for p_i). We define an *execution* $E = c[0], a[0], c[1], a[1], \ldots$ as an alternating sequence of system configurations $c[x]$ and steps $a[x]$, such that each configuration $c[x+1]$ (except the initial configuration $c[0]$) is obtained from the preceding configuration $c[x]$ by the execution of the step $a[x]$. We often associate the notation of a step with its executing node p_i using a subscript, e.g., a_i. An execution R (run) is *fair* if every correct node, $p_i \in C$, executes a step infinitely often in R. Time (e.g. needed for convergence) is measured by the number of *asynchronous rounds*, where the first asynchronous round is the minimal prefix of the

execution in which every node takes at least one step. The second asynchronous round is the first asynchronous round in the suffix of the run that follows the first asynchronous round, and so on. The message complexity (e.g. needed for convergence) is the number of messages measured in the specific case of synchronous execution.

We define the system's task by a set of executions called *legal executions* (LE) in which the task's requirements hold. A configuration c is a *safe configuration* for an algorithm and the task of LE provided that any execution that starts in c is a legal execution (belongs to LE). An algorithm is *self-stabilizing* with relation to the task LE when every infinite execution of the algorithm reaches a safe configuration with relation to the algorithm and the task.

3 Topology Discovery

The algorithm learns about the neighborhoods that the nodes report. Each report message contains an ordered list of nodes it passed so far, starting in a source node. These lists are used for verifying that the reports are sent over $f + 1$ vertex-disjoint paths.

When a report message, m, arrives to p_i, it inserts m to the queue $informedTopology_i$, and tests the queue consistency until there is enough independent evidence to support the report. The consistency test of p_i iterates over each node p_k such that, p_k appears in at least one of the messages stored in $informedTopology_i$. For each such node p_k, node p_i checks whether there are at least $f + 1$ messages from the same source node that have mutually vertex-disjoint paths and report on the same neighborhood. The neighborhood of each such p_k, that has at least $f + 1$ vertex-disjoint paths with identical neighborhood, is stored in the array $Result_i[k]$ and the total number of paths that relayed this neighborhood is kept in $Count[k]$.

We note that there may still be nodes $p_{fake} \in P \setminus (N)$, for which there is an entry $Result[fake]$. For example, $informedTopology$ may contain f messages, all originated from different Byzantine nodes, and a message m' that appears in the initial configuration and supports the (false) neighborhood the Byzantine messages refer to. These $f + 1$ messages can contain mutually vertex-disjoint paths, and thus during the consistency test, a result will be found for $Result[fake]$. We show that during the next computations, the message m' will be identified and ignored. The $Result$ array should include two reports for each (undirected) edge; the two nodes that are attached to the edge, each send a report. Hence, $Result$ includes a set of directed (report) edges. The term *contradicting edge* is needed when examining the $Result$ set consistency.

Definition 1 (Contradicting edges). *Given two nodes, $p_i, p_j \in P$, we say that the edge (p_i, p_j) is* contradicting with the set $evidence \subseteq edges(N_j)$, *if* $(p_i, p_j) \notin evidence$.

Following the consistency test, p_i examines the $Result$ array for contradictions. Node p_i checks the path of each message $m \in informedTopology_i$ with source p_r, neighborhood $neighborhood_r$ and $Path_r$. If every edge (p_s, p_j) on the path appears in $Result[s]$ and $Result[j]$, then we move to the next message. Otherwise, we found a fake supporter, and therefore we reduce $Count[r]$ by one. If the resulting $Count[r]$ is smaller than $f + 1$, we nullify the r'th entry of the $Result$ array. Once all messages are processed, the $Result$ array consisting of the (confirmed) local topologies is the output. At the end, p_i forwards the arriving message, m, to each neighbor that does not appear

in the path of m. The message sent by p_i includes the node from which m arrived as part of the visited path contained within m.

The Pseudocode of Algorithm 1. In every iteration of the infinite loop, p_i starts to compute its preliminary topology view by calling $ComputeResults$ in line 2. Then, every node p_k in the queue $InformedTopology$, node p_i goes over the messages in the queue from head to bottom. While iterating the queue, for every message m with source p_k, neighborhood N_k and visited path $Path_k$, p_i inserts $Path_k$ to $opinion[N_k]$, see line 18. After inserting, p_i checks if there is a neighborhood $Neig_k$ for which $opinion[Neig_k]$ contains at least $f + 1$ vertex-disjoint paths, see line 19. When such a neighborhood is found, it is stored in the $Result$ array (line 19). In line 20, p_i stores the number of vertex disjoint paths relayed messages that contained the selected neighborhood for p_k. After computing an initial topology view (line 3), p_i removes non-existing nodes from the computed topology. For every message m in $InformedTopology$, node p_i aims at validating its visited path. In line 24, p_i checks if there exists a node p_k whose neighborhood contradicts the visited path of m. If such a node exists, p_i decreases the associated entry in the $Count$ array (line 25). This decrease may cause $Count[r]$ to be smaller than $f + 1$, in this case p_i considers p_k to be fake and deletes the local topology of p_k from $Result[r]$ (line 26). Upon receiving a message m, node p_i inserts the message to the queue, in case it does not already exist, and just moves it to the queue top in case it does. The node p_i now needs to relay the message p_i got to all neighbors that are not on the message visited path (line 9). When sending, p_i also attaches the node identifier, from which the message was received, to the message visited path.

- $Insert(m)$: Insert item m to the queue head.
- $Remove(Messagem)$: Remove item m from the queue.
- $Iterator()$: Returns an pointer for iterating over the queue's items by their residence order in the queue.
- $HasNext()$: Tests whether the Iterator is at the queue end.
- $Next()$ Returns the next element to iterate over.
- $SizeOf()$ Returns the number of elements in the calling set.
- $MoveToHead(m)$: Move item m to the queue head.
- $IsAfter(m, S)$: Test that item m is after the items $m' \in S$, where S is the queue item set.

Fig. 1. $Queue$: general purpose data structure for queuing items, and its operation list

Algorithm's Correctness Proof. We now prove that within a linear amount of asynchronous rounds, the system stabilizes and every output is legal. The proof considers an arbitrary starting configuration with arbitrary messages in transit that could be actually in the communication channel or already stored in p_j's message queue and will be forwarded in the next steps of p_j. Each message in transit that traverses correct nodes can be forwarded within less than $\mathcal{O}(|C|)$ asynchronous rounds. Note that any message that traverses Byzantine nodes and arrives to a correct node that has at least one Byzantine node in its path. The reason is that the correct neighbor to the last Byzantine in the path lists the Byzantine node when forwarding the message. Thus, f is at most the number of messages that encode vertex-disjoint paths from a certain source that are initiated or corrupted by a Byzantine node. Since there are at least $f + 1$ vertex-disjoint paths with no Byzantine nodes from any source p_k to any node p_i and since p_k repeatedly sends messages to all nodes on all possible paths, p_i receives at least $f + 1$ messages from p_k with vertex-disjoint paths.

Algorithm 1. Topology discovery (code for node p_i)

Input: $Neighborhood_i$: The ids of the nodes with which node p_i can communicate directly;
Output: $ConfirmedTopology \subset P \times P$: Discovered topology, which is represent by a directed edge set;
Variable $InformedTopology : Queue$, see Figure 1: topological messages, $\langle node, neighborhood, path \rangle$;
Function: $NodeDisjointPaths(S)$: Test $S = \{\langle node, neighborhood, path \rangle\}$ to encode at least $f + 1$ vertex-disjoint paths;
Function: $PathContradictsNeighborhood(k, Neighborhood_k, path)$: Test that there is no node $p_j \in N$ for which there is an edge (p_k, p_j) in the message's visited path, $path \subseteq P \times N$, such that (p_k, p_j) is contradicting with $Neighborhood_k$;

1 **while true do**
2 $Result \leftarrow ComputeResults()$
3 **let** $Result \leftarrow RemoveContradictions(Result)$
4 $RemoveGarbage(Result)$
5 $ConfirmedTopology \leftarrow ConfirmedTopology \cup (\bigcup_{p_k \in P} Result[k])$
6 **foreach** $p_k \in N_i$ **do send**$(i, Neighborhood_i, \emptyset)$ **to** p_k

7 **Upon Receive** $(\langle \ell, Neighborhood_\ell, VisitedPath_\ell \rangle)$ **from** p_j;
begin
8 $Insert(p_\ell, Neighborhood_\ell, VisitedPath_\ell \cup \{j\})$
9 **foreach** $p_k \in N_i$ **do if** $k \notin VisitedPath_\ell$ **then send**$(p_\ell, Neighborhood_\ell, VisitedPath_\ell \cup \{j\})$ **to** p_k

10 **Procedure:** $Insert(k, Neighborhood_k, VisitedPath_k)$;
begin
11 **if** $\langle k, Neighborhood_k, VisitedPath_k \rangle \in InformedTopology$ **then** $InformedTopology.MoveToHead(m)$
12 **else if** $p_k \in N \wedge Neighborhood_k \subseteq indices(N) \wedge VisitedPath_k \subseteq indices(N)$ **then** $InformedTopology.Insert(\langle k, Neighborhood_k, VisitedPath_k \rangle)$

13 **Function:** $ComputeResults()$;
begin
14 **foreach** $p_k \in P : \langle k, Neighborhood_k, VisitedPath_k \rangle \in InformedTopology$ **do**
15 **let** $(FirstDisjointPathsFound, Message, opinion[]) \leftarrow (false, InformedTopology.Iterator(), [\emptyset])$
16 **while** $Message.hasNext()$ **do**
17 $\langle \ell, Neighborhood_\ell, VisitedPath_\ell \rangle \leftarrow Message.Next()$
18 **if** $\ell = k$ **then** $opinion[Neighborhood_\ell].Insert(\langle \ell, Neighborhood_\ell, VisitedPath_\ell \rangle)$
19 **if** $FirstDisjointPathsFound = false \wedge NodeDisjointPaths(opinion[Neighborhood_\ell])$ **then** $(Result[k], FirstDisjointPathsFound) \leftarrow (Neighborhood_\ell, \mathbf{true})$
20 $Count[k] \leftarrow opinion[k][Result[k.SizeOf()$
21 **return** $Result$

22 **Function:** $RemoveContradictions(Result)$;
begin
23 **foreach** $\langle r, Neighborhood_r, VisitedPath_r \rangle \in InformedTopology$ **do**
24 **if** $\exists p_k \in P : PathContradictsNeighborhood(p_k, Result[k], VisitedPath_r) = \mathbf{true}$ **then**
25 **if** $Neighborhood_r = Result[r]$ **then** $Count[r] \leftarrow Count[r] - 1$
26 **if** $Count[r] \leq f$ **then** $Result[r] \leftarrow \emptyset$
27 **return** $Result$

28 **Procedure:** $RemoveGarbage(Result)$;
begin
29 **foreach** $p_k \in N$ **do**
30 **foreach** $m = \langle k, Neighborhood_k, VisitedPath_k \rangle \in InformedTopology : \{k\} \cup Neighborhood_k \cup VisitedPath_k \not\subseteq P \vee InformedTopology.IsAfter(m, opinion[k][Result[k]])$ **do** $InformedTopology.Remove(m)$

The FIFO queue usage and the repeated send operations of p_k ensure that the most recent $f + 1$ messages with vertex-disjoint paths in $InformedTopology$ queue are

uncorrupted messages. Namely, misleading messages that were present in the initial configuration will be pushed to appear below the new $f + 1$ uncorrupted messages. Thus, each node p_i eventually has the local topology of each correct node (stored in the $Result_i$ array). The opposite is however not correct as local topologies of non-existing nodes may still appear in the result array. For example, $InformedTopology_i$ may include in the first configuration $f + 1$ messages with vertex-disjoint paths for a non-existing node. Since after $ComputeResults$ we know the correct neighborhood of each correct node p_k, we may try to ensure the validity of all messages. For every message that encodes a non-existing source node, there must be a node p_ℓ on the message path, such that p_ℓ is correct and p_ℓ's neighbor is non-existing, this is true since p_i itself is correct. Thus, we may identify these messages and ignore them. Furthermore, no valid messages are ignored because of this validity check.

We also note that, since we assume that the nodes of the system are a subset of P, the size of the queue $InformedTopology$ is bounded. Lemma 1 bounds the needed amount of node memory (the proof details appear in [18]).

Lemma 1 (Bounded memory). *At any time, there are at most $n \cdot 2^{2n}$ messages in $InformedTopology_i$, where $p_i \in C$, $n = |P|$ and $\mathcal{O}(n \log(n))$ is the message size.*

r-neighborhood Discovery. Algorithm 1 demonstrates the existence of a deterministic self-stabilizing Byzantine resilient algorithm for topology discovery. Lemma 1 shows that the memory costs are high when the entire system topology is to be discovered. We note that one may consider the task of r-neighborhood discovery. Recall that in the r-neighborhood discovery task, it is assumed that every r-neighborhood cannot be partitioned by Byzantine nodes. Therefore, it is sufficient to constrain the maximal path length in line 9. The correctness proof of the algorithm for the r-neighborhood discovery follows similar arguments to the correctness proof of Algorithm 1.

4 End-to-End Delivery

We present a design for a self-stabilizing Byzantine resilient algorithm for the transport layer protocol that uses the output of Algorithm 1. The design is based on a function (named $getDisjointPaths()$) for selecting vertex-disjoint paths that contain a set of $f + 1$ correct vertex-disjoint paths. We use $getDisjointPaths()$ and ARQ (Automatic Repeat reQuest) techniques for designing Algorithm 2, which ensures safe delivery between sender and receiver.

Exchanging Messages Over $f + 1$ Correct Vertex-Disjoint Paths. We guarantee correct message exchange by sending messages over a polynomial number of vertex-disjoint paths between the sender and the receiver. We consider a set, $CorrectPaths$, that includes $f + 1$ correct vertex-disjoint paths. Suppose that $ConfirmedTopology$ (see the output of Algorithm 1) encodes a set, $Paths$, of $2f+1$ vertex-disjoint paths between the sender and the receiver. It can be shown that $Paths$ includes at most f incorrect paths that each contains at least one Byzantine node, i.e., $Paths \supseteq CorrectPaths$. As we see next, $ConfirmedTopology$ does not always encode $Paths$, thus, one needs to circumvent this difficultly.

The case of constant r and Δ. The sender and the receiver exchange messages by using all possible paths between them; feasible considering r-neighborhoods, where the neighborhood radius, r, and the node degree Δ are constants.

The case of constant f. For each possible choice of f system nodes, $p_1, p_2, \ldots p_f$, the sender and the reciter compute a new graph $G(p_1, p_2, \ldots p_f)$ that is the result of removing $p_1, p_2, \ldots p_f$, from G_{out}, which is the graph defined by the discovered topology, $ConfirmedTopology$. Let $\mathcal{P}(p_1, p_2, \ldots p_f)$ be a set of $f + 1$ vertex-disjoint paths in $G(p_1, p_2, \ldots p_f)$ (or the empty set when $\mathcal{P}(p_1, p_2, \ldots p_f)$ does not exists) and $Paths = \bigcup_{p_1,p_2,\ldots p_f} \mathcal{P}(p_1, p_2, \ldots p_f)$. The sender and the receiver can exchange messages over $Paths$, because $|Paths|$ is polynomial at least one choice of $p_1, p_2, \ldots p_f$, has a corresponding set $\mathcal{P}(p_1, p_2, \ldots p_f)$ that contains $CorrectPaths$, see [18].

The case of no Byzantine neighbors The procedure assumes that any Byzantine node has no directly connected Byzantine neighbor in the communication graph. Specifically, this polynomial cost solution considers the (extended) graph, G_{ext}, that includes all the edges in $confirmedTopology$ and *suspicious edges*. Given three nodes, $p_i, p_j, p_k \in P$, we say that node p_i considers the undirected edge (p_k, p_j) suspicious, if the edge appears as a directed edge in $ConfirmedTopology_i$ for only one direction, e.g., (p_j, p_k).

The extended graph, G_{ext}, may contain fake edges that do not exists in the communication graph, but Byzantine nodes reports on their existence. Nevertheless, G_{ext} includes all the correct paths of the communication graph, G. Therefore, the $2f + 1$ vertex-disjoint paths that exists in G also exists in G_{ext} and they can facilitate a polynomial cost solution for the message exchange task, see [18].

Fig. 2. Implementation proposals for the function $getDisjointPaths()$

Note that even though $2f + 1$ vertex-disjoint paths between the sender and the receiver are present in the communication graph, the discovered topology in $ConfirmedTopology$ may not encode the set $Paths$, because f of the paths in the set $Paths$ can be controlled by Byzantine nodes. Namely, the information about at least one edge in f of the paths in the set $Paths$, can be missing in $ConfirmedTopology$.

We consider the problem of relaying messages over the set $CorrectPaths$ when only $ConfirmedTopology$ is known, and propose three implementations to the function $getDisjointPaths()$ in Figure 2. The value of $ConfirmedTopology$ is a set of directed edges (p_i, p_j). An undirected edge is approved if both (p_i, p_j) and (p_j, p_i) appear in $ConfirmedTopology$. Other edges in $ConfirmedTopology$ are said to be suspicious. For each of the proposed implementations, we show in [18] that a polynomial number of paths are used and that they contain $CorrectPaths$. Thus, the sender and the receiver can exchange messages using a polynomial number of paths and message send operations, because each path is of linear length.

Ensuring Safe Message Delivery. We propose a way for the sender and the receiver, that exchange a message over the paths in $getDisjointPaths()$, to stop considering messages and acknowledgments sent by Byzantine nodes. They repeatedly send messages and acknowledgments over the selected vertex-disjoint paths. Before message or acknowledgment delivery, the sender and the receiver expect to receive each message and acknowledgment at least $(capacity \cdot n + 1)$ consecutive times over at least

$f + 1$ vertex independent paths, and by that provide evidence that their messages and acknowledgments were indeed sent by them.

We employ techniques for labeling the messages (in an ARQ style), recording visited path of each message, and counting the number of received message over each path. The sender sends messages to the receiver, and the receiver responds with acknowledgments after these messages are delivered to the application layer. Once the sender receives the acknowledgment, it can fetch the next message that should be sent to the receiver. The difficulty here is to guarantee that the sender and receiver can indeed exchange messages and acknowledgments between them, and stop considering messages and acknowledgments sent by Byzantine nodes.

The sender repeatedly sends message m, which is identified by $m.ARQLabel$, to the receiver over all selected paths. The sender does not stop sending m before it is guaranteed that m was delivered to the application layer of the receiving-side. When the receiver receives the message, the set $m.VisitedPath$ encodes the path along which m was relayed over. Before delivery, the receiver expects to receive m at least $(capacity \cdot n + 1)$ consecutive times from at least $f + 1$ vertex independent paths. Waiting for $(capacity \cdot n + 1)$ consecutive messages on each path, ensures that the receiver gets at least one message which was actually sent recently by the sender. Once the receiver delivers m to the application layer, the receiver starts to repeatedly acknowledge with the label $m.ARQLabel$ over the selected paths (while recording the visited path). The sender expects to receive m's acknowledgment at least $capacity \cdot n + 1$ consecutive times from at least $f + 1$ vertex independent paths before concluding that m was delivered to the application layer of the receiving-side.

Once the receiver delivers a message to the application layer, the receiver starts to repeatedly acknowledge the recently delivered message over the selected paths. In addition, the receiver also restarts its counters and the log of received messages upon a message delivery to the application layer. Similarly the sender count acknowledgments to the current label used, when the sender receives at least $capacity \cdot n + 1$ acknowledgments over $f + 1$ vertex-disjoint paths, the sender fetches the next message from the application layer, changes the label and starts to send the new message.

The Pseudocode of Algorithm 2. In every iteration of the infinite loop, p_i fetches $Message$, prepares $Message$'s label (line 3) and starts sending $Message$ over the selected paths, see the procedure $ByzantineFaultTolerantSend(Message)$. When p_i gets enough acknowledgments for $Message$ (line 4), p_i stops sending the current message and fetches the next. Upon receiving a message msg, node p_i tests msg's destination (line 6). When p_i is not msg's destination, it forwards msg to the next node on msg's intended path, after updating msg's visited path. When p_i is msg's destination, p_i checks msg's type (line 9). When msg's type is *Data*, p_i inserts the message payload and label to the part of the data structure associated with the message source, i.e., the sender, and the message visited path (line 10). In line 12, node p_i checks whether $f + 1$ vertex-disjoint paths relayed the message at least $capacity \cdot n + 1$ times, where $capacity$ is an upper bound on the number of messages in transit over a communication link. If so, p_i delivers the msg to the application layer (line 20), clears the entire data structure and finally sends acknowledgments on the selected paths until a new message is confirmed. Moreover, in line 21 we signal that we are ready to receive a

Algorithm 2. Self-stabilizing Byzantine resilient end-to-end delivery (p_i's code)

Interface: $FetchMessage()$: Gets messages from the upper layer. We denote by $InputMessageQueue$ the unbounded queue of all messages that are to be delivered to the destination;
Interface: $DeliverMessage(Source, Message)$: Deliver an arriving message to the higher layer. We denote by $OutputMessageQueue$ the unbounded queue of all messages that are to be delivered to the higher layer. We assume that it always contains at least the last message inserted to it;
Input: $ConfirmedTopology$: The discovered topology (represented by a directed edge set, see Algorithm 1);
Data Structure: Transport layer messages: $\langle Source, Destination, VisitedPath, IntentedPath, ARQLabel, Type, Payload\rangle$, where $Source$ is the sending node, $Destination$ is the target node, $VisitedPath$ is the actual relay path, $IntentedPath$ is the planned relay path, $ARQLabel$ is the sequence number of the stop-and-wait ARQ protocol, and $Type \in \{Data, ACK\}$ message type, where DATA and ACK are constant;
Variable $Message$: the current message being sent;
Variable $ReceivedMessages[j][Path]$: queue of p_j's messages that were relayed over path $Path$;
Variable $Confirmations[j][Path]$: p_j's acknowledgment queue for messages that were relayed over $Path$;
Variable $label$: the current sequence number of the stop-and-wait ARQ protocol;
Variable $Approved$: A Boolean variable indicating whether $Message$ was accepted at the destination;
Function: $NodeDisjointPaths(S)$: Test S, a set of paths, to encode at least $f+1$ vertex-disjoint paths;
Function: $FloodedPath(MessageQueue, m)$: Test whether m is encoded by the first $capacity \cdot n + 1$ messages in $MessageQueue$.;
Function: $getDisjointPaths(ReportedTopology, Source, Destination)$: Get a set of vertex-disjoint paths between $Source$ and $Destination$ in the discovered graph, $ReportedTopology$ (Figure 2).;
Function: $ClearQueue(Source)$: Delete all data in $ReceivedMessages[Source][*]$;
Function: $ClearAckQueue(Destination)$: Delete all data in $Confirmations[Destination][*]$;

1 **while true do**
2 $ClearAckQueue(Message.Destination)$
3 $(Message, label) \leftarrow (FetchMessage(), label + 1 \ modulo \ 3)$
4 **while** $Approved =$ **false do** $ByzantineFaultTolerantSend(Message)$

5 **Upon Receive** (msg) **From** p_j;
begin
6 **if** $msg.Destination \neq i$ **then**
7 $msg.VisitedPath \leftarrow msg.VisitedPath \cup \{j\}$
8 **send**(msg) **to next** $(msg.IntendedPath)$
9 **else if** $msg.Type = Data$ **then**
10 $ReceivedMessages[msg.Source][msg.VisitedPath].insert(\langle msg.Payload, msg.ARQLabel \rangle)$
11 **let** $Paths \leftarrow \{Path : FloodedPath(Confirmations[msg.Source][Path], msg)\}$
12 **if** $NodeDisjointPaths(Paths)$ **then**
13 $NewMesssage \leftarrow$ **true**
14 $Confirm(msg.Source, m.ARQLabel, m.Payload)$
15 **else if** $msg.Type = ACK$ **then**
16 **if** $label = msg.ARQLabel$ **then**
 $Confirmations[msg.Source][msg.VisitedPath].insert(\langle msg.Payload, msg.ARQLabel\rangle)$
17 **let** $Paths \leftarrow \{Path : FloodedPath(Confirmations[msg.Source][Path], \langle msg.Payload, msg.ARQLabel \rangle)\}$
18 **if** $NodeDisjointPaths(Paths)$ **then** $Approved \leftarrow$ **true**

19 **Function:** $Confirm(Source, ARQLabel, Payload)$;
begin
20 **if** $CurrentLabel \neq ARQLabel$ **then** $DeliverMessage(Source, Payload)$
21 $(CurrentLabel, NewMessage) \leftarrow (ARQLabel,$ **false**$)$
22 $ClearQueue(Source)$
23 **while** $NewMessage =$ **false do** $ByzantineFaultTolerantSend(\langle Source, ARQLabel, ACK, Payload\rangle)$

24 **Function:** $ByzantineFaultTolerantSend(\langle Destination, ARQLabel, Type, Payload\rangle)$;
begin
25 **let** $Paths \leftarrow getDisjointPaths(ConfirmedTopology, i, Destination)$
26 **foreach** $Path \in Paths$ **do** **send**$(\langle i, Destination, \emptyset, Path, ARQLabel, Type, Payload\rangle)$ **to first**$(Path)$

new message. When msg's type is ACK, we act almost as when the message is of type $Data$. When the condition in line 18 holds, we signal that the message was confirmed at the receiver by setting $Approved$ to be $true$, in line 18. We note that the code of Algorithm 2 considers only one possible pair of source and destination. A many-source to many-destination version of this algorithm can simply use a separate instantiation of this algorithm for each pair of source and destination.

Correctness Proof. We show that message delivery guarantees hold after a bounded convergence period. The proof is based on the system's ability to relay messages over $f + 1$ correct vertex-disjoint messages (Figure 2), and focuses on showing safe message delivery between the sender and the receiver. After proving that the sender fetches messages infinitely often, we show that within four such fetches, the message delivery guarantees hold; receiver-side delivers all of the sender's messages and just them. The proof in detail appears in [18]. Let us consider messages, m, and their acknowledgements, that arrive at least $(capacity \cdot n + 1)$ times over $f + 1$ vertex-independent paths, to the receiver-side, and respectively the sender-side, with identical payloads and labels. The receiver, and respectively the sender, has the *evidence* that m was *indeed sent by the sender,* and respectively, *acknowledged by the receiver.* The sender and the receiver *clear* their *logs* whenever they have such evidences about m. The proof shows that, after a finite convergence period, the system reaches an execution in which the following events reoccur: (Fetch) the sender clears its log, fetches message m, and sends it to the receiver, (R-Get) the receiver gets the evidence that m was indeed sent by the sender, (Deliver) the receiver clears its log, delivers m, and acknowledge it to the sender, and (S-Get) the sender gets the evidence that m was acknowledged by the receiver. Namely, the system reaches a legal execution.

First we prove that event Fetch occurs infinitely often, in the way of proof by contradiction. Let us assume (towards a contradiction) that the sender fetches message m and then never fetches another message m'. The sender sends m and counts acknowledgments that has m's label. According to the algorithm, the sender can fetch the next message, $m' \neq m$, when it has the evidence that m was indeed acknowledged by the receiver. The receiver acknowledges m's reception when it has the evidence that m was indeed sent by the sender. After nullifying its logs, the receiver repeatedly sends m's acknowledgments until it has evidences for other messages, m', that were indeed sent by the sender after m. By the assumption that the sender never fetches $m' \neq m$, we have that the receiver keeps on acknowledging m until $m' \neq m$ arrives from the sender. Therefore, m arrives from the sender to the receiver, and the receiver acknowledges m to the sender. Thus, a contradiction that the sender never fetches $m' \neq m$.

The rest of the proof shows that (eventually) between every two event of type Fetch, also the events R-Get, Deliver and S-Get occur (and in that order). We show that this is guaranteed within four occurrences of event Fetch. Following the fetch of each of the first three messages and before the next one, the sender must have evidence that the receiver executed event Deliver, i.e., clearing the receiver's log. Note that during convergence, this may surely be false evidence. Just before fetching a new message in event Fetch, the sender must clear its logs and reassign a label value, say, the value is 0. There must be a subsequent fetch with label 1, because, as explained above, event Fetch occurs (infinitely often). Since the sender clears its logs in event of Fetch, from

now on and until the next event Fetch, any corrupted message found in the sender's log must be of Byzantine origin. Therefore, the next time sender gets the evidence that m was acknowledged by the receiver, the receiver has truly done so. Note that between any such two (truthful) acknowledgments (with different labels), say with label, $1, 2, \ldots$, the receiver must execute event Deliver and clean its log, see Algorithm 2, line 22. Since the sender sends over $f+1$ correct paths, and the receiver's logs are clear, eventually the receiver will have evidence for the message with label 0. As corrupted messages originate only from Byzantine nodes and there are at most f such nodes, the receiver's log may not contain evidence for non-sender messages. To conclude, starting from the 4-th message, the receiver will confirm all of the sender's messages, and will not confirm non-sender messages.

5 Extensions and Conclusions

As an extension to this work, we suggest to combine the algorithms for r-neighborhood network discovery and the end-to-end capabilities in order to allow the use of end-to-end message delivery within the r-neighborhoods. These two algorithms can be used by the nodes, under reasonable node density assumptions, for discovering their r-neighborhoods, and, subsequently, extending the scope of their end-to-end capabilities beyond their r-neighborhood, as we describe in the following. We instruct further remote nodes to relay topology information, and in this way collect information on remote neighborhoods. One can consider an algorithm for studying specific remote neighborhoods that are defined, for example, by their geographic region, assuming the usage of GPS inputs; a specific direction and distance from the topology exploring node defines the exploration goal. The algorithm nominates $2f + 1$ nodes in the specific direction to return further information towards the desired direction. The sender uses end-to-end communication to the current $2f + 1$ nodes in the *front* of the current exploration, asks them for their r-neighborhood, and chooses a new set of $2f + 1$ nodes for forming a new front. It then instructs each of the current nodes in the current front to communicate with each node in the chosen new front, to nominate the new front nodes to form the exploration front.

To ensure stabilization, this interactive process of remote information collection should never stop. Whenever the current collection process investigates beyond the closest r-neighborhood, we concurrently start a new collection process in a pipeline fashion. The output is the result of the last finalized collection process. Thus, having a correct output after the first time a complete topology investigation is finalized.

In this work we presented two deterministic, self-stabilizing Byzantine-resilience algorithms for topology discovery and end-to-end message delivery. We have also considered an algorithm for discovering r-neighborhood in polynomial time, communication and space. Lastly, we mentioned a possible extension for exploring and communicating with remote r-neighborhoods using polynomial resources as well.

The obtained end-to-end capabilities can be used for communicating the public keys of parties and establish private keys, in spite of f corrupted nodes that may try to conduct man-in-the-middle attacks, an attack that the classical Public key infrastructure (PKI) does not cope with. Once private keys are established encrypted messages can

be forwarded over any specific $f + 1$ node independent paths, one of which must be Byzantine free. The Byzantine free path will forward the encrypted message to the receiver while all corrupted messages will be discarded. Since our system should be self-stabilizing, the common private secret should be re-established periodically.

References

[1] Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: 10th Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, pp. 51–59 (1991)
[2] Dolev, S.: Self-Stabilization. MIT Press (2000)
[3] Lynch, N.: Distributed Computing. Morgan Kaufmann Publishers (1996)
[4] Al-Azemi, F.M., Karaata, M.H.: Brief announcement: A stabilizing algorithm for finding two edge-disjoint paths in arbitrary graphs. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 433–434. Springer, Heidelberg (2011)
[5] Hadid, R., Karaata, M.H.: An adaptive stabilizing algorithm for finding all disjoint paths in anonymous mesh networks. Comp. Comm. 32(5), 858–866 (2009)
[6] Dubois, S., Masuzawa, T., Tixeuil, S.: Maximum metric spanning tree made byzantine tolerant. In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 150–164. Springer, Heidelberg (2011)
[7] Nesterenko, M., Tixeuil, S.: Discovering network topology in the presence of byzantine faults. IEEE Trans. Parallel Distrib. Syst. 20(12), 1777–1789 (2009), see errata via `http://vega.cs.kent.http://vega.cs.kent.edu/~mikhail/Research/topology.errata.html`
[8] Awerbuch, B., Sipser, M.: Dynamic networks are as fast as static networks (preliminary version). In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science (SFCS 1988), pp. 206–220. IEEE Computer Society (1988)
[9] Minsky, Y., Schneider, F.B.: Tolerating malicious gossip. Distributed Computing 16(1), 49–68 (2003)
[10] Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: Bar gossip. In: 7th Symposium on Operating Systems Design and Implementation, OSDI 2006, Berkeley, CA, USA, pp. 191–204. USENIX Association (2006)
[11] Dolev, S., Gilbert, S., Guerraoui, R., Newport, C.C.: Gossiping in a multi-channel radio network. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 208–222. Springer, Heidelberg (2007)
[12] Alvisi, L., Doumen, J., Guerraoui, R., Koldehofe, B., Li, H.C., van Renesse, R., Trédan, G.: How robust are gossip-based communication protocols? Operating Systems Review 41(5), 14–18 (2007)
[13] Burmester, M., Le, T.V., Yasinsac, A.: Adaptive gossip protocols: Managing security and redundancy in dense ad hoc networks. Ad Hoc Net. 5(3), 313–323 (2007)
[14] Fernandess, Y., Malkhi, D.: On spreading recommendations via social gossip. In: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2008), pp. 91–97. ACM (2008)

[15] Drabkin, V., Friedman, R., Segal, M.: Efficient Byzantine broadcast in wireless ad-hoc networks. In: Proceedings of IEEE International Conference on Dependable Systems and Networks (DSN 2005), pp. 160–169. IEEE Computer Society (2005), Self-stabilizing Byzantine Resilient 57

[16] Paquette, M., Pelc, A.: Fast broadcasting with byzantine faults. Int. J. Found. Comput. Sci. 17(6), 1423–1440 (2006)

[17] Awerbuch, B., Varghese, G.: Distributed program checking: a paradigm for building self-stabilizing distributed protocols (extended abstract). In: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (SFCS 1991), pp. 258–267. IEEE Computer Society (1991)

[18] Dolev, S., Liba, O., Schiller, E.M.: Self-stabilizing Byzantine resilient topology discovery and message delivery. CoRR abs/1208.5620 (2012)

FreeRec: An Anonymous and Distributed Personalization Architecture

Antoine Boutet[1], Davide Frey[1], Arnaud Jégou[1], Anne-Marie Kermarrec[1,2], and Heverson B. Ribeiro[1]

[1] INRIA Rennes, France
{antoine.boutet,davide.frey,arnaud.jegou,anne-marie.kermarrec, heverson.ribeiro}@inria.fr
[2] EPFL, Switzerland

Abstract. We present and evaluate FreeRec, an anonymous decentralized peer-to-peer architecture, designed to bring personalization while protecting the privacy of its users. FreeRec's decentralized approach makes it independent of any entity wishing to collect personal data about users. At the same time, its onion-routing-like gossip-based overlay protocols effectively hide the association between users and their interest profiles without affecting the quality of personalization. The core of FreeRec consists of three layers of overlay protocols: the bottom layer, RPS, consists of a standard random peer sampling protocol ensuring connectivity; the middle layer, PRPS, introduces anonymity by hiding users behind anonymous proxy chains, providing mutual anonymity; finally, the top clustering layer identifies for each anonymous user, a set of anonymous nearest neighbors. We demonstrate the effectiveness of FreeRec by building a decentralized and anonymous content dissemination system. Our evaluation by simulation and through extensive PlanetLab experiments show that FreeRec effectively decouples users from their profiles without hampering the quality of personalized content delivery.

1 Introduction

The Web 2.0 has transformed the way users interact with the Internet. Users are no longer pure consumers, but they now generate a large portion of the available content. As a result, personalized services have become a requirement for most online applications. While personalization and social applications greatly enhance user experience, they amplify the Internet's inherent privacy risks and concerns. For instance, personalization in a social application can lead to the revelation of potentially embarrassing information to friends, family, and colleagues. In addition, users publishing controversial or prohibited information on social platforms can easily be identified and located through their IP addresses.

The reason for the privacy risks associated with personalized services lies in their inevitable dependence on personal data. As another example, consider one of the most common forms of personalized services: recommendation. A common

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 58–73, 2013.

technology providing this service is user-based Collaborative Filtering (CF) [24]. This paradigm leverages interest similarities to identify correlations between the preferences of different users. While users are not generally aware of who else shares their own interests, their centralized implementation requires service providers to store accurate information about the interests of users. This clashes with the need to protect personal data.

Anonymity services provide an attractive way to overcome the privacy issues associated with personalized services. They hide the real identity (*i.e.* IP address) of a user through pseudonym (*e.g.* IP address of another node). Several such solutions are available on the Internet [1] and offer users the possibility to navigate anonymously behind a proxy. However, the use of a single proxy is vulnerable to adversaries that can observe traffic going in and out of the proxy. Distributed solutions, such as Tor [11] provide better guarantees. Nonetheless, they do not eradicate the concentration of personal data within the servers of a single provider. Decentralized personalization based on the P2P paradigm [5,8] addresses the issue of concentrated data while providing naturally scalability. Yet, they remain vulnerable to the presence of malicious users.

Clearly anonymity alone does not protect users privacy, nor does decentralization alone. In this work, we seek to address these issues by combining the benefits of decentralized personalization and anonymity. The result is FreeRec, an anonymous and distributed personalization architecture. Our solution implements a distributed user-based (CF) scheme through an anonymous and interest-based topology and uses the resulting overlay to recommend items to users. Unlike existing decentralized personalization platforms, FreeRec protects the interest profiles stored at every node by means of anonymous exchanges with other peers. This makes FreeRec a generic personalization architecture that can be leveraged to build a number of distributed applications that may benefit from recommendation services.

FreeRec builds anonymous chains of nodes by relying on three layers of gossip protocols providing mutual anonymity. A standard random-peer-sampling protocol provides nodes with the members of their anonymous chains. A second private peer-sampling protocol uses these chains to provide each node with an anonymous sample of the network. A top clustering layer implements a decentralized collaborative-filtering overlay by creating decentralized clusters of anonymous profiles. This layered architecture makes FreeRec self organizing and capable to adapt to the arrival and departure of nodes and to changes in the interests of users. We evaluate FreeRec using both simulation and a real deployment on PlanetLab. Our results on a news-personalization use case show that users are able to effectively receive and publish content even in presence of path failure with reasonable overhead.

2 System Model

We consider a decentralized user-based collaborative filtering (CF) system [5,8]. Such systems build interest-based overlay networks by clustering nodes according

to the similarity among their interest profiles[1]. This task relies on two protocols: a random peer sampling (RPS) and a clustering protocol (CLUSTERING). The RPS [26] protocol ensures connectivity by providing each node with a continuously changing random sample of the graph. This comes in the form of a view data structure: a list of references to other nodes. Each entry in a view consists of (*i*) a node's IP address and port, (*ii*) a profile describing the node's interests, and (*iii*) a timestamp indicating when the associated profile was generated. While the RPS allows nodes to continuously discover new nodes, the CLUSTERING protocol identifies, at each node, the k-nearest neighbors in term of interests, and ensures connectivity between the node and this neighborhood.

Periodically, each protocol selects the node in its view with the oldest timestamp and sends it a message containing its profile with half of its view for the RPS and its entire view in case of the clustering protocol (standard parameters [14,27]). In the RPS, the receiving node renews its view by keeping a random sample of the union of its own view and the received one. In the clustering protocol, it computes the union of its own and the received view, and selects the nodes whose profiles are closest to its own according to a similarity metric. Several similarity metrics have been proposed [25], we use the Jaccard index in this paper.

3 FreeRec

Our anonymous personalization architecture extends the model described in Section 2 to achieve anonymity by executing gossip exchanges through onion-like encryption chains: the *proxy chains*. The *proxy chain* of a node n is a sequence starting with n and containing a random number of other nodes – from $\mathsf{ch}_{\mathsf{min}}$ to $\mathsf{ch}_{\mathsf{max}}$ – as depicted in Figure 1. We refer to node n, the first node in the chain, as its *initiator*. The last, p, is the chain's *proxy*, (or n's proxy), while the remaining ones are *intermediate* nodes. Messages can travel along the chain in two directions: *forward*, from the initiator to the proxy; or *backward*, the other way around. The proxy acts as a placeholder for n, hiding n's identity in all the gossip exchanges that include n's interest profile.

Proxy chains effectively hide the very fact two nodes are communicating. Two nodes n and m can learn their respective profiles without knowing their respective identities. Moreover, their profiles are hidden from all other nodes in the chains. A node n that wishes to send a message to another node builds a sequence of encryption layers around it, including the corresponding routing information. Each of the nodes along its proxy chain removes one of these layers and sends the inner encrypted layer to the next hop indicated in the message. The process continues until the destination node's proxy. At this point, the message goes through the destination chain in the backward direction using routing information and encryption keys maintained by each node in the chain. Each of these, starting from the proxy, adds one encryption layer and routes the message until it arrives at the destination node, which removes all the layers.

[1] We use the term node to refer both to a user and to her machine.

3.1 Chain-Based Routing

We now present the data structures that allow nodes to build, maintain, and use proxy chains.

Chain and Message Keys. The onion-like encryption process outlined above relies on three types of keys: two sets of public/private key pairs, and one set of secret keys. First, each node, n, maintains a key pair, (K_n, k_n)[2], called *message key pair*. Nodes use it to send and receive encrypted messages through proxy chains to and from any other node while preventing the proxies and the other chain nodes from accessing the content of this communication.

Each node also maintains a second key pair: the *chain key pair*, (C_n, c_n). While the message key pair hides the content of a message from the nodes in the chain, the chain key pair makes it possible to construct the onion-like encryption layers when traversing the chain in the forward direction.

Finally, each node, n, generates and dispatches a secret key, s_i^n, to each node, i, in its own proxy chain. Nodes use this key to add onion layers to messages that travel along the chain in the backward direction, *i.e.* towards n. The use of onion-like encryption in the forward and backward directions causes messages to change at each hop, thus preventing external observers from recognizing the messages in a proxy chain. We summarize the roles of the three types of keys in Table 1, and provide details about their distribution in Section 3.3.

Data Structures and Routing IDs. To route messages along proxy chains we use a combination of source and hop-by-hop routing. Each node maintains information about the members of its own proxy chain in a chainTable. This data structure is essentially a list: each entry consists of the identifier of a node, and of its associated public chain key. The information in the table allows the initiator of a chain to encrypt messages in onion layers.

The destination proxy, however, cannot use source routing to reach the destination node: a node may in fact act as a proxy or an intermediate node in multiple proxy chains. To route backwards along the chain, we therefore use a set of routingIds as depicted in Figure 2. For routing purposes, all the nodes in a chain could use the same routingId to identify their next hops. However, this would easily allow colluding nodes to verify if they are part of the same chain. We therefore associate a unique (with high probability) routingId with each link in a chain. The proxy routingId (e.g. p_a and p_b in Figures 1 and 2) serves as a pseudonym for the destination node, while the remaining ones (r_{ij} in the figures) enable backward routing on the destination chain.

Nodes store the routingIds of the chains they belong to in a routingTable. With reference to Figure 2, let node p be a proxy in the chain of node b. p's routingTable contains an entry indexed by b's proxy routingId (p_b in the figure). This entry contains (i) p's secret key for the chain (s_p^b), (ii) the identifier of the previous node in the chain (v), (iii) the public chain key of v (C_v), and (iv)

[2] We use uppercase characters for public keys and lowercase for private or secret keys.

the ROUTINGID of the link between p and v (r_{pv}). Intermediate chain nodes also have an analogous entry in their routing tables, but indexed by the ROUTINGID of the link to the next node in the chain. Node v therefore has an entry indexed by r_{pv} and containing (i) v's secret key for the chain (s_v^b), (ii) the identifier of z, (iii) z's public chain key (C_z), and (iv) the ROUTINGID of the link to z (r_{zv}).

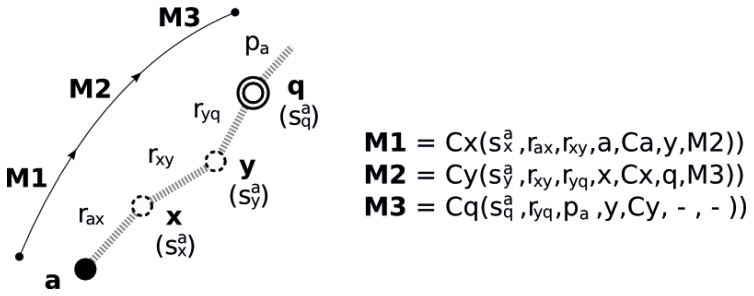


Fig. 1. Proxy chain creation

3.2 FreeRec Three-Layer Architecture

Our goal in building proxy chains is to enable the architecture described in Section 2 to operate anonymously. To make this possible, we replace the two protocols of Section 2 with a three-layer architecture. We introduce a RPS protocol layer, which provides each node with a sample of the network from which to choose the members of its proxy chain. The RPS operates like a normal peer sampling protocol with one addition: it associates each node n with the information required for creating the chains. This comprises only the node's IP address, its public chain key C_n, and a timestamp. Interest profiles do not appear in the RPS views: they are protected by the anonymous PRPS layer.

The PRPS (Proxied Random Peer Sampling) uses the information provided by the RPS to build a proxy chain for each node. It then exploits these chains in gossip exchanges thereby providing each node with a random sample of anonymous nodes. In doing this, it also allows nodes to learn about the necessary information to route messages anonymously to other nodes. Consider a PRPS view containing an entry for node b. The entry does not include b's IP address and port. Rather, it is identified by b's proxy ROUTINGID (p_b). In addition, it contains the IP address and port of b's proxy (p), p's public chain key (C_p), b's public message key (K_b), and b's interest profile.

PRPS views allow nodes to learn about the anonymous information referring to another node without being able to associate it with the node's precise identity. Nodes exchange views like in a standard RPS. However, they channel all view exchanges through their proxy chains. The PRPS thus replaces the RPS protocol of the architecture of Section 2, thus enabling anonymous profile exchanges.

The PRPS serves as a basis for the top layer of our architecture: a CLUSTERING protocol, like the one in Section 2. However, unlike in Section 2, the clustering

Table 1. Data structures maintained on a node v, followed by p and preceded by z in b's chain

(C_n, c_n)	Chain key pair of node n	c_n private key, C_n public key
(K_n, k_n)	Message key pair of node n	k_n private key, K_n public key
s_n^x	Secret key generated by node x and shared with node n	
RPS	Random peers sampling	@ip, timestamp, C_n
PRPS	Random proxies sampling	@ip, timestamp, ROUTINGID, C_n, profile, K_n
CLUSTERING	Interest-based neighborhoods	@ip, timestamp, ROUTINGID, C_n, profile, K_n
RT	ROUTINGTABLE $[r_{pv}]$	s_v^b, z, C_z, r_{zv}

protocol also performs all its view exchanges using the proxy chains built by the PRPS layer. This allows our architecture to build decentralized personalized services in a completely anonymous manner.

3.3 Protocol Details

In the remainder of this section, we provide additional details about how the PRPS protocol manages chains and encrypted routing.

Building Proxy Chains. A node a can start building its proxy chain once its RPS view is filled with a random set of nodes. Specifically, a first determines how many other nodes should be in its chain by extracting a random number k from $\mathsf{ch}_{\mathsf{min}}$ to $\mathsf{ch}_{\mathsf{max}}$ included. Then it extracts k nodes from its RPS view and it sets the first extracted node as a proxy p and the remaining ones (if any) as intermediate nodes i in the order they were extracted. a builds a create-chain message as described on Figure 1.

The message consists of concentric *onion* layers. Each layer is a CREATECHAIN message encrypted with the chain key of one of the nodes that will constitute the chain. The innermost message, $M3$ in the figure, is encrypted with the proxy's chain key and contains (i) the proxy's secret key (s_q^a), (ii) the proxy ROUTINGID for the chain (p_a in the figure), and (iii) the ROUTINGID for the link between the proxy and the last intermediate node (r_{yq} in the figure), (iv) the previous node's IP address and port (y), and (v) its public chain key (C_y).

After creating $M3$, the initiator creates a message for the last intermediate node in the chain (y in the figure). This message contains (i) the previously encrypted message for the proxy ($M3$), (ii) the next node's IP address (the IP address of the proxy q in this case) and port (q), (iii) node y's secret key (s_y^a), (iv) the ROUTINGID of the link between y and the next node in the chain (r_{yq}), (v) the ROUTINGID of the link between y and the previous node in the chain (r_{xy}), (vi) the previous node's IP address and port (x), and (vii) its public chain key (C_x). The initiator encrypts $M2$ with y's chain key and then it repeats the process by adding a layer for each of the nodes it selected for its chain, the last of these

being the one closest to the initiator itself, x in the figure. The initiator then sends the outermost message to this node initiating the chain-creation process.

Each node receiving a CREATECHAIN message decrypts it and uses its content to update the information in its routing table. It then forwards the encrypted inner-layer message to the next node in the chain, which operates analogously. The proxy performs the same operations except that it does not forward the message further. If a chain node is already part of another chain with the same ROUTINGID, it replies with an error message to the initiator, which will recreate the chain using a different ROUTINGID.

Sending Messages through Chains. FreeRec achieves mutual anonymity: when two nodes exchange messages, both the sender and the receiver are anonymous. Nodes use their proxy chains to send and receive encrypted messages as part of the PRPS and CLUSTERING protocols. Consider the example in Figure 2. Node a is sending a message m to a node with proxy p (not knowing b's id), public message key K_b, and proxy ROUTINGID p_b. Node a will have discovered this node, which happens to be b, through PRPS or CLUSTERING exchanges. As a result, the association between b's identity and p, K_b or p_b is unknown to a as well as to every other node in the system. The process unfolds as follows.

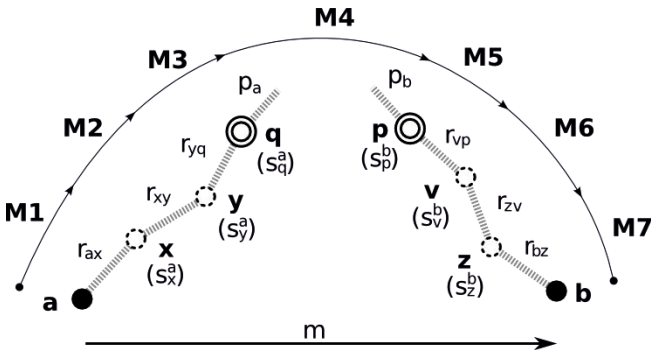


Fig. 2. Message exchange between nodes a and b: a knows b's profile, the identity of p, but not the identity of b. Node b knows a's profile, the identity of q, but not the identity of a. Nodes in the chain cannot access a's or b's profile.

First, a encrypts m using the destination node's public message key yielding $K_b(m)$. Then it prepares the first layer of its onion message. Specifically, a includes $K_b(m)$, and the destination's proxy incoming ROUTINGID, p_b in the figure. Then it encrypts the resulting message with the destination proxy's public chain key, yielding $M4$ in the figure. Node a continues the creation of the onion message by adding one layer from each of the nodes in its own chain, starting from the proxy. The first of these layers, $M3$ is encrypted with the proxy's public chain key, and contains both $M4$ and the address of $M4$'s target: the destination node's proxy. The subsequent one, $M2$ contains $M3$ and the address of $M3$'s

target, q in the figure. In general, consider a node n that is followed by a node m in a proxy chain. The corresponding onion layer will be encrypted with n's public chain key and will contain the IP address and port of m together with the immediate inner onion layer encrypted with m's public chain key. In the case of the figure, the outermost onion layer ($M1$) will be encrypted with node x's public chain key and will contain $M2$ and the IP address and port of node y.

After creating $M1$, node a sends it to x, which starts peeling off the first layer. It first decrypts the message using its private chain key and then forwards the contained encrypted message ($M2$) to the node indicated in $M1$ (y). Upon receiving the corresponding onion layer, each node in the chain proceeds analogously until the source node's proxy (q) forwards the innermost layer ($M4$) to the destination's proxy (p). This completes the first part of the routing process.

The destination's proxy (p) initiates the second part. It decrypts $M4$ and retrieves its content: a ROUTINGID, p_b, and an encrypted message for the destination node ($K_b(m)$). p first looks up p_b in its routing table and it retrieves (i) the associated secret key (s_p^b), (ii) the address and port of the previous node in the destination chain (v), and (iii) the ROUTINGID of the link leading to this node (r_{vp}). It then encrypts $K_b(m)$ using the retrieved secret key, yielding ($s_p^b(K_b(m))$). Finally it builds a message containing $s_p^b(K_b(m))$, and the ROUTINGID of the link to the previous node in the destination chain (r_{vp}). It encrypts this message using v's public chain key, yielding $M5$, and sends it to v.

When v receives $M5$, it decrypts it using its private chain key and retrieves $s_p^b(K_b(m))$ and the ROUTINGID of its link to p (r_{vp}). It looks up this ROUTINGID in its routing table and retrieves its own secret key s_v^b, the IP address and port of the previous node in the chain (z), z's public chain key, and the ROUTINGID of the link leading to it (r_{zv}). Node v encrypts $s_p^b(K_b(m))$ using s_v^b and places it in a message together with the retrieved ROUTINGID. It then further encrypts this message with z's public chain key and sends it to z. This process repeats at each of the intermediate nodes in the chain. Each adds an onion layer by encrypting the content of the message with its secret key and then wraps the result into a message with the routing information for the previous node in the chain.

When the destination node (b) receives the final message, it first decrypts it using its private chain key. Then it starts peeling off each of the onion layer added by the nodes in its proxy chain. To do so, it uses the secret keys it stored in its CHAINTABLE, starting with the one associated with the first intermediate node. After decrypting the layer added by its proxy, it obtains $K_b(m)$, which it further decrypts using its own private message key, ultimately retrieving the original message m.

Initialization. For this process to work, the source of a message must not only have built its proxy chain, but it must also have the necessary information about the destination node. This consists of the destination node's public message key (K_b), and of its proxy's IP address, public chain key (C_p), and ROUTINGID (p_b).

During normal operation, nodes obtain this information through PRPS exchanges. However, this poses a problem during initialization when the PRPS view of a node is still empty.

Consider a node n with an empty PRPS view. The first time n establishes a proxy chain, it sends a PRPS view containing only its information to all the nodes in its RPS view. The corresponding messages go through the proxy chain of n until its proxy and then go directly to their targets. Consider a target node t receiving one such message. If t is a proxy for another node m, then it forwards the message to m along m's proxy chain. Otherwise t caches the message until it becomes a proxy for some other node. When a node m receives the initialization message forwarded by its proxy, it adds its content to its PRPS view.

In principle, the target node t could also add the information received from n to its own PRPS view. However, this would weaken the protocol's anonymity guarantees. An attacker n could send its entry to only one target node t. If it subsequently received a message from a proxy p, it could conclude that p is likely to be the proxy of t.

Changing Proxy. Nodes change their proxy chains periodically. This provides several benefits. It sustains anonymity over time by limiting the impact of attackers that may corrupt a node's proxy. It provides protection from attackers that may guess a node's keys. Moreover, it allows a node to react to path failures in its chain as a result of churn.

To change proxy chain, a node repeats the chain creation process every $t1$ time units. Once it has established a new anonymous path, it informs all the nodes in its PRPS and CLUSTERING view of its new proxy. To keep track of these changes, all proxies and intermediate nodes associate a timer $t2$ with each of the entries in their routing tables. When $t2$ expires, they delete the corresponding entry. Nodes choose the timer value so that $t2 > t1 + \delta$ where δ is an upper bound on the time required to create a chain.

After a node has set up a new chain, it initiates a PRPS exchange with all nodes in its PRPS view and a CLUSTERING exchange with all those in its CLUSTERING view. A node that receives a fresher PRPS entry with the same proxy ROUTINGID as an existing entry (*i.e.* entry pointing to the same destination node) updates this entry with the new proxy identifier, proxy chain key, message key, and profile.

An important side effect of changing proxies is that the minimum length of the chain $\mathsf{ch}_{\mathsf{min}}$ should be at least as large as 1. If $\mathsf{ch}_{\mathsf{min}} = 0$, then a node n would be its own proxy with probability $1/\mathsf{ch}_{\mathsf{max}}$. An attacker could easily exploit the fact that this is significantly larger than the probability of choosing a random proxy. For $\mathsf{ch}_{\mathsf{min}} \geq 1$, a node that serves in n's proxy chain for several times could still observe that n appears as a previous chain node more often than others. Yet, inferring this information would require n to choose the attacker as the first node in its chain for several times. This makes the attack for $\mathsf{ch}_{\mathsf{min}} \geq 1$ very unlikely to succeed in practice.

4 Evaluation

4.1 Experimental Setup

We evaluated FreeRec by simulating its behavior and by deploying its implementation on PlanetLab in the context of a news-personalization use case. We combine FreeRec with a gossip-based dissemination protocol to recommend news items to a population of users. A user interest profile contains the news items she received and liked. When a user generates an item or expresses a positive opinion on a received item, she forwards it to her neighborhood in FreeRec's anonymous interest-based topology. Gossip frequency in all protocols is set to one per simulation cycle and of one every 2s in PlanetLab.

Dataset. We use a real dataset: we conducted a survey on around 250 news items (selected randomly from a set of RSS feeds on various topics). We exposed the item list to around 100 colleagues and relatives and gathered their reactions (like/dislike) to each news item. This provided us with a small but real dataset of users exposed to exactly the same news items. To scale our system, we generated 5 instances of each user and news item in the experiments. The resulting dataset gathers 1235 news items for 530 users. We inject each item into the system at a random time instant by selecting a random source node.

Metrics. We evaluate FreeRec along two metrics of performance and quality. Firstly we measure the overhead of the system in terms of the network traffic it generates. For simulations, we compute the total number of sent messages, the number of messages which have not reached its destination due to message loss and the number of hops for messages. For our PlanetLab deployment, we instead measure the average consumed bandwidth and the latency to receive a message. Secondly, to assess the impact of FreeRec on the quality of the recommendation, we compute *recall* and *precision*. Both measures are in $[0, 1]$. For an item, a recall of 1 means that all interested users have received the item. Yet, this measure does not account for spam since a trivial way to ensure a maximum recall is to send all news items to all users. This is precisely what precision accounts for. A precision of 1 means that the news item has reached only the users that are interested in it. An important challenge in information retrieval is to provide a good trade-off between these two metrics. This is expressed by the F1-Score, defined as the harmonic mean of precision and recall [25].

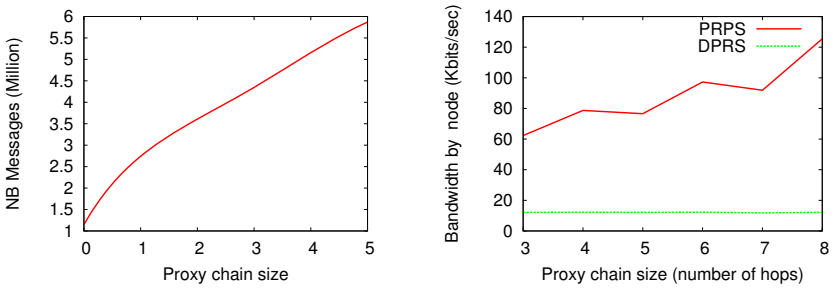
$$Precision = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{reached\ \ users\}|}$$

$$Recall = \frac{|\{interested\ \ users\} \cap \{reached\ \ users\}|}{|\{interested\ \ users\}|}$$

$$F1 - Score = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

4.2 Results

Overhead. We start by considering the overhead of the proxy chain in terms of number of messages. Clearly the longer the chain, the more anonymous the system. This cost is a function of the length of the proxy chain: the more the intermediate nodes in the chain, the higher the cost. Figure 3 depicts the number of messages according to the size of the proxy chain with a neighborhood fixed to 25. Results (Fig. 3a) show that a chain with only one proxy without intermediate nodes (*i.e.* size=2) brings a three-fold increase in the number of messages with respect to a chain-less system (size=0). This is because a message needs to go through two proxies (*i.e.* 3 hops) to reach its destination. Further adding intermediate nodes in the proxy chain proportionally increases the number of hops and the number of messages. Fig. 3b shows the overhead in PlanetLab of the two protocols RPS and PRPS in terms of bandwidth consumption. We observe that the RPS overhead remains stable regardless of the size of the proxy chains for RPS exchanges carry only information about chain keys while PRPS carries the encrypted messages. For this reason, the cost of PRPS increases linearly with the length of the chain.[3]



(a) Simulation (b) Deployment

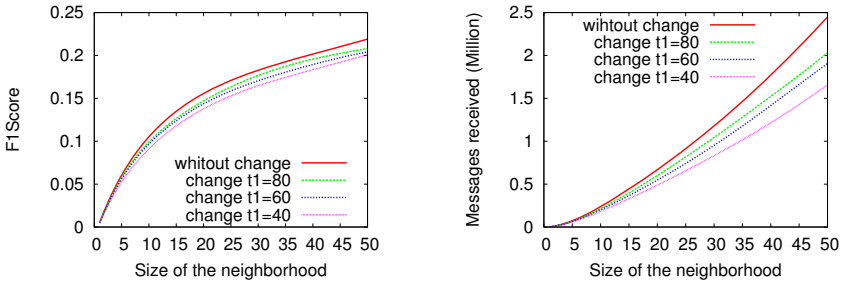
Fig. 3. Overhead according to the size of the proxy chain, in function of number of messages and bandwidth consumption for simulation and PlanetLab deployment

The Impact of Proxy Changes. To remain anonymous over time, nodes periodically renew their proxy chains. After setting up a new chain, a node advertises the information about its new proxy through PRPS and CLUSTERING exchanges. However, propagating this information takes time and some nodes only learn about the new proxy after several cycles. During this interval, a node that is unaware of the proxy change will send its messages to the old proxy. Consequently, messages will correctly go through the source node's possibly-new proxy chain, but they will reach the destination node's old proxy chain. If any of the nodes in this chain has already removed the corresponding entries from its routing table, it will silently discard the message.

[3] CLUSTERING (not shown) has a similar behaviour as PRPS with a bandwidth consumption exactly twice as much as that of the PRPS due to the larger gossip size.

As explained in Section 3.3, nodes remove entries from their routing tables δ time units after the creation of the new chain. During this time, the nodes in the old chain can still forward information backwards towards the chain owner. This leaves some time for the propagation of the new chain's information, but it does not eliminate the possibility of losing messages. Figure 4 evaluates the impact of this aspect in the context of our news-dissemination testbed as a function of the size of the CLUSTERING view, with $\delta = 10$ cycles.

Figure 4a shows that the impact of message loss on the F1-Score is very limited. When nodes change proxy every 80 cycles (*i.e.* t1=80), performance is almost indistinguishable from the stable case where nodes keep the same proxy over the whole experiment. When the chain changes more frequently (smaller values of $t1$) the percentage loss in F1-Score is slightly higher, but it remains lower than 10%. Figure 4b completes these results by comparing the number of sent messages with those that are actually received. Clearly message loss increases with the frequency of proxy changes. When nodes change proxies every 40 cycles (*i.e.* three times in the experiment) the number of lost messages is one fourth of the total number of messages.



(a) F1-Score (b) Message loss

Fig. 4. F1-Score and received messages for various $t1$ values (Simulations)

Latency. Figure 5 analyzes latency in our PlanetLab deployment (PL). The plot shows the time required by the PRPS protocol to establish a proxy chain, and by a message exchange that uses the chains both on the source and on the destination side. In the case of chain creation (CC), latency results from key generation, encryption/decryption operations, and message transmission. In the case of message exchanges (ME), there are only encryption/decryption operations and message transmission; yet messages have to travel for twice as many hops as in the case of chain creation.

The time required to create the proxy chain increases significantly with its size, while time required for exchanging messages increases only slightly. Moreover, creating the chain takes approximately two to three times as long as forwarding a message (40s vs 15s with 8-hop chains), even though forwarded messages have to travel for twice as many hops. This clearly shows that latency results mainly from computational cost. To understand the reasons for this seemingly poor

performance, we ran the same test by instantiating all the nodes on a local server (LS). In this case, both operations complete in less than 3s, and exchanging messages does take longer than creating chains. This confirms that the high latency exhibited in a PlanetLab setting results mainly from long processing times when performing cryptographic operations on overloaded machines.[4]

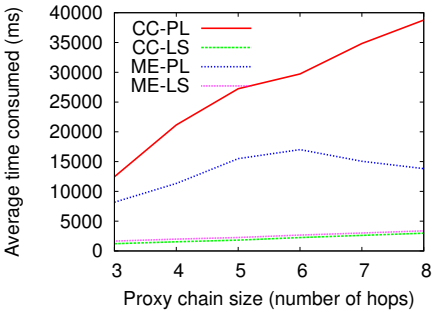


Fig. 5. Latency of chain creation and message forwarding (deployment)

5 Related Work

While personalization greatly enhances user experience, it raises privacy risks and concerns. Several collaborative-filtering approaches [20,21] have tried to preserve the privacy of sensitive data address by applying randomized masking and distortion techniques to user profiles. However, [12,16] show that privacy-sensitive information can be separated from such random distortion. To overcome this limitation, [7] uses noise that is not random but depends on the interest of users. This limits the amount of information exchanged between users to coarse-grained user profiles that only reveal the least sensitive information. Other decentralized approaches such as [9,10,19] exploit homomorphic encryption in a P2P environment. [13], in turn, addresses privacy by trust where participants exchange information and profile only across trusted content producer/consumer pairs. [2] proposes a differentially private protocol to measure the similarity between profiles. While differential privacy provides a strong notion of privacy, [18] highlights its important trade-off between privacy and accuracy.

A number of authors have proposed to address privacy by means of anonymity. Some, like [4] achieve receiver anonymity using group communication primitives like broadcasting, multicasting, and flooding. Others [11] focus on sender anonymity and relay messages from a node along a single anonymous path formed by nodes within the infrastructure.

[4] PlanetLab machines are notoriously overloaded, and the proximity of the SIGCOMM deadline might have resulted in even higher load.

Onion routing belongs to the latter group. It uses chains of router nodes that pack messages into onions: recursively encrypted data structures that contain the necessary routing information at each layer. When receiving an onion, a router removes a layer by decrypting it with its private key. At this point, it discovers either that it is the destination of the message, or the identity of the next router in the onion's forwarding path. . Tor [11] uses this model but cannot be readily integrated with decentralized personalization services.

Some authors have already suggested the integration of gossip and anonymous services. The work in [23] uses gossip protocols to improve the robustness of trust-based overlays to provide privacy-preserving data dissemination. More precisely, it creates and maintains additional anonymous links on top of an existing social overlay. Similarly, [22] relies on gossip protocols to supports confidential communications and private group membership. This solution leverages existing multi-hops paths to circumvent network limitations such as NAT and firewalls to form anonymous channel. Neither however combines anonymity with personalization. Gossple [5] does this to some extent and builds a fully decentralized anonymous collaborative network. Its gossip-on-behalf protocol hides the association between a user and her profile. Yet, in Gossple, a proxy controls some of the node's data structures. This is a significant disadvantage if the proxy wishes to censor specific information. In FreeRec, on the other hand, a proxy can at most drop messages randomly as it has no way to access their content.

Other works on gossip-based protocols have focused on tolerating byzantine faults such as BAR gossip [17], the secure peer sampling [15], Brahms [6] or PuppetCast [3]. In this work, we do not consider that nodes can act as active adversary by operating maliciously in the protocol. In case of malicious nodes cheating in the protocol, FreeRec could leverage one of these solutions to tolerate byzantine nodes. Finally, some authors [29,28] have suggested to address churn by replacing each onion router with a group of nodes. Such a technique could easily be integrated with our solution.

6 Conclusions

We presented FreeRec, a decentralized architecture for building anonymous personalized services. FreeRec equips nodes with bidirectional onion-routing-like proxy chains that allow nodes to exchange their interest profiles without ever revealing their identities. FreeRec's core consists of three layers of gossip protocols. The bottom one is a standard random-peer-sampling protocol that provides nodes with the necessary information to build their proxy chains. The middle layer, the prps, constitutes the main contribution of this work and is an augmented RPS protocol: it builds and maintains proxy chains and uses them to provide each node with a continuously changing anonymous sample of the network. The top layer completes the picture by providing each node with a cluster of anonymous interest profiles that most closely resemble its own.

References

1. Anonymous surfing solution, `http://anonymouse.org/`
2. Alaggan, M., Gambs, S., Kermarrec, A.-M.: BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom filters. In: Richa, A.W., Scheideler, C. (eds.) SSS 2012. LNCS, vol. 7596, pp. 202–216. Springer, Heidelberg (2012)
3. Bakkerand, A., van Steen, M.: Puppetcast: A secure peer sampling protocol. In: EC2ND (2008)
4. Bansod, N., Malgi, A., Choi, B.K., Mayo, J.: Muon: Epidemic based mutual anonymity in unstructured p2p networks. Comput. Netw. (2008)
5. Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A.-M., Leroy, V.: The gossple anonymous social network. In: Gupta, I., Mascolo, C. (eds.) Middleware 2010. LNCS, vol. 6452, pp. 191–211. Springer, Heidelberg (2010)
6. Bortnikov, E., Gurevich, M., Keidar, I., Kliot, G., Shraer, A.: Brahms: byzantine resilient random membership sampling. In: PODC (2008)
7. Boutet, A., Frey, D., Guerraoui, R., Jegou, A., Kermarrec, A.-M.: Privacy-preserving distributed collaborative filtering. In: Activity Report (2013)
8. Boutet, A., Frey, D., Guerraoui, R., Jegou, A., Kermarrec, A.-M.: Whatsup decentralized instant news recommender. In: IPDPS (2013)
9. Canny, J.: Collaborative filtering with privacy. In: SP (2002)
10. Canny, J.: Collaborative filtering with privacy via factor analysis. In: SIGIR (2002)
11. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: USENIX Security Symposium (2004)
12. Huang, Z., Du, W., Chen, B.: Deriving private information from randomized data. In: SIGMOD (2005)
13. Isaacman, S., Ioannidis, S., Chaintreau, A., Martonosi, M.: Distributed rating prediction in user generated content streams. In: RecSys (2011)
14. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: Gossip-based peer sampling. TOCS (2007)
15. Jesi, G.P., Montresor, A., van Steen, M.: Secure peer sampling. Comput. Netw. (2010)
16. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: ICDM (2003)
17. Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: Bar gossip. In: OSDI (2006)
18. Machanavajjhala, A., Korolova, A., Sarma, A.D.: Personalized social recommendations: accurate or private. In: VLDB (2011)
19. Miller, B.N., Konstan, J.A., Riedl, J.: Pocketlens: toward a personal recommender system. TOIS (2004)
20. Polat, H., Du., W.: Privacy-preserving collaborative filtering using randomized perturbation techniques. In: ICDM (2003)
21. Polat, H., Du, W.: Svd-based collaborative filtering with privacy. In: SAC (2005)
22. Schiavoni, V., Riviere, E., Felber, P.: Whisper: Middleware for confidential communication in large-scale networks. In: ICDCS (2011)
23. Singh, A., Urdaneta, G., van Steen, M., Vitenberg, R.: Robust overlays for privacy-preserving data dissemination over a social graph. In: ICDCS (2012)
24. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in Artificial Intelligence (2009)
25. van Rijsbergen, C.J.: Information retrieval. Butterworth (1979)

26. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management (2005)
27. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1143–1152. Springer, Heidelberg (2005)
28. Zhu, Y., Hu, Y.: Tap: A novel tunneling approach for anonymity in structured p2p systems. In: ICPP (2004)
29. Zhuang, L., Zhou, F., Zhao, B.Y., Rowstron, A.: Cashmere: resilient anonymous routing. In: NSDI (2005)

Establishing Efficient Routes between Personal Clouds

Ercan Ucan and Timothy Roscoe

Systems Group,
Department of Computer Science, ETH Zurich

Abstract. We address the problem of establishing efficient routes between nodes in disjoint peer-to-peer overlay networks, motivated by the case of personal overlays, each consisting of an ensemble of fixed, mobile, and virtual devices belonging to an individual user. We argue that the problem of route optimization between such systems is different from both routing between single hosts and inter-domain internet routing – in particular, scale and heterogeneity play a significant role, and the peer networks may wish to hide their topology for privacy reasons. We show that there is a significant tradeoff between efficiency and the degree of network information exposed to one peer network by the other, and present an approach that allows users to flexibly advertise desired information about their networks to one another. In this paper, we focus on optimizing the routes for latency and infer the potential to do the same for various other metrics such as bandwidth, monetary cost and energy consumption.

1 Introduction

In this paper, we address the problem of establishing efficient routes between a pair of personal overlay networks, where each network has incomplete knowledge of the other's topology. We are motivated by the scenario of *personal clouds* [21], ensembles of devices (phones, tablets, PCs, rented virtual machines, etc.) owned by single users, and which interact in a peer-to-peer fashion as an alternative to centralized cloud services such as Facebook, Dropbox and Apple's iCloud.

The problem we address is as follows: how can we establish efficient routes between pairs of nodes in two such personal clouds based on the current network state and user preferences? We argue that this problem is new, and different from both routing between single hosts, and inter-domain internet routing, for several reasons.

Firstly, a number of different routing metrics (latency, bandwidth, monetary cost per byte, energy budget for an object transfer, etc.) are important, sometimes simultaneously. In addition, these properties are likely to change frequently.

Secondly, the overlay nodes themselves, and the available connectivity between them, are highly diverse in these metrics: a 3G wireless link has very different cost, power consumption, and latency from a wired Ethernet, for example. Moreover, some nodes (such as phones) may have limited resources.

Thirdly, the small scale of the networks involved (around 10 devices per user) allows more computationally sophisticated reasoning about the best route for a given operation, exploiting detailed information about the diverse set of options available.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 74–88, 2013.

Finally, information about the set of devices in a user's personal cloud – their location, address, type, connectivity, and even in some cases existence – is potentially sensitive data the user may not wish to fully share with peers.

We consider this an important problem to tackle within the context of peer-to-peer device ensembles, not least because resource usage matters a great deal. Money is a scarce resource, and metered 3G/2G data connections or virtual machines rented from cloud providers can incur significant monetary cost, depending on usage. Time is a scarce resource, and a careless data transfer and routing approach in a personal data replication system can overload a single node or a single link for transferring data, even though the transfer could be carried out in a more time-efficient way using other links. Energy is also a scarce resource: a Nokia N900 on a 3G connection sending a file at 150 kbit/s draws 375 mA, and when receiving at 200 kbit/s draws 275 mA [15]. This is more power than consumed by continuously playing an MP3, activating the camera with a preview image, or vibrating the phone battery continuously. Efficient use of resources is critical.

Our contributions in this paper are threefold. First, we introduce and motivate the problem of inter-overlay routing in personal clouds. Second, we show simulation and early system implementation and deployment results which illustrate a significant trade-off between route efficiency between overlays and the degree of information exposed to one peer network by the other. Third, we present our work on an approach that allows users to flexibly advertise information about their networks to one another and optimize routes for metrics such as latency, bandwidth, monetary cost and energy.

In the next section, we present the background and related work. In Section 3 we further motivate the problem using concrete scenarios. In Section 4, we present our initial approach to tackle the problem. Section 5 describes the initial implementation of our research prototype. We present a preliminary evaluation of our approach via simulations in Section 6. In Section 7, we present our real system implementation and experimental results. Finally, we conclude and discuss the ongoing and future work in Section 8.

2 Background and Related Work

Our work on personal clouds is inspired by work on personal storage systems [19, 21–23] which aim to support personal data and content-based partial replication, without the need for centralized online services. In these systems, a user generates new data items by taking photos and videos, downloading music and documents. These items are replicated across the devices according to a system policy, and vary in size from a few kilobytes up to a few gigabytes. Most of the devices in the personal cloud offer multiple options for communication, such as Ethernet, WiFi, UMTS, Bluetooth, and USB.

Perspective [22] is a storage system designed for home devices. Cimbiosys [19] is designed for users to be able to selectively distribute data across their devices by associating content filters through opportunistic peer-to-peer synchronization. Eyo [23] is a personal media collections storage system. Anzere [21] is a personal storage system aimed for policy-based replication showing how to flexibly replicate data in response to a complex, user-specified set of policies in a dynamically changing environment. All of

these systems encounter the inter-overlay routing problem as soon as two instances need to exchange data. Despite influential naming and architecture work in this area such as UIA [8], as far as we know the problem of enabling efficient routes and optimizing data object transfers among different personal clouds has received little attention to date.

Dexferizer [26] presents an approach to optimizing the transfer of data objects *within* a user's collection of computation devices, subject to a variety of user-defined quality metrics such as cost, power consumption, and latency. Dexferizer employs techniques from declarative networking together with application-defined transfer policies and priorities to select appropriate transfer mechanisms and schedules.

We are also influenced by ideas from large-scale network architectures. SCAFFOLD [9] aims at better support for widely-distributed services, and argues that rather than today's host-based unicast, the main abstraction of future networks should be service-based anycast. Content-Centric Networking (CCN) [11] proposes replacing the host-to-host communication scheme of the Internet, arguing that *named data* is a better abstraction for today's network applications than *named hosts*. Similarly, DONA [13] proposes a service-centric approach, arguing that most Internet usage is data retrieval and service access, and this requires clean-slate design of Internet naming and name resolution. The eXpressive Internet Architecture (XIA) [1] argues that elevating only one principal type above others hinders communication between other types of principals and, thus, the evolution of the network. Instead, XIA provides native support for multiple principal types.

Our work is related to the problem of cross-layer visibility [12]. Strong layer abstractions hinder network management tasks, failure diagnosis, and ultimately the reliability of the network [3]. A similar observation has been made Plutarch [5] and Metanet [28]. Plutarch suggests making heterogeneity among different networks explicit through the notion of *contexts* which are linked via *interstitial functions*. Metanet proposes a new architectural object called a *region* as a first-class component of future networks. Our approach is based on similar general observations.

Research on MANETs has greatly contributed to the problem of routing in personal computing environments. MANET routing protocols [4, 6, 18] cannot rely on a centralized component or infrastructure, and must quickly react to device and link failures, changes in topology, and network partitions. MANETs can also be seen as an extension to the Internet: if a node in the MANET has Internet connectivity, it can function as a gateway [7, 24]. However, handling heterogeneity does not appear to be a primary focus of MANETs.

Our research is also related to the work on Content Distribution Networks (CDNs) [10, 16, 27], which aim to address similar problems, in a different context. CDNs aim to choose the *best* replica to deliver data to end users. In this paper, we have a focus on enabling selective advertisement of the network information, depending on the trust level, between the peer overlay networks that are owned by different users. Moreover, our work is in a smaller and more heterogeneous context as compared to CDNs.

The approach we have developed in tackling the problem is partly inspired by BGP [20]. However, BGP addresses a significantly larger and different problem scenario that involves large networks acting as traffic carriers.

3 Motivation and Challenges

In this section, we first motivate the problem further with three use-cases, and outline the design space, trade-offs and the challenges involved.

3.1 Use Cases

As a first example, consider the scenario in which Alice meets Bob in a cafe and they want to exchange a short movie file of 100MB. The file is initially replicated on Alice's phone, home computer and office PC. Bob wants to add this item to his collection so that he can watch it later. The phone is running on a 3G connection whereas the home computer and the office PC are on faster broadband connections. In such a scenario, ideally, the phone should be simply an initiator of the data transfer: the file should be moved between machines with better connectivity.

As a second example, consider the scenario illustrated in Figure 1, where Alice and Bob want to make their devices (phone and laptop) talk to each other. One option is to establish a communication through the cloud. Another option which may not always be possible is that, whenever the two devices happen to be on the same WiFi or Bluetooth network (even though TCP/IP over Bluetooth is not very mainstream) and they are aware of each other's available network interfaces, they may be able to establish a much more efficient route. Hence, exposing more network information can significantly improve routing.

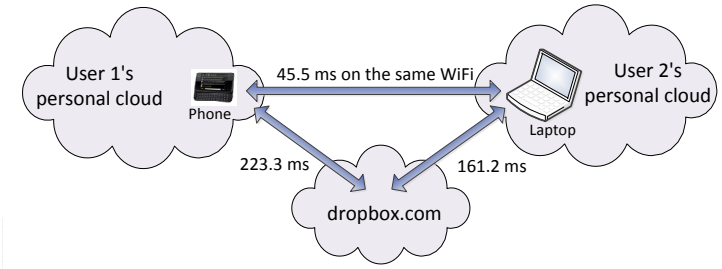


Fig. 1. Exposing more network information can significantly improve routing

As a third example, imagine that Alice wants to receive a large movie file from Bob who keeps two replicas of the movie, one on his home computer and one on his cloud storage. If Alice happens to own cloud storage from the same provider as Bob and if Bob exposes this information about his network to Alice, then they may be able to perform a very fast copy of the large file in the cloud. Alternatively, Alice could simply flag the file as *shared with Bob* on the cloud storage, without actually having the data to be moved from one location to another. Of course, in practice, the realization of sharing on the cloud can mean a variety of things such as sharing a URL (*e.g.*, in Dropbox's case). Moreover, it may also require implementation of additional mechanisms/interfaces depending on the cloud provider's mode of operation.

3.2 Design Space, Trade-offs and Challenges

As we show later in this paper, there is a trade-off between how much information the users expose about their personal clouds to each other, and the efficiency of subsequent routes and transfers. The more information is available across the peer networks about topology, node locations and capacities, and current network conditions, the better the resulting connectivity can be. In addition, here are some of the technical challenges involved in solving this problem:

- Dynamic nature of connectivity: many of the devices used in a user's personal cloud are mobile. These devices do not have permanent connectivity to the rest of the cloud.
- Routing challenges (NATs, firewalls, mobile devices, etc.): the current Internet architecture (that is, based on end-hosts) poses challenges for routing and data transfer among multiple personal clouds.

4 Design

We start this section by first describing the scenario we target, with disjoint peer overlays and their respective network information and routes. We then illustrate the problem of establishing efficient routes, and present our initial approach to tackle this problem.

4.1 Setting

We can characterize the challenge we address in inter-overlay routing as follows:

- Initially the two disjoint networks do not know any information about each other regarding the nodes they contain, the internal network topology, connectivity interfaces, item distribution, the current state of the network, etc.
- There may be multiple routes and multiple connectivity interfaces between the nodes.
- Links and the topology of the network are dynamic. We might or might not have the schedules of when certain links will appear or disappear.
- The network links have associated costs. This may be monetary cost, power, bandwidth or latency. Moreover, this information may not be known to other peer overlays.
- For privacy reasons, users may want to expose only certain parts of their network topology to each other.

4.2 General Approach: Selective Advertisement

The overall idea of our approach is inspired by BGP advertisements. We aim to provide the users of personal clouds with the means to flexibly advertise their network information to the users of other peer personal clouds, as they wish, depending on the network, transfer circumstances and user preferences.

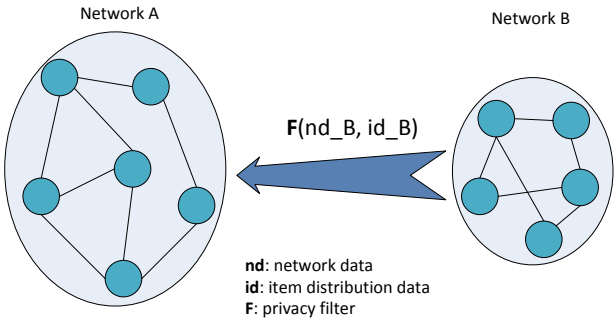


Fig. 2. Overview of our approach at an abstract level

Figure 2 illustrates at an abstract level, the general approach of advertising network information between personal clouds. The main idea is that the user (owner of network B in this case) applies his own *privacy filter* to the data of his network and sends this information to the peer overlay. Here we describe further the elements that are shown in the illustration.

- *nd_B*: is the network data that belongs to the network B. It consists of the information such as devices (their hostnames/IP address, port numbers to connect to), the internal network topology, link status information (ping latency, bandwidth estimation etc.). We focus on this type of information in the context of this paper.
- *id_B*: is the item data belonging to network B. Item data consists of the tuples showing where an item is replicated within the personal cloud. This information becomes relevant in the context of scheduling and optimizing object transfers.
- *F*: is the privacy filter applied by the owner of network B. This filter can be adjusted differently by the users of the systems depending on the trust level to each other.

4.3 Calculating Efficient Routes between Two Networks

In this section, with the help of an example scenario shown in Figure 3, we describe how we calculate efficient routes between two networks using network advertisements and employing Dijkstra's shortest path routing algorithm in a variant of link-state routing. The figure shows two disjoint networks, network A and B, with their representative weights to represent the network latency. At this point, we do not claim that these topologies and the numbers we show are entirely realistic. Our aim here is to rather have a means to illustrate our approach and to have an initial reference point for our simulations.

Following are the steps we use for the detection and the calculation of efficient routes between the members of two disjoint personal clouds via flexible advertisements of the network information:

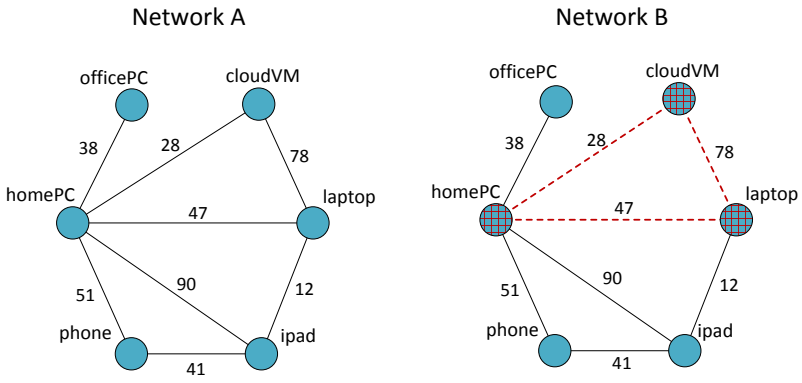


Fig. 3. The example network scenario used in our simulations

1. Network B sends a network advertisement to network A. An advertisement message consists of a list of reachable nodes and topology information between these advertised nodes. In this example, the network B may choose to advertise information about its 3 nodes (homePC, cloudVM and laptop) and part of the connectivity information among them. The information about a node in this case consists of a combination of its hostname/IP address (or a gateway) and the port number to use in order to reach that node. The topology information consists of the edges between the devices together with the weights.
2. Network A then adds the network information it received from network B into its own network knowledge base.
3. Network A tries to establish connections (preferably as many as possible) between each of its nodes and the nodes advertised by the network B. Successfully formed connections are also added as edges to the topology graph of the Network A.
4. Network A then calculates all-pairs-shortest-paths using Dijkstra's algorithm on this extended set of nodes and edges.

If a node is not exposed from the network B, the shortest paths to that node are calculated with the assumption that the exposed nodes of the network B will act as network gateways to the non-exposed node. At the end of the last step, each node in network A knows the shortest paths to the advertised nodes of network B.

5 Implementation

The work we present in this paper is done in the context of a broader project: Anzere [21], a data storage and replication system that we are developing, aimed for personal clouds, integrating personal computers, mobile phones, tablets and virtual machines acquired on demand from cloud providers such as Amazon EC2 and Planetlab.

Anzere is a system in which the objects are replicated according to declarative replication policies specified by the users. The policies written in Anzere do not need to refer to specific devices, but are based on device properties. Anzere obtains its replication actions by solving a constrained cost-based optimization problem derived from the set of replication policies. Anzere employs a replication subsystem to replicate user data as well as information necessary for the system operation (*e.g.*, overlay and sensors information). It builds on existing replication techniques, in large part on PRACTI [2]. In order to achieve consistency, Anzere employs Paxos [14]: all nodes in the overlay reach consensus on the total order of updates. The combination of these protocols provides the basis for a broad range of consistency possibilities in Anzere, even though we have not yet worked with update scenarios in the context of this paper.

Anzere currently runs a little over 32,000 lines of Python. The implementation makes heavy use of the Python Twisted framework [25], which is an event-driven networking engine. The software architecture of Anzere is modular, which was initially inspired by the OSGi [17] module management system. The main motivation of this modular architecture is to make the system maintenance easier and also to enable us to customize the functionality and the libraries running on each device based on its hardware architecture and OS platform. The most relevant module of Anzere regarding this paper is the *overlay network*, which includes *network sensors* that run on every node in the ensemble and continuously monitor link and device status. In the context of this paper, the network sensors perform ping latency measurements (such as every 5 seconds) between the devices and keep track of both the instantaneous and the exponentially smoothed RTT values. Anzere also employs models to estimate the expected bandwidth, energy consumption and the throughput of the network links using the measurements performed.

The *storage* module is our primary application driver for this work: it contains *storage sensors* that monitor the status of the data items in the system and partially replicated content according to user-supplied policies. The inter-personal cloud communication architecture we are currently developing makes use of the continuous information flow generated by these two modules in order to compute the optimal routes and object transfer schedules. At the initial simulation stage, the route establishment algorithm was driven using a prototype simulator implemented in Python, with the goal to first establish the potential benefits of our approach. At the system implementation and deployment stage, we employed two disjoint Anzere instances. We present the details of these networks in the following sections.

6 Simulation

Our evaluation consists of two parts: Simulation and system implementation. In this section we present initial simulation results that are aimed at evaluating our approach. Since personal clouds are newly emerging and they may vary quite significantly from one user to another, currently we do not have extensive data about how a typical personal cloud and its connectivity topology looks like. Therefore, at the simulation stage of our research, we have been trying out our ideas using hypothetical model graphs which in practice may resemble personal clouds. In general, during the set of simulations to evaluate this approach, we identified 3 different parameters which we think are important.

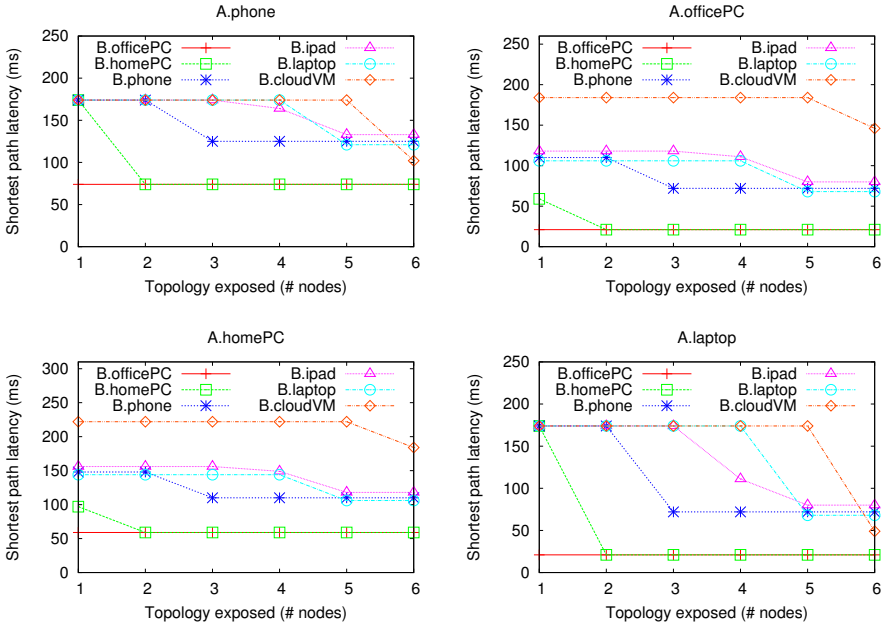


Fig. 4. Shortest path latencies to each node of network B from A.phone, A.officePC, A.homePC and A.laptop

- *Topology exposed*: This parameter denotes how much of the network topology is exposed to the peer network. In these simulations, the network topology is exposed at the granularity of number of nodes, together with *all* the edges that belong to that particular set of nodes.
- *Connecting edges*: This parameter denotes how many connecting edges are present in between the two networks.
- *Weights*: This parameter denotes the weights of the connecting edges.

In the context of our simulations, we have been experimenting with the first item in the list: The network topology that is exposed to the peer overlay.

6.1 Potential Benefits of the Approach

In this section we show initial simulation results aimed at illustrating how much our approach can be beneficial to a user. At the moment, we experiment with metrics such as end-to-end latency (msec) and shortest path to the peer overlay, but we conjecture that other metrics such as bandwidth (bytes/sec), cost (price/byte), power consumption (energy/byte) can also benefit from such an approach.

Pairwise End-to-End Routes. In this simulation we look at the potential benefits our approach can provide in terms of improving pairwise routes between the nodes that belong to two different personal clouds.

For our simulations, we used the two example peer overlays shown in Figure 3. As we mentioned earlier in the paper, we realize that these topologies and the numbers may not be entirely realistic, but they still give us an initial reference point. Our initial experience with a real system follows in the next section of this paper. In this simulation, the network B advertises its network topology to network A and then we look at how the pairwise latencies between the nodes of the two networks are affected depending on the amount of network information exposed. At each stage of the experiment, Network B gradually increases its exposed network topology by one node using the following order of the nodes: officePC, homePC, phone, ipad, laptop, cloudVM. In all the cases (except for the case in which there is only one node advertised from network B), there are 3 connecting edges between the two networks: A.officePC-B.officePC, A.laptop-B.officePC, A.officePC-B.homePC. The weights of all the connecting edges are the same (21) for this simulation.

Figure 4 shows the effect of varying the amount of exposed topology information on the shortest paths to nodes in network B. It shows the shortest path latencies to the nodes of network B from the phone, homePC, laptop and officePC of the network A. The numbers shown here are obtained from a single run of the complete simulation. For the sake of brevity we do not show the latencies originating from the remaining two nodes (ipad and cloudVM) of network A.

These initial figures we present here suggest that maybe not for all, but for some of the node pairs in these networks, changing the amount of exposed network information can significantly improve the pairwise end-to-end route quality.

7 System Experiments

In this section, we present the initial results of our experience with the deployment of our approach in the context of a real system implementation. As mentioned before in Section 5, we implemented our ideas within the Anzere personal storage system [21].

So far, we have investigated two aspects of the approach in our system implementation. Firstly, we investigated the effect of increasing the number of advertised nodes from one Anzere instance to another. Secondly, we looked at the the effect of exposing the internal topology information of an Anzere instance on the shortest paths achievable between the members of two disjoint Anzere instances.

Figure 5 illustrates the network elements and the internal network topologies of Anzere instances we employed in our experiments. The current experimental setup consists of two (initially disjoint) Anzere instances. The first network consists of five devices while the second one contains four devices. Other than the locations noted in parentheses in the figure, the rest of the devices reside in Switzerland.

Anzere's current network and routing layer is designed to establish as many connections as possible between each member of the different instances when an advertisement is received. Therefore, the current topologies in both of the systems can be visualized as fully-connected graphs. The smart-phones run on a wireless connection and are residing

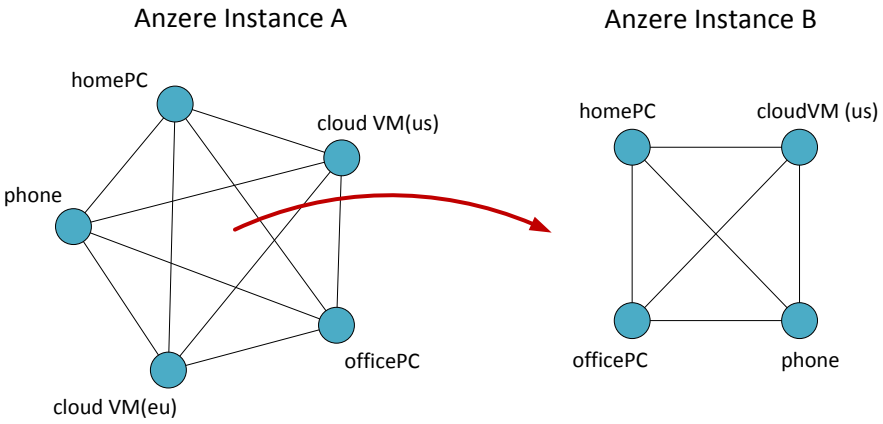


Fig. 5. The network and the topologies of the two Anzere instances used in our system experiments

behind a NATs. The rest of the devices have publicly accessible IP addresses. Typically, the homePCs are also behind NATs but the ones we used in this experiment had public IP addresses. In Anzere, the devices that reside behind the NATs utilize hole punching techniques in order to establish connections to the other members of the Anzere instances.

For the experiments performed in this section, the link measurements were taken every 4 seconds and 0.125 (the same as the value used by TCP) was chosen for the value of α, which is used for the exponential smoothing of the measured data. The advertisement messages were sent from the officePC of the instance A to the officePC of the instance B.

7.1 Effect of Increasing the Number of Advertised Nodes

In this experiment, we try to investigate whether increasing the number of advertised nodes from one Anzere instance to another changes the shortest paths between the all the members of each instances. The Anzere instance A increases the number of nodes it exposes one by one, with every iteration of the experiment. The information exposed about a node is a combination of the hostname/IP address and the port number to be connected to.

The advertisement scheme in this experiment works as follows: The coordinator node (officePC) of the instance A sends its advertisement to the coordinator node of the instance B. The advertisement consists of the exposed nodes' hostnames (or IP addresses) and the port numbers, in addition to the connectivity and topology information (the links and their weights) that exists between the advertised nodes.

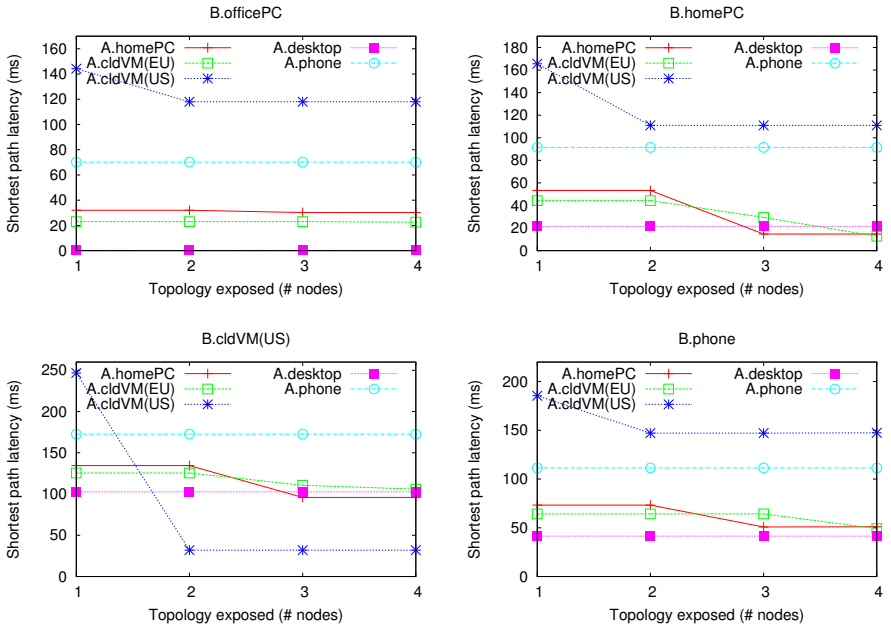


Fig. 6. Latencies to each node in network A originating from B.officePC, B.homePC, B.cloudVM (US) and B.cloudVM (EU)

If a node is not exposed from the network A, the shortest paths to that node is calculated by using the assumption that one of the exposed nodes will act as a gateway to this non-exposed node.

The nodes from the instance A to instance B are advertised incrementally in the following order: OfficePC, cloudVM (US), homePC, cloudVM (EU). In other words, at each iteration of the experiment, one more node is added to the advertisement message. As soon as an an advertisement is received, all the nodes belonging to the instance B try and establish connections to the set of exposed nodes of instance A and then start measuring their link properties, in case the connection establishment attempt has been successful.

Figure 6 shows the change in the shortest paths between the pairs of nodes as the number of exposed nodes from instance A to instance B increases. Similar to the case of simulations, the numbers shown here are obtained from a single run of the complete experiment. As illustrated by this figure, our initial experience with the implementation in a real system supports the figures we have presented in the preliminary simulations. The main message of these plots is that, while it may not affect some of the shortest paths between some pairs of the nodes in disjoint personal clouds, depending on the topology and the network configuration, the routes between some pairs devices can be improved significantly by increasing the number of exposed nodes between the two Anzere instances.

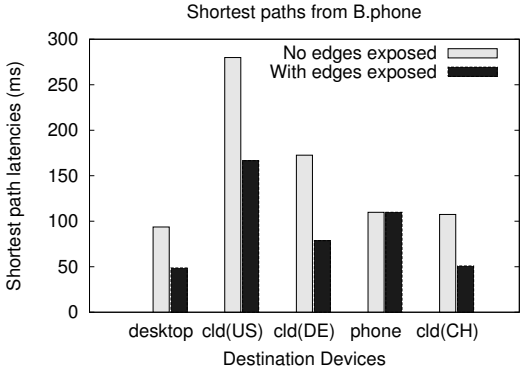


Fig. 7. The effect of exposing internal network topology information on the shortest paths achievable by B.phone to the members of the Anzere instance A

7.2 Effect of Exposing the Topology between the Advertised Nodes

The advertisement messages in the previous experiment included the nodes exposed as well as the internal network topology information in between the advertised nodes of the instance A. In this experiment we try to investigate whether or not exposing the internal network topology information in addition to the advertisement of the nodes makes a significant difference in terms of the shortest paths achievable between the pairs of devices.

Figure 7 shows the shortest paths from B.phone to the members of the instance A. The shortest path values are shown for both the case in which the internal network topology information (the set of edges and their weights) is shared to the peer Anzere instance, and also the case in which this information is not shared. The four nodes that are exposed from the instance A in this experiment are the following: officePC, cloudVM (US), homePC and cloudVM (EU). The set of devices and the topologies of both the personal clouds employed in this experiment are the same as in Figure 5 except for the location of the cloud VM in Europe. A VM in Switzerland was employed instead of a VM in Germany due to the failure of the instance in Germany. Again, the numbers shown here are obtained from a single run of this experiment.

As shown by Figure 7, exposing the internal network topology of an Anzere instance can actually reduce the shortest path latencies to some of the instance A nodes significantly.

8 Conclusion

Establishing efficient routes between personal clouds and the higher-level problem of optimally transferring data objects among personal clouds are new and important problems. In this paper we have presented a technique which provides users with a means to selectively advertise their network information to each other and still arrive at efficient routes. Current results show the benefits for one metric, latency.

In our ongoing work, we are integrating this technique into our personal cloud platform, and incorporating other metrics such as bandwidth, power consumption and monetary cost. Our initial observations show that one can really get more or less benefit from exposing more or less information. Hence, the tradeoff is significant. As an immediate future work, we are planning to extend our approach and apply the idea to the larger problem of optimizing data object transfers between personal clouds. This involves implementation of advertising item distribution data between the two personal clouds in addition to advertising network information.

References

1. Anand, A., Dogar, F., Han, D., Li, B., Lim, H., Machado, M., Wu, W., Akella, A., Andersen, D.G., Byers, J.W., Seshan, S., Steenkiste, P.: XIA: an architecture for an evolvable and trustworthy internet. In: Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets 2011), pp. 2:1–2:6. ACM, New York (2011)
2. Belaramani, N.M., Dahlin, M., Gao, L., Nayate, A., Venkataramani, A., Yalagandula, P., Zheng, J.: PRACTI Replication. In: Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI 2006). USENIX Association (2006)
3. Brewer, E.A., Katz, Y.H., Chawathe, Y., Gribble, S.D., Hodes, T., Nguyen, G., Stemm, M., Henderson, T.: A network architecture for heterogeneous mobile computing. IEEE Personal Communications 5, 8–24 (1998)
4. Clausen, T., Jacquet, P.: Optimized Link State Routing Protocol (OLSR). Internet RFCs, RFC 3626 (2003)
5. Crowcroft, J., Hand, S., Roscoe, T., Mortier, R., Warfield, A.: Plutarch: An argument for network pluralism. In: Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA 2003), pp. 258–266 (2003)
6. Das, S.R., Perkins, C.E., Belding-Royer, E.M.: Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), pp. 3–12 (March 2000)
7. Elizabeth, Y.S., Sun, Y., Belding-Royer, E.M., Perkins, C.E.: Internet connectivity for ad-hoc mobile networks. International Journal of Wireless Information Networks 9(2), 75–88 (2002)
8. Ford, B., Strauss, J., Lesniewski-Laas, C., Rhea, S., Kaashoek, F., Morris, R.: Persistent personal names for globally connected mobile devices. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006), pp. 233–248. USENIX Association (2006)
9. Freedman, M.J., Arye, M., Gopalan, P., Ko, S.Y., Nordstrom, E., Rexford, J., Shue, D.: Service-centric networking with SCAFFOLD. Technical Report TR-885-10, Department of Computer Science. Princeton University (September 2010)
10. Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with Coral. In: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI 2004). USENIX Association (2004)
11. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.: Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2009), pp. 1–12. ACM (2009)
12. Kompella, R.R., Greenberg, A., Rexford, J., Snoeren, A.C., Yates, J.: Cross-layer Visibility as a Service. In: Proceedings of 4th Workshop on Hot Topics in Networks, HotNets IV (2005)

13. Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K.H., Shenker, S., Stoica, I.: A data-oriented (and beyond) network architecture. In: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2007), pp. 181–192. ACM (2007)
14. Lamport, L.: The part-time parliament. ACM TOCS 16(2), 133–169 (1998)
15. Nokia N900 Hardware Power Consumption, http://wiki.maemo.org/N900_Hardware_Power_Consumption
16. Nygren, E., Sitaraman, R.K., Sun, J.: The Akamai network: a platform for high-performance internet applications. SIGOPS Oper. Syst. Rev. 44(3), 2–19 (2010)
17. OSGi Alliance. OSGi Service Platform, Core Specification Release 4, Version 4.1, Draft (2007)
18. Perkins, C.E., Royer, E.M.: Ad-Hoc On-Demand Distance Vector Routing (AODV). In: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999), pp. 90–100 (February 1999)
19. Ramasubramanian, V., Rodeheffer, T.L., Terry, D.B., Walraed-Sullivan, M., Wobber, T., Marshall, C.C., Vahdat, A.: Cimbiosys: a platform for content-based partial replication. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2009), pp. 261–276 (2009)
20. Rekhter, Y., Li, T., Hares, S.: A Border Gateway Protocol 4 (BGP-4). Internet RFCs, RFC 4271 (2006)
21. Riva, O., Yin, Q., Juric, D., Ucan, E., Roscoe, T.: Policy expressivity in the Anzere personal cloud. In: Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC 2011). ACM (2011)
22. Salmon, B., Schlosser, S.W., Cranor, L.F., Ganger, G.R.: Perspective: semantic data management for the home. In: Proceedings of the 7th Conference on File and Storage Technologies (FAST 2009), pp. 167–182 (2009)
23. Strauss, J., Lesniewski-Laas, C., Paluska, J.M., Ford, B., Morris, R., Kaashoek, F.: Device-Transparency: a New Model for Mobile Storage. In: Proceedings of the Workshop on Hot Topics in Storage and File Systems (HotStorage 2009) (October 2009)
24. Stuedi, P., Bihr, M., Remund, A., Alonso, G.: SIPHoc: Efficient SIP Middleware for Ad Hoc Networks. In: Cerqueira, R., Campbell, R.H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 60–79. Springer, Heidelberg (2007)
25. Python Twisted, http://twistedmatrix.com/trac/
26. Ucan, E., Roscoe, T.: Dexferizer: a service for data transfer optimization. In: Proceedings of the 19th International Workshop on Quality of Service (IWQoS 2011), pp. 33:1–33:9. IEEE Press (2011)
27. Wang, L., Park, K.S., Pang, R., Pai, V., Peterson, L.: Reliability and security in the CoDeeN content distribution network. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC 2004). USENIX Association (2004)
28. Wroclawski, J.T.: The Metanet. White Paper. In: Proceedings of Workshop on Research Directions for the Next Generation Internet (1997)

Developing, Deploying and Evaluating Protocols with ManetLab

François Vessaz[1], Benoît Garbinato[1], Arielle Moro[1], and Adrian Holzer[2]

[1] Université de Lausanne, Lausanne, Switzerland
{francois.vessaz,benoit.garbinato,arielle.moro}@unil.ch
[2] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
adrian.holzer@epfl.ch

Abstract. Evaluating the performance of MANET-specific communication protocols is essential to build robust mobile ad hoc applications. Unfortunately, most existing evaluation results are either based on simulations – which makes it difficult to draw conclusions beyond confined lab settings – or they are based on custom testbed results – which makes it difficult to reproduce them. In order to overcome this challenge, we introduce ManetLab, a modular and configurable software framework for creating and running testbeds to evaluate MANET-specific protocols. With ManetLab, one can easily configure and automate reproducible protocol executions on standard computer hardware, and thus provides both the *accuracy* of testbed-based evaluations and the *reproducibility* of simulation-based evaluations. After presenting ManetLab's extensible architecture, based on the notion of modular protocol stack, we show how it helps evaluate the performance of different broadcast protocols in real MANETs and how its results compare with simulation-based results.

1 Introduction

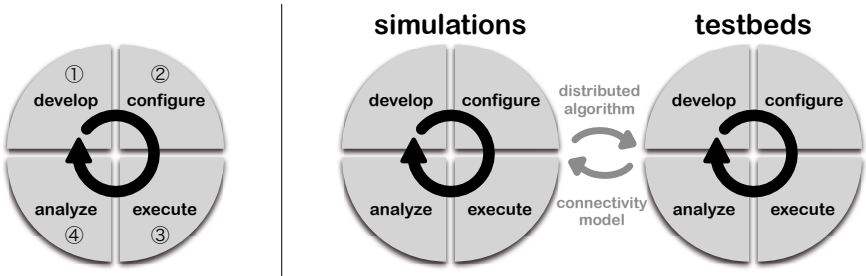
With the tidal wave created by the arrival of smart devices and tablets, the prospects of seeing MANET-based applications appear in the distributed systems landscape has become more promising than ever. To encourage the emergence of such applications, system developers must provide solid communication building blocks for application developers, such as multi-hop broadcast, multicast, unicast, and other dissemination and routing protocols. Along that line, a large amount of research effort have been spent investigating mobile ad hoc routing protocols over the past decade. Central to this effort are the specialized tools that allow researcher to *develop* and *evaluate* their protocols.

1.1 Protocol Development and Evaluation

The development of an effective and efficient protocol, be it wired, wireless infrastructure-based or ad hoc, is an iterative process consisting of four steps, as illustrated in Figure 1a. For a start, one has to devise the protocol in the form of a distributed algorithm, ideally proving it formally and ultimately implementing

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 89–104, 2013.

it in some programming language (Step 1). Then, one has to configure some test environment in which the protocol will be executed (Step 2) and run the actual tests (Step 3). Finally, one has to analyze the collected data (Step 4), which might then lead to fine-tune the protocol and trigger a new iteration.



(a) general iterative process

(b) combining simulations and testbeds

Fig. 1. Development of communication protocols

Existing tools for evaluating protocol performance can be categorised as either *simulators* or *testeds*. When using testbeds, the iterative process sketched in Figure 1a can be very time consuming, especially if one wants to evaluate performances in various distributed environments, which is clearly a must. This is particularly true for Steps 2 and 3, since they imply the deployment of the protocol code and of the test configuration to various distributed nodes, the launching of the testbed execution and the gathering of the results obtained at each individual node. For this reason, dedicated tools aimed at facilitating the creation and execution of testbeds have been proposed by the research community for specific families of distributed environments. This is for example the case of PlanetLab[1] for large-scale distributed systems [9].

Performance Evaluation Tools for MANETs. When it comes to evaluate the performance of MANET-specific protocols however, until now researchers have had essentially the choice between the reproducibility offered by simulators and the accuracy offered by testbeds. On the positive side, simulators are widely used by the research community and their source code is generally accessible online, which makes simulation-based evaluations fairly reproducible. Unfortunately, as they rely on the modeling of complex physical and logical parameters, it is difficult to draw general conclusions about the behavior of such protocols in real settings [26]. Section 5 further discusses evaluation tools for MANETs.

Testbeds on the contrary rely on real mobile ad hoc networking and therefore tend to offer a high-level of accuracy. For this very reason however, they also tend to impose a high development and deployment barrier [1]. In addition, most testbeds are not directly available to other researchers and often require

[1] `http://www.planet-lab.org`

specialized or incompletely specified hardware. For these reasons, the level of reproducibility of the resulting performance evaluation is generally quite low.

1.2 Contribution and Roadmap

The ManetLab framework precisely aims at filling this gap, by supporting both accurate and reproducible performance evaluations of MANET-specific protocols, in a similar way PlanetLab does it for large-scale distributed systems.

In Section 2, we discuss the need for a tool such as ManetLab and its key requirements to achieve high accuracy and reproducibility. In Section 3, we present ManetLab in detail, by showing how it helps develop protocol layers and assemble them into a full protocol stack, which can then be deployed on remote nodes. We also introduce ManetLab's graphical tool, which helps configure performance evaluations, launch them and gather the corresponding results. In Section 4, we then compare the results of various performance evaluations obtained using ManetLab with those obtained with two simulations tools. While simulation-based performance evaluations are fairly accurate for simple MANET environments, they diverge significantly from the results obtained in reality for more complex environments, typically involving physical obstacles between nodes. Interestingly, when injecting the topological constraints observed with ManetLab back into the two considered simulators, simulation-based evaluation tend to augment their level of accuracy. Finally, we discuss existing testbeds for MANETs in Section 5, and ongoing work on improving ManetLab in Section 6.

2 Achieving Accuracy and Reproducibility

To be useful, performance evaluations of MANET-specific protocols should obviously achieve a high degree of both accuracy and reproducibility. As a consequence, tools supporting such evaluations should mimic real-life environments as accurately as possible and should allow researchers to easily reproduce experiments with the purpose of comparing evaluation results.

2.1 Combining Simulations and Testbeds

Although this paper focuses on the need for a robust testbed tool, we advocate the combination of simulations and testbeds, as illustrated in Figure 1b. While simulations can indeed be considered an acceptable first approximation, communication protocols tend to perform in a more unpredictable way when actually deployed in a real MANET than in an infrastructure-based network, be it wired or wireless. For this reason, we believe that ultimately accuracy can only be achieved with testbed approaches, at least at that stage of the evaluation.

In the early phases of a protocol development, simulations can be very useful to validate the protocol in simple environment settings, e.g., in the absence of walls or obstacles, using a basic wireless propagation model. Once this first validation is done, the protocol should then be evaluated in a real mobile ad hoc

network, using testbeds. As shown in Figure 1b, the results of testbed-oriented evaluations can then be injected back into simulations, typically in the form of a more accurate model of the wireless connectivity among network nodes. This is precisely the approach we follow in Section 4.

2.2 Creating Accurate and Reproducible Testbeds

While various simulation tools exists, some of which have become de facto standards, the situation is very different when it comes to testbeds for mobile ad hoc networks. Moreover, coming up with universal and rigid testbeds for MANETs might not even be a desirable goal, given the great variability of actual deployment settings. Rather, we believe that there exists a need for a framework that facilitates the development, deployment and evaluation of MANET-specific protocols, by making it easy to create accurate and reproducible testbeds. That is, *accuracy* and *reproducibility* should be the two key requirements for such a testbed framework.

Accuracy. Devising a tool that offers an accurate evaluation of the behavior of a protocol running in a MANET can be very challenging. The central issue stems from the fact that MANETs tend to exhibit very erratic behaviors in terms of connectivity and of reliability, depending on their physical environment and on how nodes are moving. In addition, the various layers that stack up, in particular TCP/IP, tend to distort the actual performance evaluation of communication protocols in MANETs. For this reason, an adequate testbed framework should offer flexibility in protocol layering, all the way down to the lowest-level layers, typically by making it easy to compose protocol stacks from elemental layers.

Reproducibility. To evaluate the accuracy of results provided by an evaluation tool, other researchers must be able to reproduce the testbeds described in the literature, and scrutinize the evaluation tool itself. Thus, it is important that any evaluation tool, in particular a testbed framework, be easily accessible for the research community. A testbed framework should in addition be configurable, in order to easily switch from one deployment setting to another, and it should offer support for automatically launching evaluations and gathering results.

3 Introducing ManetLab

ManetLab is a framework supporting the creation and execution of accurate and reproducible testbeds for MANETs-specific protocols, using mainstream hardware.[2] On each computer where ManetLab is installed, the wireless network interface is used to connect the MANET, while the wired network interface is used as control network to provide feedback about the protocol performance.

[2] ManetLab runs on Apple's computers with Mac OS X 10.7 or higher. We justify our choice of the Apple's platform by the fact that the hardware and the OS are highly standardized and by the fact that Apple has the highest market share for portable selling to U.S. higher education since 2007.

More specifically, as illustrated in Figure 2, each computer running ManetLab is hosting an *agent* connected to the MANET. In addition, one of the computers hosts the *controller*, which acts as a conductor orchestrating the protocol execution.[3] That is, the controller uploads the protocol stack to each agent, triggers the execution of the protocol and collects feedback from all agents about the protocol execution, via the wired network interface. In the following, we discuss how each step pictured in Figure 1 is performed with ManetLab.

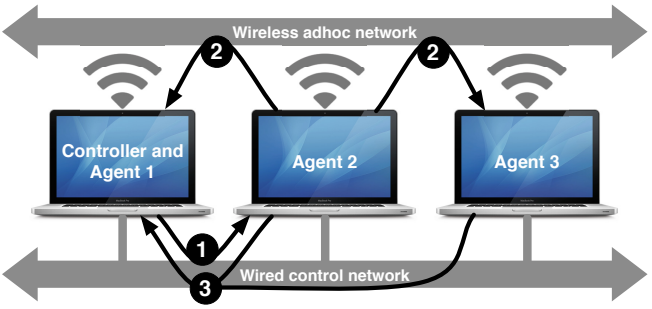


Fig. 2. ManetLab — Using an ad hoc network and a control network

3.1 Development

In order to test an ad hoc protocol, one has to first implement it. ManetLab proposes an API[4] to help MANET-specific protocol developers in this task. The protocol must be implemented in Objective-C and designed as layers, inheriting from the MLStackLayer class, in a stack (MLStack) provided by the API. The layer above the stack represents the application, whereas the layer below the stack is the antenna. At any given time, a ManetLab node is executing at most one protocol stack. Communication between nodes and between layers inside a stack is achieved via message passing (MLMessage). Figure 3 illustrates a stack containing two layers, i.e., a fragmentation layer and a gossip layer.

As pointed out in [26], researchers rarely make the effort to provide the source code of their MANET-specific protocols to the community, which makes it very difficult to seriously compare different protocols pursuing the same goal. Moreover, even when the source code is provided, the absence of a standardized tool to compose and deploy protocols leads to low reproducibility of most research results. For this reason, ManetLab proposes a plugin architecture that makes it easy to package all the layers (subclasses of MLStackLayer) and message types (subclasses of MLMessage) composing a MANET-specific protocol stack. As a result, stacks created using ManetLab can be shared and reused by other researchers. In addition, ManetLab is an open source project, which further promotes the scrutiny and reproducibility of testbeds relying on it.

[3] The example depicted in Figure 2 is further discussed in Section 3.3.

[4] The API is distributed with the ManetLab software and available at `http://doplab.unil.ch/manetlab`

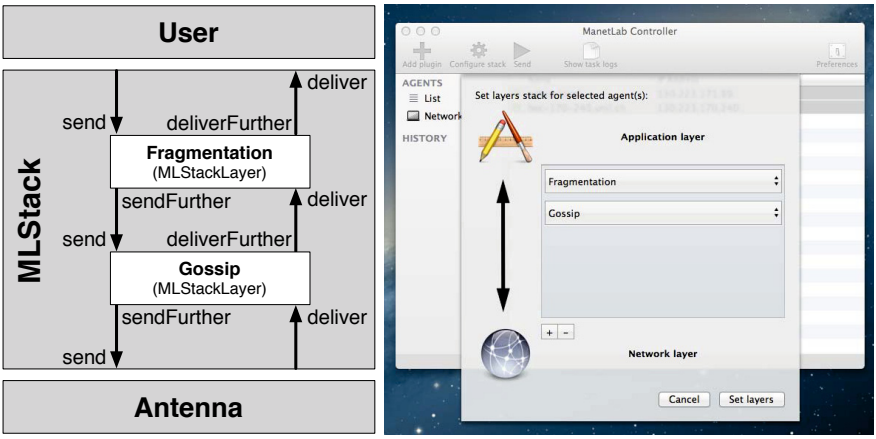


Fig. 3. A stack with two layers (left) and its corresponding GUI (right)

3.2 Configuration

The configuration step greatly differs in a simulator-based approach and in a testbed-based approach. However, in both cases, configuration entails *installation* of the tool (simulator or testbed) and its *parameterization*, and then *deployment* of the protocol code in the tool.

Installation. Installing a simulator is usually non-trivial because it implies to download the source code from the Internet and then to build the simulator from that code. As for testbeds, they are rarely made publicly available and when they are, they tend to be even more difficult to install and to use, since they often require specialized hardware. In contrast, ManetLab requires just a few clicks to be installed on a standard desktop or laptop computer.[5] Yet, if one wants to access its source code, it is also made available via GitHub.[6]

Parameterization. For both simulators and testbeds, parameterization implies to load the protocol stack into the tool, usually in some binary form. Apart from this obvious step, parameterization is where simulators and testbeds differ the most. As testbeds rely on real MANET implementations, one has only a few parameters to set, e.g., the wireless channel used to communicate; this is typically the case with ManetLab. In addition, as already suggested in Section 3.1, ManetLab makes it easy to dynamically load plugins containing protocol layers into the tool, to then graphically compose a protocol stack from these layers and deploy it of each node of the MANET (Figure 3). When it comes to simulators however, many more parameters have to be set, such as the mobility model, the connectivity model, the node distribution model, the interference model, etc.

[5] ManetLab executable is available from `http://doplab.unil.ch/manetlab`.
[6] ManetLab source code is available from `http://github.com/doplab/ManetLab`.

In terms of *accuracy*, this step is critical because unrealistic values may result in misleading or even erroneous performance evaluations.

Deployment. When using testbeds, one of the major obstacles to reproducibility often lies in the need to deploy and maintain specialized hardware, typically in the form of prototype devices. In order to solve this problem, at least partially, ManetLab is implemented on the OS X platform, which is widespread in research institutions today. In addition, Apple's hardware is known to be very standardized and traceable, e.g., using a tool like Mactraker,[7] which is clearly an advantage in term of *reproducibility*. Moreover, building ManetLab on top of OS X, which shares the same code basis as iOS when it comes to low-level services, opens the opportunity to port ManetLab to iOS devices in the future.

3.3 Execution

With ManetLab, protocols communicate using the IEEE 802.11 wireless ad hoc network (IBSS mode), so the accuracy of performance evaluations is naturally ensured. On the controller, the graphical user interface shown in Figure 4 is used to prepare and launch the testbed execution.

Execution Example. Arrows pictured in Figure 2 illustrate an execution with ManetLab, where the controller simply requests one agent to broadcast a message. First, the controller asks Agent 2 to broadcast a message (Arrow 1). As a result, Agent 2 does indeed broadcast a message on the wireless ad hoc network (Arrow 2). Finally, all agents have received the message and provide feedback to the controller, using the wired and reliable control network (Arrow 3).

Offline Control Mode. Since the controller communicates with agents via a wired control network, ManetLab does not allow to test protocol with mobility in its first version. To overcome this limitation, we are currently implementing an *offline control mode*, which uses the wireless network for both control messages and protocol messages. The idea is to have each agent log its evaluation results locally during the testbed execution, so that it can send them to the controller after the execution.

3.4 Analysis

Most simulators and testbeds do not provide specific analysis tools. Rather, they allow protocol developers to produce log files or populate databases, which can then be fed into some analysis tool, such as a graphical network animator like NetAnim for example. This is also the case of the ManetLab testbed framework: its API offers a `log` methods that allows developers to produce whatever trace they need for their performance evaluation.

[7] `http://mactracker.ca`

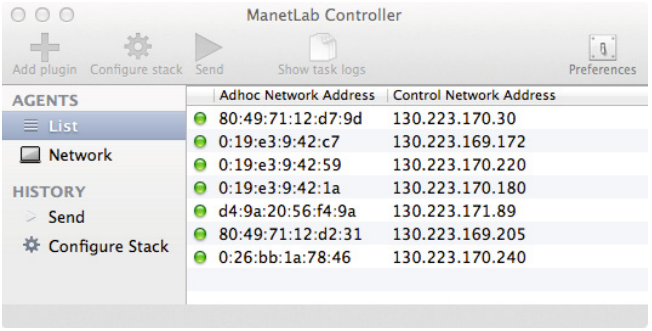


Fig. 4. ManetLab Controller — Graphical user interface

4 ManetLab Testbeds *vs.* Simulations

To compare performance evaluations obtained from ManetLab with those obtained from simulations, we study the behavior of various broadcast protocols in ManetLab and in two simulators. These two simulators are ***NS-3*** [15], the latest simulator from the *NS* family, and ***Sinalgo***,[8] a simple Java-based simulator we used to evaluate several of our own MANET-specific protocols [11,12,13]. For our comparison, we rely on two simple, but widely used in the research community, measures: the ***delivery ratio***, defined as the *number of nodes who received a message over the total number of nodes*, and the ***forward ratio***, defined as the *number of nodes who send or retransmit the message over the total number of nodes*. Each measure is the average of 1'000 distinct executions.

4.1 Network Settings

In order to avoid a potential distortion or overhead caused by TCP/IP (in particular its routing scheme), all protocols are directly using the MAC layer when it comes to broadcast a message in the MANET. Along that line, we parameterize the MAC layer of each tool in a similar way, as discussed hereafter.

ManetLab. Since ManetLab is a real MANET implementation, there are only a very few settings we can change. All other settings are constraints deriving from the operating system and the hardware on which ManetLab is running. Basically, ManetLab creates an IEEE 802.11a ad hoc network with a theoretical data rate of maximum 6 MBits/s for broadcast.

NS-3. We use the *WifiNetDevice* from NS-3 with the *YansWifiChannel* and the *YansWifiPhy* models. We set all the settings we can to similar values of what ManetLab uses on real computers. That is, we configure NS-3 to use an IEEE 802.11a physical layer model, with a data rate of maximum 5.5 Mbits/s and a MTU of 1'500 bytes.

[8] `http://www.disco.ethz.ch/projects/sinalgo/`

Sinalgo. Being a higher-level simulator than NS-3, Sinalgo does not rely on an implementation of the IEEE 802.11 standard. So we configure Sinalgo to send messages smaller than 1'500 bytes, with a *Unit Disk Graph* connectivity model and a *Signal to Interference plus Noise Ratio* interference model.

4.2 Protocols, Environments and Communication Patterns

Because we aim at providing a solid first comparison, we consider a number of *broadcast protocols*, *physical environments* and *communication patterns*. They are presented in details hereafter.

Broadcast Protocols. We consider three probabilistic broadcast protocols, namely ***Simple Flooding***, ***Gossip***, and ***Counter-Based Scheme*** (CBS), which are well-known to the research community. With Simple Flooding [14], each node systematically retransmits a message the first time it receives it, so the delivery ratio is always equal to the forward ratio. With Gossip [29], each node retransmits a message with a probability p the first time it receives it. For our comparisons, we set p to 0.7, 0.5, and 0.2. Finally, with CBS [29], a node waits for some random delay between 0 and w_{max} before retransmitting a message, only if it received it only once. That is, if a node receives a message more than once, it does not retransmit it. For CBS, we set w_{max} to 0.1 and 0.5 seconds.

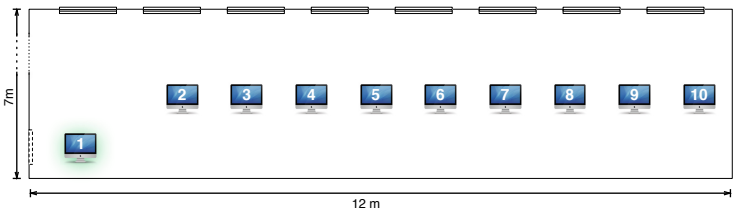


Fig. 5. Sketch of the open space environment

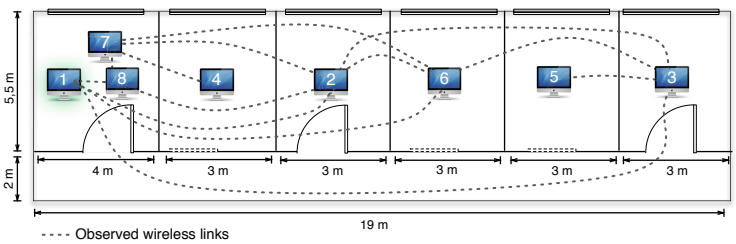


Fig. 6. Sketch of the private offices environment

Physical Environments. We consider two environments: an ***open space*** and ***private offices***. In the open space, ten computers are placed in one open space as illustrated in Figure 5; this is typically the case in a classroom. With private

offices, eight computers are placed in adjacent offices as depicted in Figure 6. In each physical environment, Computer 1 acts as the initial broadcaster.

Communication Patterns. We consider two communication patterns: a ***one-shot*** message, which corresponds to a low network load, and the ***streaming*** of 1'000 messages, which correspond to a high network load. With the *one-shot* pattern, Computer 1 broadcasts a single 1'400-bytes message. Those 1'400 bytes are encapsulated in just one network frame consisting of 1'485 bytes, including headers. With the *streaming* pattern, Computer 1 sends 1.4 Mbytes, which are fragmented into 1'000 network frames of 1'485 bytes each.

4.3 Results in the Open Space Environment

Figure 7 shows the delivery and forward ratios of the one-shot communication pattern. Since all nodes are connected (fully connected graph) and there are almost no interference, the delivery ratios are strictly equal to 1.0 for all protocols in NS-3 and Sinalgo, and above 0.99 for ManetLab. For this reason, the forward ratios tend to converge towards their theoretical values, i.e., 1 for flooding, p for gossip and much smaller values for CBS. Overall, we can say that in this scenario (one-shot in an open space), simulations are quite accurate since they faithfully mimic the results obtained by ManetLab in a real MANET.

In the second scenario (streaming in an open space), interference starts to disturb the behavior of the protocols and affect both the delivery ratio and the forward ratio, as shown in Figure 8. The more messages are transmitted, e.g., for flooding or for gossip with $p = 0.7$, the more the delivery ratio decreases. Interestingly, the delivery ratios of ManetLab are over 0.7, whereas the delivery ratios of NS-3 and Sinalgo are under 0.6. That is, the interference models used in the two simulators are discarding too many frames, which indicates that their accuracy is diminishing. As for the forward ratios, they tend to be only slightly lower with the simulators than with ManetLab.

4.4 Results in the Private Offices Environment

In the private offices environment, the real MANET experienced by ManetLab is no longer a fully connected graph, due to various physical obstacles (mainly walls but also furniture, possibly people, etc.). It is thus not surprising that the delivery ratio of a one-shot communication pattern in ManetLab tends to drop compared to the open space, as shown in Figure 9. NS-3 and Sinalgo, on the contrary, continue to view the MANET as a fully connected graph, so their delivery ratios remain strictly equal to 1. This clearly indicates that their level of accuracy is dropping. As for the forward ratios, NS-3 and Sinalgo have similar results to those of ManetLab except for CBS.

Streaming in private offices is by far the worst scenario when it comes to the delivery ratio, as shown in Figure 10. Both physical obstacles and interference are significantly decreasing the performances of all protocols. Again, the accuracy of NS-3 and Sinalgo is compromised, as they only roughly approximate the delivery ratio observed with ManetLab. Furthermore, since fewer nodes receive

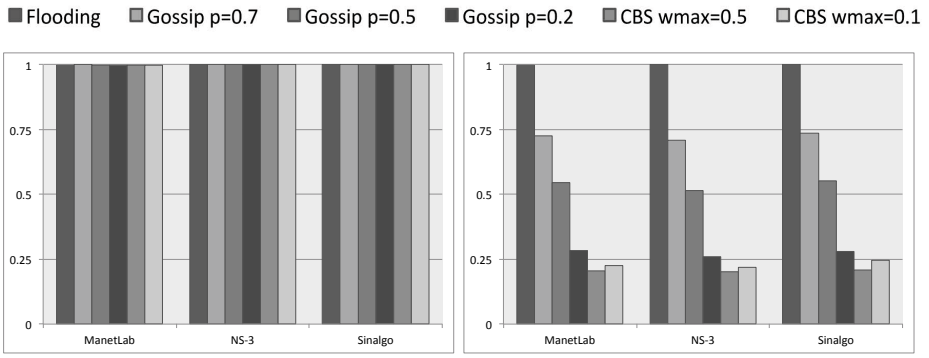


Fig. 7. One-shot in an open space – delivery ratio (left) and forward ratio (right)

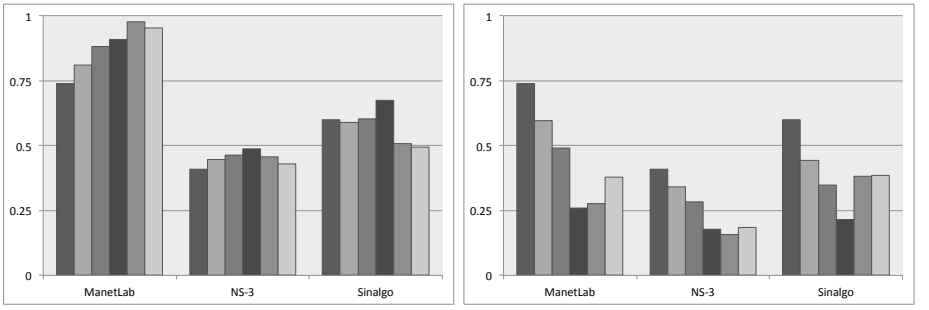


Fig. 8. Streaming in an open space – delivery ratio (left) and forward ratio (right)

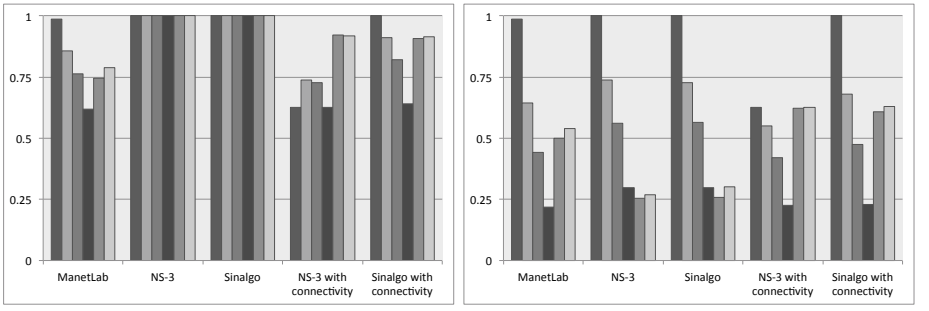


Fig. 9. One-shot in private offices – delivery ratio (left) and forward ratio (right)

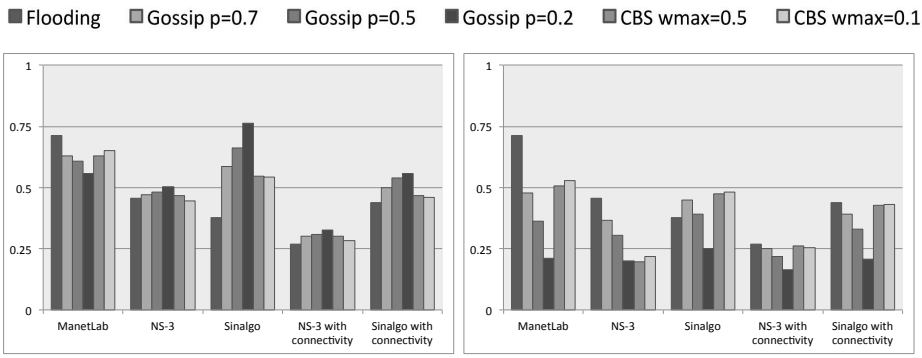


Fig. 10. Streaming in private offices – delivery ratio (left) and forward ratio(right)

the messages being broadcast, the forward ratios with NS-3 and Sinalgo are also dropping and thus diverge from those observed with ManetLab.

4.5 Injecting the Observed Connectivity into Simulations

It seems reasonable to assume that the drop in accuracy we observe for both NS-3 and Sinalgo, when considering private offices, is largely due to their erroneous modeling of the MANET connectivity. In order to confirm this assumption, we inject the connectivity graph experienced by ManetLab (see Figure 6) into NS-3 and Sinalgo, and we re-run our performance evaluations.

As shown in Figure 9, after the injection the accuracy of both NS-3 and Sinalgo is improved for the one-shot communication pattern. Interestingly, and somewhat surprisingly, the results for the simple flooding protocol are more accurate with Sinalgo than with NS-3. With the streaming communication patterns however, injecting the connectivity graph is not sufficient to improve the accuracy of NS-3 and Sinalgo, as shown in Figure 10. It seems that the effect of interference, combined with a lower connectivity, leads both simulators to produce results that are significantly lower than what happens in reality.

5 Related Work

To evaluate the behavior of their protocols, researchers should rely on simulators and testbeds that aim at providing *accurate* and *reproducible* performance evaluations. Hereafter, using these two dimensions, we review a wide range of evaluation tools for MANETs found in the literature [26,28,24,16,25] and we compare them with ManetLab. For accuracy, we focus on their communication support, as this is a critical element when it comes to evaluate performance in a MANET. For reproducibility, we assess the availability of the tools.

5.1 Communication Support

Simulators on the one hand do not provide a real implementation of a wireless communication layer. For this reason, assessing the accuracy of their communication support, i.e., of their modelling of wireless communications, is very difficult and can only be achieved by comparing their results with those of a real MANET (as we did in Section 4). Such tools include NS-2 [8], NS-3 [15], GloMoSim [33] and its commercial version Qual Net[9], OPNET [7], OMNet++ [32], and others such as Sinalgo or JIST / SWANS [3,4].

Testbeds on the other hand tend to be more accurate because they rely on real wireless communication links. Such tools include Castadiva [17], MASSIVE [27], MobiEmu [34], mLab [21], Carnegie Mellon University Wireless Emulator [22], ORBIT [30], Seawind [23] and WHYNET [35] or JEmu [18], PoEM [19], and of course ManetLab. Some testbeds however tend to oversimplify topological constraints, e.g., by simply piling up a stack of wireless devices. In addition, while one-hop communication is provided by all testbeds, multi-hop communication is only found in tools such as WHYNET, RoofNet [6], ManetLab and Airplug-emu [5]. Other testbeds also provide multi-hop communication, but they shorten their wifi range. Such tools include ORBIT, TrueMobile [20] and MiNT [10]. Other tools provide a logical multi hop communication, such as mLab, MobiEmu and Castadiva.

5.2 Tool Availability

To assess the availability of each tool, we evaluate if it is available *online*, if its *source code* is disclosed and available for download and installation, if detailed *documentation* is provided, and if *specialized hardware* is required in order to run testbeds relying on that tool.

Online Availability. While many of the surveyed tools are available online for download, just as ManetLab, some other tools are only described in scientific papers, with no further details provided online. This is for instance the case of TrueMobile, PoEM, MASSIVE, JEmu, and the tool described by Barolli et al. [2]. This makes it very hard for other researchers to get a hold of these tools and reproduce experiments.

Source Code Availability and Documentation. In order to evaluate the accuracy of a performance evaluation tool, providing the disclosed source code is another important aspect. Most reviewed simulators, except GloMoSim and OPNet, provide a downloadable version of their code. Among testbeds however, source code becomes much more scarce: only Castadiva, MobiEmu, mLab, Airplug-emu and MIT Roofnet provide access to their code, some of them without much documentation. ManetLab on the other hand provides both its source code and an easy-to-install binary file, with documentation and examples online.

[9] `http://www.scalable-networks.com/content/products/qualnet`

Specialized Hardware. While most simulators can be easily deployed on almost any computer, many testbeds require specialized hardware. This requirement makes it harder for other researchers to install the testbed and execute existing protocols. Moreover, some testbeds are deployed in specific lab settings and allow remote users to connect, such as CMUTrueMobile (based on the Emulab testbed [31]), which offers access to its testbed built on custom robots, or ORBIT which offers a testbed of 400 fixed WiFi devices placed in a grid formation on the ceiling of a single room. Other tools are devised to use special hardware, such as MiNT-m that uses Roomba vacuum cleaner robots as underlying hardware in order to support mobility and custom hardware on which to run protocols. Airplug-emu is another such example and is designed to emulate vehicular network and runs on laptops connected to specific GPS and radio receivers. Remote solutions have the advantage of side-stepping the tool deployment stage, and often allowing node mobility, but they impose restrictions on the execution scenarios. Other tools along with ManetLab can be deployed on standard equipment, which makes it easier for others to deploy and evaluate them. These tools include: Castadiva, MobileEmu, mLab and Airplug-emu.

6 Conclusion

Even though performance evaluation is central when it comes to designing robust MANET-specific communication protocols, we believe this problem has not been addressed in a satisfactory manner so far. Either protocols were evaluated through simulations and the results might not be valid in a real MANET environment, or they were evaluated in a customized testbed, which makes it hard to reproduce experiments. In this paper we presented ManetLab as a solution to this conundrum: ManetLab aims at offering the best of both worlds, i.e., accurate and reproducible results. In future work, we plan to extend ManetLab to iOS devices and to add an offline control mode.

Acknowledgement. This research is partially funded by the Swiss National Science Foundation under project numbers 138092 and 140762.

References

1. Al-Bado, M., Sengul, C., Merz, R.: What details are needed for wireless simulations? - a study of a site-specific indoor wireless model. In: INFOCOM 2012, pp. 289–297 (2012)
2. Barolli, L., Ikeda, M., Xhafa, F., Durresi, A.: A testbed for manets: Implementation, experiences and learned lessons. IEEE Sys. Journ. 4(2), 243–252 (2010)
3. Barr, R., Haas, Z., van Renesse, R.: Jist: An efficient approach to simulation using virtual machines. Software Practice & Experience 35(6), 539–576 (2005)
4. Barr, R., Haas, Z., van Renesse, R.: Scalable wireless ad hoc network simulation. In: Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks, ch. 19 (2005)

5. Buisset, A., Ducourthial, B., El Ali, F., Khalfallah, S.: Vehicular networks emulation. In: ICCCN 2010, pp. 1–7 (2010)
6. Chambers, B.A.: The grid roofnet: A rooftop adhoc wireless network. M.S. Thesis, MIT, Cambridge, Massachusetts (June 2002)
7. Chang, X.: Network simulations with opnet. In: Wintersim 1999, pp. 307–314 (1999)
8. Chen, Q., Schmidt-Eisenlohr, F., Jiang, D., Torrent-Moreno, M., Delgrossi, L., Hartenstein, H.: Overhaul of ieee 802.11 modeling and simulation in ns-2. In: MSWiM 2007, pp. 159–168 (2007)
9. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: An Overlay Testbed for Broad-Coverage Services. ACM CCR 33(3) (2003)
10. De, P., Raniwala, A., Sharma, S., Chiueh, T.: Mint: A miniaturized network testbed for mobile wireless research. In: INFOCOM 2005, pp. 2731–2742 (2005)
11. Garbinato, B., Holzer, A., Vessaz, F.: Six-Shot Broadcast: A Context-Aware Algorithm for Efficient Message Diffusion in MANETs. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 625–638. Springer, Heidelberg (2008)
12. Garbinato, B., Holzer, A., Vessaz, F.: Context-aware broadcasting approaches in mobile ad hoc networks. In: Computer Networks, pp. 1210–1228 (2010)
13. Garbinato, B., Holzer, A., Vessaz, F.: Six-Shot Multicast: A Location-Aware Strategy for Efficient Message Routing in MANETs. In: NCA 2010, pp. 1–9 (2010)
14. Heinzelman, W.R., Kulik, J., Balakrishnan, H.: Adaptive protocols for information dissemination in wireless sensor networks. In: MOBICOM 1999, pp. 174–185 (1999)
15. Henderson, T.R., Lacage, M., Riley, G.F.: Network simulations with the ns-3 simulator. In: SIGCOMM 2008 (2008)
16. Hortelano, J., Cano, J.-C., Calafate, C.T., Manzoni, P.: Testing applications in manet environments through emulation. EURASIP J. Wirel. Commun. Netw., 47:1–47:9 (2009)
17. Hortelano, J., Nacher, M., Cano, J.-C., Calafate, C.T., Manzoni, P.: Castadiva: A Test-Bed Architecture for Mobile AD HOC Networks. In: PIMRC 2007, pp. 1–5 (2007)
18. Flynn, H.T.J., O'Mahony, D.: Jemu: A real time emulation system for mobile ad hoc networks. In: Symp. on Tel. Sys. Res. (2001)
19. Jiang, W., Zhang, C.: A portable real-time emulator for testing multi-radio manets. In: IPDPS 2006, pp. 169–169 (2006)
20. Johnson, D., Stack, T., Fish, R., Flickinger, D., Ricci, R., Lepreau, J.: Truemobile: A mobile robotic wireless and sensor network testbed?, flux technical note ftn-2005-02. In: INFOCOM 2006 (2006)
21. Karygiannis, A., Antonakakis, E.: mlab: A mobile ad hoc network test bed. In: SecPerU 2005 (2005)
22. Borries, K., Xiaohui, W., Judd, G., Steenkiste, P., Stancil, D.: Experience with a wireless network testbed based on signal propagation emulation. In: EW 2010 (2010)
23. Kojo, M., Gurtov, A., Manner, J., Sarolahti, P., Alanko, T., Raatikainen, K.: Seawind: a wireless network emulator. In: MMB 2001 (2001)
24. Kropff, M., Krop, T., Hollick, M., Mogre, P., Steinmetz, R.: A survey on real world and emulation testbeds for mobile ad hoc networks. In: RIDENTCOM 2006 (2006)
25. Kulla, E., Ikeda, M., Barolli, L., Xhafa, F., Iwashige, J.: A survey on manet testbeds and mobility models. In (Jong Hyuk) Park, J.J., Chao, H.-C., Obaidat, M.S., Kim, J. (eds.) CSA 2012. LNEE, vol. 114, pp. 651–657. Springer, Heidelberg (2012)

26. Kurkowski, S., Camp, T., Colagrosso, M.: MANET simulation studies: the incredibles. Mob. Comput. Commun. Rev. 9(4), 50–61 (2005)
27. Matthes, M., Biehl, H., Lauer, M., Drobnik, O.: Massive: An emulation environment for mobile ad-hoc networks. In: WONS 2005, pp. 54–59 (2005)
28. Qabajeh, M.M., Hashim, A.A., Khalifa, O.O., Qabajeh, L.K., Daoud, J.I.: Performance evaluation in manets environment. Australian J. of Basic and Appl. Sciences 6(1), 143–148 (2012)
29. Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., Sheu, J.-P.: The broadcast storm problem in a mobile ad hoc network. In: MobiCom 1999, pp. 151–162 (1999)
30. Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., Singh, M.: Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In: WCNC 2005, pp. 1664–1669 (2005)
31. Stack, T., Fish, R., Flickinger, D.M., Stoller, L., Ricci, R., Lepreau, J.: Mobile emulab: A robotic wireless and sensor network testbed. In: INFOCOM 2006, pp. 1–12 (2006)
32. Varga, A.: The omnet++ discrete event simulation system. In: ESM 2001 (2001)
33. Zeng, X., Bagrodia, R., Gerla, M.: Glomosim: A library for parallel simulation of large-scale wireless networks. In: PADS 1998, pp. 154–161 (1998)
34. Zhang, Y., Li, W.: An integrated environment for testing mobile ad-hoc networks. In: MobiHoc 2002 (2002)
35. Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., Bagrodia, R.: Whynet: a hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. In: WiNTECH 2006, pp. 111–112 (2006)

Airtime Ping-Pong Effect in IEEE 802.11s Wireless Mesh Networks

Mohamed Riduan Abid[1] and Saâd Biaz[2]

[1] School of Science and Engineering
Alakhawayn University in Ifrane, 53000, Ifrane, Morocco
R.Abid@aui.ma
[2] Computer Science and Software Engineering Department
Shelby Center for Engineering Technology, Suite 3101
Auburn University, AL, 36849-5347, USA
biazsaa@auburn.edu

Abstract. Airtime is set as the default routing metric for the ongoing IEEE 802.11s wireless mesh networking standard. The metric is designed to minimize channel resource consumption by accounting for loss rate, bandwidth, and channel characteristics. However, the metric exhibits a noticeable ping-pong effect whose nature is still vague, and the very few references to this in the literature condemn it for being a perilous behavior.

In this paper, we present a thorough study of the Airtime ping-pong effect, and highlight its correlation to the underlying rate control algorithms. Using different rate control algorithms (e.g., ARF, AARF, ONOE, AMRR and Constant rate), we establish that transmission rate adaptation is the principal cause behind the effect. We show that the effect is an inherent behavior, and that an accurate characterization of it can help improve network performance.

We present a *ping-pong-aware* mechanism that, by detecting when a link undergoes such an effect, adapts the routing protocol for better network performance. The mechanism is O(1), decentralized, and can be easily integrated into the IEEE 802.11s routing protocol.

Keywords: Wireless mesh networks, IEEE 802.11s, Airtime, Hybrid wireless mesh protocol, Routing, Adaptive rate control algorithms.

1 Introduction

The IEEE 802.11s mesh networking standard [Ie1] is still undergoing process. However, HWMP (Hybrid Wireless Mesh Protocol) [Ba1] has been set as the default routing protocol, with Airtime as the default routing metric. Airtime accounts for loss rate, transmission rate, and channel characteristics. These latter depend on the underlying modulation technique, e.g., IEEE 802.11a/b/g. When an active link becomes loaded, HWMP "disfavors" the link by increasing its Airtime metric value. This increase would occur as consequence of a "detected" increase in the observed loss rate because the link is becoming loaded. Thus the routing protocol would advise selecting another unloaded link. However, the newly selected link becomes loaded after a while as well (especially in a congested network) urging the routing protocol to start the same procedure over

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 105–119, 2013.

again. This may merely result in re-selecting the first discarded link, especially when there is not enough link redundancy. In this manner, the procedure continues on, generating a "ping-pong" effect which consists on the cyclic procedure of switching back and forth between loaded and unloaded links.

By running extensive ns-3 simulations, we noticed that Airtime indeed exhibits a strong ping-pong effect. This latter consisted on having the network throughput frequently oscillating (during the life time of the experiments) between high and low throughput values. Furthermore, we noticed that Airtime exhibits a relatively weak ping-pong effect when used with a non-adaptive rate control algorithm, such as Constant rate.

When the link quality is degrading due to the link becoming loaded, adaptive rate control algorithms (e.g., ARF [KM1], AARF [LM1]) tend to decrease the transmission rate to avoid further losses. Since Airtime is inversely proportional to the transmission rate, this increases Airtime. On the other hand, when the link becomes unloaded, the adaptive rate control algorithms increase the transmission rate in order to profit from the good link quality (since the loss rate would have decreased), and Airtime decreases. In this way, we observe that the Airtime metric is *simultaneously* impacted by the variances in both the loss rate and the transmission rate. This renders Airtime very prone to frequent changes which are the main reason behind the effect.

In attempt to shed further light into the effect, we conducted a literature review, and we, found, surprisingly, only a single reference to the effect [GG1]. This reference superficially addresses the effect and condemns it as a perilous behavior without any solid clarification. Furthermore, the authors even changed the IEEE 802.11s formulation of Airtime [Ba2] to render it independent of the transmission rate and eliminate the "so-named" perilous behavior.

Given that the ping-pong effect is relevant to the overall network throughput by means of the Airtime routing metric which is an integral part to the IEEE 802.11s HWMP routing protocol, and given the very scarce literature on the effect, we think that this effect should be allotted further research, and that a good characterization of it can help improve the overall network performance by merely accounting for the effect's presence.

In this paper, we present a thorough study of the effect. Using extensive ns-3 simulations, we depict and analyze the effect. We show that the effect is an inherent behavior, and not necessarily a perilous one (by establishing that IEEE 802.11s can perform better under "intense" instances of the ping-pong effect). We also highlight the strong correlation between the effect and the underlying rate control algorithms, and establish that the latter are the main cause behind the effect. We allude also to the idea that an accurate characterization of the effect can benefit the IEEE 802.11s overall network performance by shaping *ping-ping-aware* mechanisms which account for the effect. In this context, we present a simple mechanism that, by detecting when the system is under the effect, reacts to it and adapts the routing protocol to improve network performance. The mechanism deems a multi-hop wireless path as a chain of one-hop links, whose strength is *equal* to the strength of its weakest link. The mechanism is O(1), decentralized, and easy-to-deploy. The proposed mechanism can reach up to 40% improvement in the overall network performance. This result is very encouraging towards further research

on the effect, and towards shaping further ping-pong-aware mechanisms (specifically at the level of routing and rate control).

The main contributions of this work are:

- Highlighting and characterizing the Airtime ping-pong effect.
- Proving that the effect is an inherent behavior and not forcibly a necessarily one.
- Proving the strong correlation of the effect to adaptive rate control algorithms.
- Suggesting a decentralized ping-pong-aware mechanism that improves the overall network performance.
- Encouraging further research on the effect, especially in the direction of shaping ping-pong-aware mechanisms that improve the overall network performance.

The rest of this paper is organized as follows. In Section 2, an overview of the IEEE 802.11s standard is presented. Section 3 presents and highlights the characterization process of the ping-pong effect. The analysis of the ping-pong effect characterization is covered in Section 4, and a ping-pong-aware mechanism is presented in Section 5. Finally, in Section 6, we conclude and present future work.

2 IEEE 802.11s: An Overview

Due to their easy-to-deploy and self-healing features, WMNs (Wireless Mesh Networks) [AW1] are emerging as a promising technology. WMNs are easy-to-deploy as setting a WMN involves minimal wiring and configuration overhead: Placing the WMN nodes and powering them *On* is all that is required to operate a WMN. On the other hand, WMNs are self-healing due to the redundancy of wireless links: A failure in a wireless link causes the network to seek alternative operational links, thus continuously maintaining the network.

WMNs may serve a rich set of applications, e.g., wireless community networks, wireless enterprise networks, transportation systems, home networking, and last-mile wireless Internet access. Providing last-mile wireless Internet access is one of the most promising applications, as WMNs tremendously reduce the cost and configuration overhead when compared with current solutions, e.g., Wi-Fi (IEEE 802.11) LANs.

In 2005, IETF (Internet Engineering Task Force) set a mesh networking TG (Task Group) to standardize IEEE 802.11s. The work is still in progress, with the last IEEE 802.11s TG meeting held in May, 2010 [Ie1]. Even though the standard is not yet final, its main traits are set, e.g., architecture and MAC routing. HWMP (Hybrid Wireless Mesh Protocol) [Ba1] is set as the default routing protocol for IEEE 802.11s compliant devices. HWMP uses MAC addresses for routing. IEEE 802.11s *Airtime* [Ba2] has been set as the default routing metric. These two amendments aim to maintain a minimum compatibility between the IEEE 802.11s devices to be manufactured by different companies.

2.1 IEEE 802.11s Architecture

IEEE 802.11s defines three types of stations:

1. MPs (Mesh Points): MPs are wireless stations that perform routing *only*.
2. MAPs (Mesh Access Points): MAPs are MPs with additional access point capabilities. Besides performing routing, MAPs aggregate traffic from/towards legacy 802.11 stations. A MAP can be thought of as a legacy access point which performs routing *as well*.
3. MPPs (Mesh Portal Points): MPPs are MPs that serve as gateways to other non-mesh networks, e.g., Internet. MPPs aggregate traffic from/towards the non-mesh networks.

2.2 IEEE 802.11s HWMP Routing Protocol

HWMP is an adaptation of the well-known AODV [PR1] protocol. It hybrids a reactive on-demand routing component with proactive tree-based routing [OT1] components. The two components can be used separately or simultaneously, depending on the type of the application for which the WMN is deployed.

RM-AODV (Radio-Metric Ad hoc On Demand Distance Vector) [AS1] is the reactive protocol in HWMP. RM-AODV is an adaptation of the AODV [PR1] protocol, using Airtime [Ba2] as a link quality metric. Reactive routing protocols initiate route discovery requests only when needed, such as in cases of route failure or route time-expiration.

The HWMP proactive mode [Ba1] is a tree based routing protocol [RC1, OT1].

In the proactive mode, every root mesh point (i.e., MPP) periodically broadcasts PREQ messages bearing unique sequence numbers. *Airtime* is a radio-aware metric that is designed to measure the amount of consumed channel resources when a frames is transmitted over a wireless link. Airtime is computed as follows:

$$Airtime = \left(O_{ca} + O_p + \frac{B_t}{r} \right) \frac{1}{1 - e_{fr}} \tag{1}$$

where O_{ca}, O_p, and B_t are constants quantifying the Channel Access Overhead, the Protocol Overhead, and the number of Bits in a probe frame, respectively. O_{ca} and O_p depend solely on the underlying modulation technique (see Table 1). r is the transmission rate (in Mbps) for a frame of size B_t, and e_{fr} is the frame error rate. IEEE 802.11s did not delineate a specific way to measure e_{fr}; It is left as an implementation issue.

Unlike ETX (Expected Transmission Count) [CA1] which accounts solely for frame error rate, Airtime accounts for both frame error rate and link bandwidth. This is also the case with ETT (Expected Transmission Time) [DP1]. However, Airtime further accounts for channel access and protocol overheads.

Table 1. IEEE 802.11s Airtime metric constants

	802.11a	**802.11b/g**
O_{ca}	75 μs	335 μs
O_p	110 μs	364 μs

3 Characterizing The Airtime Ping-Pong Effect

To characterize the ping-pong effect, we ran extensive ns-3 [On1] simulations using different settings, e.g., bandwidth, rate control algorithms, and topologies. The simulations establish the existence of such an effect. The next section highlights the experiments.

3.1 Experimental Settings

The WMN in this sample experiment is a 4x4 grid topology with a distance of 300 meters (see Figure 1). This simulates the case where WMNs are used for last-mile Internet access. Node(0) is set as a sink representing the WMN gateway, and nodes (3, 7, 15, 14, and 13) are set as sources representing WMN access points. The sources transmit UDP traffic, at a constant rate (10 Mbps), towards the sink. All source nodes start transmitting at the same time and continue transmission for periods of 60 seconds.

The WMN is configured to run the *proactive* HWMP routing protocol since the traffic is always directed towards the sink node. This shapes a tree-like topology with the sink as the tree root. The inter-time between the HWMP proactive PREQ (Path Request) broadcast messages is 2 seconds. We used the following rate control algorithms: ARF [KM1], AARF [LM1], AMRR [LM1], ONOE [Ma1], and Constant Rate.

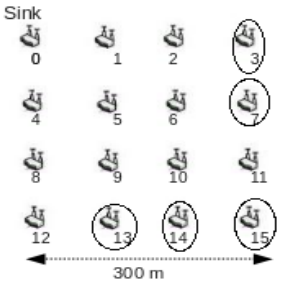


Fig. 1. Topology

3.2 Results

We tracked the variance of the *aggregated* received packets at the sink (i.e., the number of packets received from *all* five source nodes) by means of a packet counter (which was initially set to zero and incremented every time the sink receives a packet). After every time slot of 100 ms, the counter is saved and re-initialized back to zero. This process continues for the life time of the experiment (60 seconds), thus tracking the number of received packets in a total of 600 time slots. Figures 2, 3, and 4 plot the number of aggregated received packets (at the sink) for every 100 ms time slot and for each of the underlying rate control algorithms.

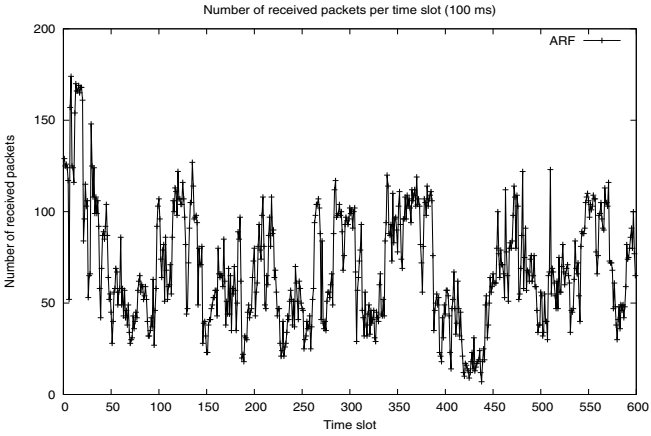


Fig. 2. ARF

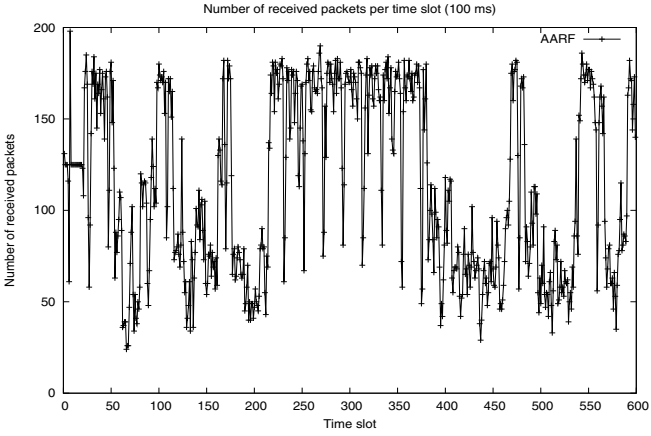


Fig. 3. AARF

In the last figures, we clearly notice that the different rate control algorithms are all exhibiting ping-pong effects, but with different magnitudes. To quantify these magnitudes, we computed the standard deviations (of the number of received packets per time slot) for the different rate control algorithms (see Figure 5).

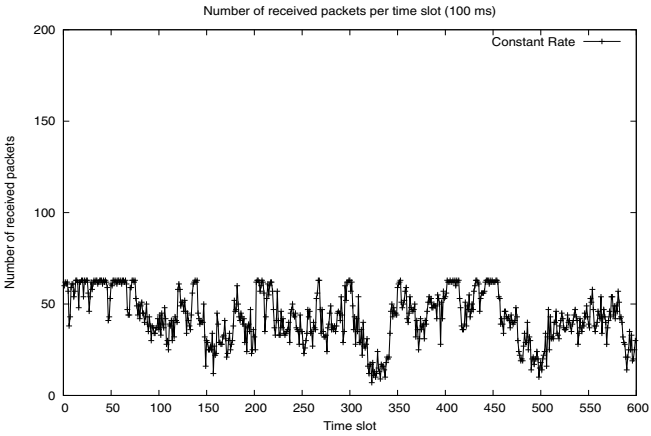


Fig. 4. Constant Rate

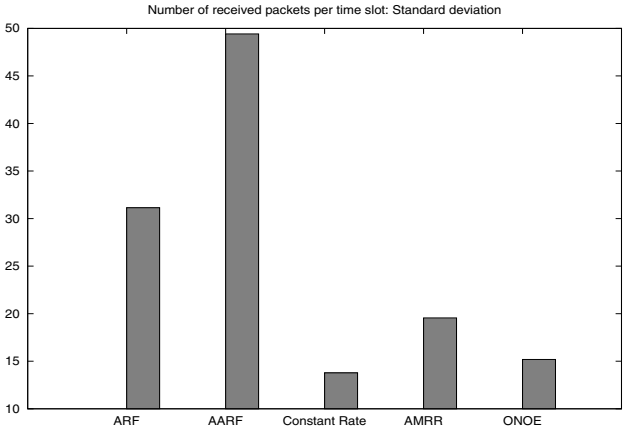


Fig. 5. Ping-pong effect magnitudes

In the last figure, we see that the different rate adaptation algorithms have, indeed, different ping-pong effect magnitudes. Airtime exhibits the strongest effect under AARF and ARF, and the weakest effect under Constant Rate.

This result was *general* to *all* experiments we ran except that ARF and AARF alternated in terms of which exhibited the strongest effect. Airtime always exhibited the weakest ping-pong effect when using Constant Rate. As a primary observation, we postulated the following:

1. There is a strong correlation between the ping-pong effect and the underlying rate control algorithm.
2. The ping-pong effect is not perilous.

These observations are discussed and analyzed in the following section.

4 Analysis

According to Equation (1), Airtime depends on both the loss rate (e_{fr}) and the transmission rate (r). With adaptive rate control algorithms [BW1], the transmission rate is continuously adjusted in order to cope with the varying link quality. This is mainly represented by the observed loss rate. Under good link quality, adaptive rate control algorithms increase the transmission rate in order to profit from the actual quality by injecting the maximum possible number of frames. However, and after a certain period of time the link quality "*forcibly*" degrades because of the *expected* increase in the link load:

- Since adaptive rate control algorithms increase the transmission rates as a reaction to good link quality, stations send frames at higher rates, thus generating more traffic/load.
- Since the transmission rate increased, Airtime decreases in order to reflect the actual good link quality (*Airtime is inversely proportional to transmission rate*). This urges the routing protocols of other stations to favor the link and route their traffic through it,thus generating more traffic/load.

Thus, every unloaded link is "condemned" to become loaded after a while (assuming there is enough traffic). When this occurs, the loaded link witnesses more frame losses, mainly because of congestion (due to high load). The link quality then starts deteriorating. In such a case, the adaptive rate control algorithms decreases the transmission rate in order to cut down the load. This is captured by Airtime through *both* increase in frame loss and decrease in transmission rate. The parallel change in both loss rate and transmission rate *"accelerates"* the procedure of increasing Airtime, thus advising the routing protocol of the local station (as well as of other stations) to disfavor the link and redirect traffic towards other less-loaded links.

In this way, adaptive rate algorithms largely impact the Airtime metric values and "*accelerate*" processes of both loading and unloading links. This acceleration intensifies the process of switching back and forth between loaded and unloaded links, thus intensifying the ping-pong effect. On the other hand, when no adaptive rate algorithm is used (i.e., Constant rate), the process of loading and unloading links is "*slower*". This is because Airtime is impacted by the *sole* variation in the observed loss rate, and not by variations in the transmission rate (as the latter remains constant).

Finally, we assert that the Airtime ping-pong effect is highly correlated to adaptive rate control algorithms, and that such a correlation is due to the induced changes in the transmission rates as a reaction to the observed link quality.

4.1 Is the Airtime Ping-Pong Effect a Perilous Behavior?

Being an exceptional behavior, the ping-pong effect tends to qualify itself as being perilous merely because of the "unusual" frequent oscillations in the network throughput. However, what we try to establish in this section, is that the effect is not as perilous as it seems [GG1].

By looking at the former numbers of received packets per time slot, see Figures 2, 3, and 4, we notice that the rate control algorithms that are exhibiting strong ping-pong effects (e.g., AARF and ARF) are oscillating between "*very*" high and low values of received packets (The low values are not qualified of "*very*" because they are quite in the same range ([0, 30]) for all rate control algorithms). To quantify these facts, we computed and plotted the corresponding network throughput values (see Figure 6). By

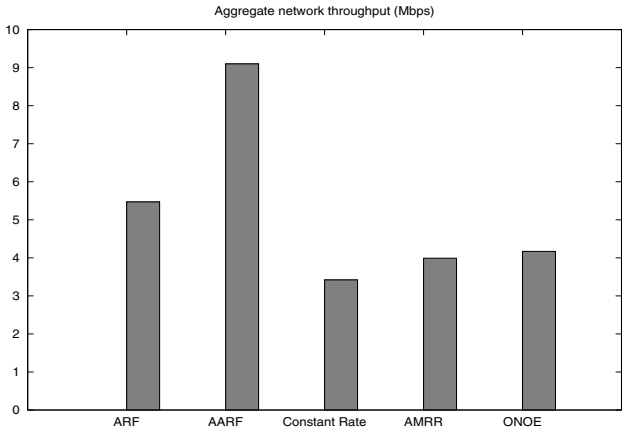


Fig. 6. Network Throughput Comparison

comparing these values to the corresponding ping-pong effect standard deviations (see Figure 5) we clearly see that the rate control algorithms that exhibit the strongest ping-pong effects are the ones that have the highest network throughput.

These facts sustain that the ping-pong effect is not a perilous (if not to say a healthier) behavior, since strong ping-pong effects (in the experiments) were exhibiting parallel strong throughput values. We further sustain this assertion by drawing the following analogy:

In the former section (Section 4), we outlined how the main cause behind the ping-pong effect is the "accelerated" process of switching back and forth between loaded and unloaded paths. This "acceleration" is due to the frequent changes in link transmission rate from the adaptive nature of the underlying rate control algorithm. In fact, when a link is loaded, it becomes very natural, and even a *must*, for it to be alleviated by reducing its load. On the other hand, when a link is unloaded, it becomes very beneficial for it to serve more traffic. Thus, we analogically compare the ping-pong effect to a "*breathing*" mechanism whereby systems, under harsh circumstances (analogous to a congested network), are repeatedly, and strongly, switching back and forth (ping-pong effect) between the inhalation (more load) and exhalation (less load) processes. More thoroughly and precisely, a steady and "harmonious" switching behavior (even if strong) is a symptom of a "healthier" system. Such was the case with AARF in the

former experiment (see Figure 3), where the switching back and forth was indeed strong (e.g., high received packets) and frequent (i.e., high standard deviation, see Figure 5). Here, the corresponding network throughput was the highest as well. To further generalize these assertions, we conducted further experiments, which are highlighted in the next section.

4.2 Generalization

In these experiments, five sources nodes, whose locations and transmission starting times are randomly picked, are deployed in the same 4x4 grid topology as in Figure 1. Node(0) always represents the sink, and the sources traffic generation rates are made variable in order to experiment with different network loads and congestion levels.

Using 10 different scenarios, 10 different traffic generation rates, and 3 different rate control algorithms, we ran a total of 300 experiments. Table 2 depicts the settings of the randomly generated scenarios. The experiments were run for a duration of 60 seconds.

Table 2. Scenarios Settings

Scenario Seq#	(Node#, Node transmission start time (Sec))				
1	(3,0)	(4,9)	(11, 8)	(12, 38)	(10, 21)
2	(8,0)	(3,26)	(4, 9)	(9, 39)	(1, 7)
3	(4,0)	(13,24)	(9, 5)	(6, 31)	(11, 7)
4	(8,0)	(3,18)	(7, 12)	(5, 15)	(15, 30)
5	(3,0)	(14,23)	(1, 20)	(9, 15)	(4, 9)
6	(5,0)	(13,8)	(6, 6)	(8, 12)	(14, 25)
7	(13,0)	(7,21)	(2, 39)	(4, 32)	(10, 5)
8	(12,0)	(14,19)	(11, 27)	(7, 23)	(5, 13)
9	(5,0)	(6,1)	(13, 22)	(8, 5)	(3, 6)
10	(9,0)	(6,2)	(11, 19)	(4, 9)	(2, 31)

Note that the transmission times were randomly generated in the [0, 40] seconds interval in order to always keep the last 20 seconds witnessing simultaneous transmissions of all existing nodes. Also, the first randomly picked node is always transmitting at second zero.

Figure 7 depicts the *average* number of received packets (at the sink) for each of the 10 scenarios, using the three different rate control algorithms (ARF, AARF, and Constant rate). The average is obtained by using 10 different source traffic generation rates ranging from 1 to 10Mbps with a 1 Mbps step. We notice that Airtime is performing considerably better with ARF and AARF than with Constant Rate. This holds for all 10 scenarios. This generalizes the result in Section 4.1 (see Figure 6) about having the worst Airtime performance when using Constant rate.

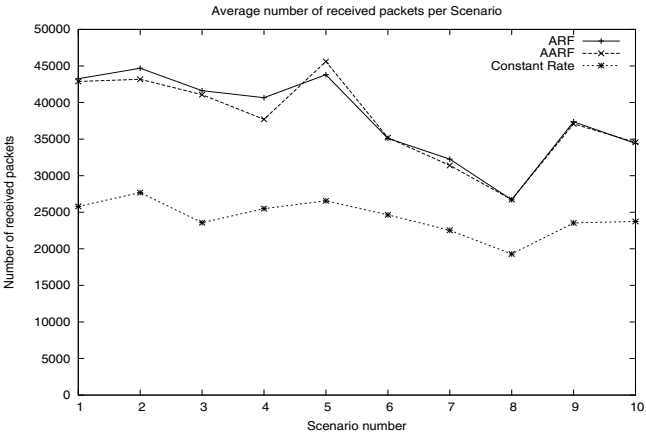


Fig. 7. Scenarios throughput comparison

5 Ping-Pong-Aware Mechanisms

Since the Airtime ping-pong effect is an inherent behavior in IEEE 802.11s wireless mesh networks, it becomes necessary to (*re*)shape mechanisms to be aware of such an effect. To our knowledge, there no such mechanism in literature so far.

Every *ping-pong-aware* mechanism, regardless of its functionality, should first be able to detect when a link is witnessing, or about to witness, a ping-pong effect. Hence, a crucial prerequisite step toward designing ping-pong-aware mechanisms is the "accurate" detection of the effect occurrence.

The proposed ping-pong-aware mechanism operates at the routing level. The mechanism is O(1) and decentralized. The rationale behind the mechanism is two-fold:

- Boost the network to decrease the high loss rate during the Ping phase due to the link becoming loaded.
- Boost the network to further increase the throughput by profiting from the very good link quality witnessed in the Pong phase.

The mechanism accomplishes its goals by the following:

If a link is detected to be in the Ping phase, we further increase its Airtime metric value. This is done to prevent, as much as possible, the routing protocol from forwarding further frames through the link, and thus the loss rate is decreased. The idea behind this is to make the link look worse by giving it more weight, thus labeling it as a candidate "*weakest chain link*": The strength of a chain is *equal* to the strength of its weakest link. By analogy, a multi-hop wireless path is a chain of one-hop links. However, the way most routing protocols measure the "strength" (i.e., quality) of a multi-hop path is by summing up the "strengths" links [CA1, DP1]. These strengths correspond to the numeric values of the underlying routing metrics. This does not go along with the "weakest link" fact. To depict this fact, let us formally denote a three-hops path P which is made of the following one-hop links: (a, b), (b, c), and (c, d), where a, b,

c, and d denote four wireless stations. Let Q_P, $Q_{(a,b)}$, $Q_{(b,c)}$, and $Q_{(c,d)}$ denote the corresponding link quality metric values (e.g., Airtime). Let us assume the following two cases:

- Case 1: $Q_{(a,b)}$ = 40, $Q_{(b,c)}$ = 50, $Q_{(c,d)}$ = 60;
  Q_P=$Q_{(a,b)}$+$Q_{(b,c)}$+$Q_{(c,d)}$=150;
- Case 2: $Q_{(a,b)}$ = 100, $Q_{(b,c)}$ = 10, $Q_{(c,d)}$ = 10;
  Q_P=$Q_{(a,b)}$+$Q_{(b,c)}$+$Q_{(c,d)}$=120;

The routing protocol definitely favors the path in Case 2 as it has a much better Airtime value (smaller value) than in Case 1. However, the path in Case 2 ought to be less favored as it has the worst "weakest link" ($Q_{(a,b)}$ = 100). This would constitute a strong bottleneck which greatly affects the path throughput .

In most routing protocols, individual links are given the same weight regardless of being the weakest or not. In the proposed ping-pong-aware mechanism we try to give more weight to the weakest links by presuming that a link which is in the Ping phase is a de facto "weakest link" (as it is witnessing a sharp decrease in the throughput).

On the other hand, when a link is in the Pong phase, we further decrease its metric in order to "attract" more frames. In this way, we profit from the exceptional link quality and increase the throughput.

We set the step by which we should increase/decrease the Airtime metric to be equal to the step by which the metric last decreased/increased. Alg. 1 depicts the procedure.

Alg. 1. Ping-pong aware mechanism Algorithm

compute $Airtime$
$Ping \Leftarrow 0$
$Pong \Leftarrow 0$
if $loss_{current} > loss_{previous}$ ***and*** $rate_{current} < rate_{previous}$ **then**
    $Ping \Leftarrow 1$
elseif $err_{current} < err_{previous}$ ***and*** $rate_{current} > rate_{previous}$ **then**
    $Pong \Leftarrow 1$
if $Ping = 1$ ***or*** $Pong = 1$ **then**
    get $Airtime_{previous}$
    $Airtime+ = Airtime - Airtime_{previous}$
    if $Airtime < 0$ **then**
        $Airtime \Leftarrow 0$
return $Airtime$

5.1 Experiments

Using ns-3, we implemented the ping-pong-aware algorithm as a module in the HWMP routing protocol. We ran the same experiments, as in Section 4.2, and we compared the network performance and behavior under ARF and AARF (as rate adaptation algorithms), and with and without using the ping-pong-aware mechanism (PPAM). Figures 8 and 9 depict the average number of received packets at the sink.

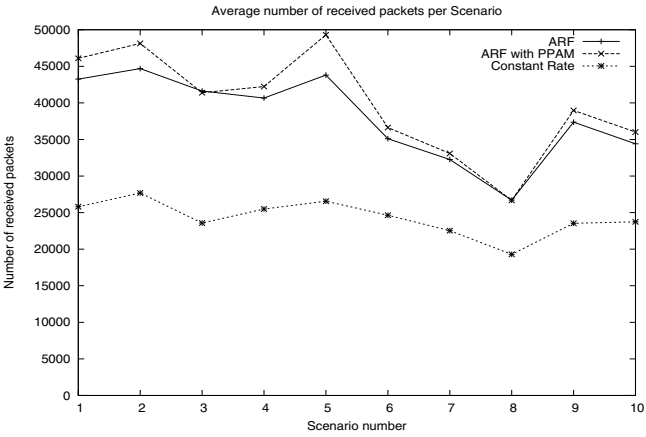


Fig. 8. Throughput comparison: ARF vs. ARF with PPAM

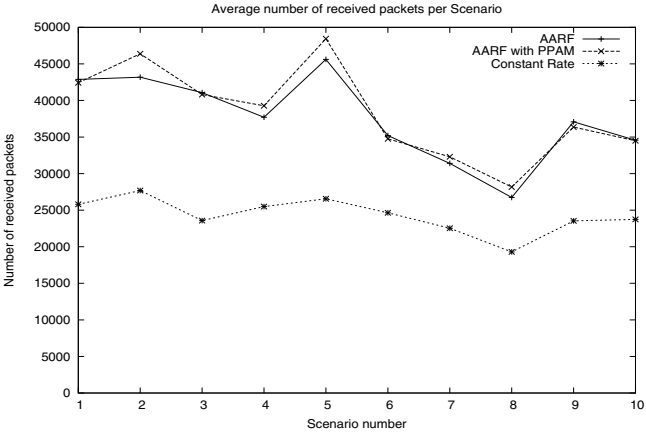


Fig. 9. Throughput comparison: AARF vs. AARF with PPAM

From the latter figures, we notice that the network is performing better under the ping-pong-aware mechanism (for both ARF and AARF). However, the improvement is on average a slight one: 5% for ARF and 4% for AARF. Nevertheless, the algorithm performance reaches up to 40% improvement in some scenarios with specific traffic generation, e.g., for scenario 5 (see Table 2) with a 7 Mbps traffic rate.

However, this is still proof of concept for this new research direction in suggesting and shaping ping-pong-aware mechanisms. We think that better ping-pong detection algorithms and more efficient ping-pong-aware mechanisms can be shaped and at different level (especially at the level of rate control).

6 Conclusion and Future Work

We presented a thorough characterization of the Airtime ping-pong effect, and provided an analytical study for its causes as well as its behavior. We highlighted its high correlation to the underlying rate control algorithms by using different algorithms (e.g., ARF and AARF), and we showed that it is an inherent behavior stemming from links alternating between the loaded and unloaded status. Contrary to the available literature, we proved that the Airtime ping-pong effect is not a perilous one. In fact, a good characterization of it can help in shaping ping-pong-aware mechanisms. which can improve network performance. The shaping of such mechanisms relies on the crucial prerequisite of detecting when a link undergoes such an effect.

We proposed a ping-pong-aware mechanism that is O(1), decentralized, and that can easily be integrated into the IEEE 802.11s routing protocol. The mechanism proved to have a slight increase in the average overall network throughput. This showed how the ping-pong effect can be adapted for a better network performance. By doing so, we are suggesting a novel research direction which is based on deeming the ping-pong effect as an inherent behavior, thus introducing the shaping of ping-pong-aware mechanisms that can cope with the effect for the improvement of network performance improvement.

In a future work, we intend to continue further research in this direction, especially improving the ping-pong effect detection mechanism, shaping more efficient ping-pong-aware mechanisms, and integrating the latter into other levels (especially Rate Control).

References

[Ba1] Bahr, M.: Update on the Hybrid Wireless Mesh Protocol of IEEE 802.11s. In: IEEE Conference on Mobile Adhoc and Sensor Systems, pp. 1–6 (2007)

[Ba2] Bahr, M.: Proposed Routing for IEEE 802.11s WLAN Mesh Networks. In: The 2nd Annual International Wireless Internet Conference, WICON, pp. 6–13 (2006)

[On1] Online: The ns-3 network simulator, `http://www.nsnam.org/`

[Ie1] IEEE TGs: Status of Project IEEE 802.11s (2011), `http://www.ieee802.org/11/Reports/tgs_update.htm`

[AS1] Aoki, H., Shinji, T., Kengo, Y., Akira, Y.: IEEE 802.11s Wireless Mesh Network Technology. IEEE NTT DoCoMo Technical Journal 8, 13–21 (2006)

[PR1] Perkins, C., Royer, E.: Ad-hoc On-demand Distance Vector Routing. In: The 2nd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA, pp. 90–100 (1999)

[RC1] Raniwala, A., Chiueh, T.: Architecture and Algorithms for an 802.11-Based Multi-Channel Wireless Mesh Network. In: Proc. of IEEE INFOCOM, vol. 3, pp. 2223–2234 (2005)

[CA1] De Couto, D., Aguayo, D., Bicket, J., Morris, R.: High-throughput path metric for multi-hop wireless routing. In: ACM Annual International Conference on Mobile Computing and Networking, MOBICOM, pp. 134–146 (2003)

[GG1] Garroppo, R., Giordano, S., Iacono, D., Tavanti, L.: Notes on implementing a IEEE 802.11s mesh point. Elsevier Computer Communications 33, 336–349 (2010)

[AW1] Akyildiz, F., Wang, W.: Wireless mesh networks: a survey. Computer Networks and ISDN Systems 47, 445–487 (2005)

[DP1] Draves, R., Padhye, J., Zill, B.: Routing in multi-radio, multi-hop wireless mesh networks. MOBICOM. In: ACM Annual International Conference on Mobile Computing and Networking, pp. 114–128 (2004)

[OT1] Orgier, R., Templin, F., Lewis, M.: Topology dissemination based on reverse-path forwarding (TBRPF). RFC 3684. IETF (2004)

[BW1] Biaz, S., Wu, S.: Rate adaptation algorithms for IEEE 802.11 networks: A survey and comparison. In: IEEE Symposium on Computers and Communications, pp. 130–136 (2008)

[KM1] Kamerman, A., Monteban, L.: WaveLAN II: A high-performance wireless LAN for the unlicensed band. Bell Labs Technical Journal, 118–133 (1997)

[LM1] Lacage, M., Manshaei, M., Turletti, M.: IEEE 802.11 Rate Adaptation: A Practical Approach. In: Proc. of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 126–134 (2004)

[Ma1] Madwifi: The Madwifi Project, `http://sourceforge.net/projects/madwifi`

Planning UMTS Base Station Location Using Genetic Algorithm with a Dynamic Trade-Off Parameter

Mohammed Gabli, El Miloud Jaara, and El Bekkaye Mermri

Department of Mathematics and Computer Science, Faculty of Science, University Mohammed Premier, BV Mohammed VI, Oujda, Morocco
{medgabli,emjaara}@yahoo.fr,
b.mermri@fso.ump.ma
http://sciences1.univ-oujda.ac.ma/

Abstract. In this paper, we address the problem of planning the universal mobile telecommunication system (UMTS) base stations location for uplink direction. The objective is to maximize the total trafic covered f and minimize the total installation cost g. This problem is modelled in the form of multi-objective optimization problem that can be transformed into a mono-objective problem of the form $f + \lambda g$, where $\lambda > 0$ is a trade-off parameter between the objective functions f and g. Our aim here is to present a solution method to the problem based on a genetic algorithm (GA), which automates the choice of the parameter λ by varying it at each iteration of the algorithm. To apply the GA to our problem, we have proposed a special coding that combines the binary and integer coding. To validate the proposed method some numerical examples are given. The obtained results show the efficiency of our approach.

Keywords: UMTS, Multi-objective optimization, Genetic algorithm.

1 Introduction

Universal Mobile Telecommunications System (UMTS) is a third generation mobile cellular technology for networks based on the Global System for Mobile Communications standard (GSM). The deployment of UMTS networks involves a colossal investment for the operators. In this context, the optimization of these networks becomes, for an operator, a fundamental task.

The problem of planning second-generation cellular systems adopting a time-division multiple access (TDMA)-based access scheme has usually been simplified by subdividing it into a coverage planning problem and a frequency planning problem which are driven by a coverage and a capacity criterion, respectively [1,2,3]. With the wideband code-division multiple access (W-CDMA) air interface of UMTS, this two-phase approach is not appropriate mainly because the bandwidth is shared by all active connections and no actual frequency assignment is strictly required. The access scheme allows for a more flexible use

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 120–134, 2013.

of radio resources and the capacity of each cell (e.g., the number of connections) is not limited a priori by a fixed channel assignment as in TDMA systems, but it depends on the actual interference levels which determine the achievable signal-to-interference ratio (SIR) values. As these values depend on both traffic distribution and base stations (BSs) positions, BS location in UMTS networks cannot only be based on coverage but it must also be capacity driven [1,2]. Furthermore, since interference levels depend both on the connections within a given cell and on those in neighboring cells, the SIR values and the capacity are highly affected by the traffic distribution in the whole area [1].

1.1 Related Work

UMTS networks planning problems have been the interest of many researchers. St-Hilaire et al. [4] proposed a global approach for planning UMTS networks in the uplink direction. In [5] the objective is to determine the cost-optimal number and location of the Radio Network Controller (RNC) nodes and their connections to the Radio Base Stations (RBS) according to a number of planning constraints, by using a tree topology. Hashemi et al. [6] examined the same problem but this time by using the hybrid ant colony algorithm. Meunier [7] introduced a multicriteria model for the design of mobile telecommunication networks. In [8] the problem of optimization is defined by the adjustment of parameter of UMTS networks. Amaldi et al. [1] studied the UMTS base station (BS) location based on propagation models with power control. This problem has two objective functions to optimize: maximize the total trafic covered and minimize the total installation cost. To solve the problem authors proposed two randomized greedy procedures and a tabu search algorithm.

In the literature, a special attention is given to the problems of two criteria using exact and approximate (heuristic) algorithms. Exact methods such as Branch and Bound, the A* algorithm and Dynamic Programming are effective for problems of small sizes. When problems become harder, usually because of their NP-hard complexity, approximate algorithms are mandatory. Several adaptations of metaheuristics have been proposed in the literature for solving multi-objective problems (MOP): simulated annealing, tabu search, genetic algorithms and evolutionary strategies (see for instance [9,7]). The telecommunication area has been one of the most exciting domain in which multi-objective metaheuristics have been applied [10]. Several methods using genetic algorithms (GAs) have been developed for solving MOP, for example, VEGA [11], MOGA [12], NPGA [13], SPEA [14] and others. The approaches used for solving MOP can be classified into three categories: approaches based on the transformation of the problem into a mono-objective problem, non-Pareto approaches and Pareto approaches (see [15,16,17]).

In this paper, we are interested in the UMTS base station location problem presented in [1,18]. The problem is modelled as a mono-objective optimization problem with a trade-off parameter λ fixed. When we transform a multi-objective optimization problem into a mono-objective one, the choice of the trade-off parameter λ is not an easy task for both the decision maker and the system

analyser. To remedy this problem we introduce a solution method based on a genetic algorithm approach which automates the choice of the trade-off parameter λ at each iteration of the algorithm. In Section 2, we describe the problem and we present its formulation. In Section 3, we introduce a GA approach using a dynamic trade-off parameter. In Section 4, we present an encoding method for the GA based on a combination of binary and integer coding, then we describe the crossover and mutation operators. In Section 5 we give an application of our approach to some problems, then we present the obtained numerical results. Finally, in Section 6 we give some concluding remarks.

2 Problem Statement and Model Presentation

Consider a territory to be covered by a UMTS service. Let $S = \{1, ..., m\}$ be a set of candidate sites (CS) where a base station (BS) can be installed and $I = \{1, ..., n\}$ be a set of test points (TPs). Each base station BS_j, $j \in S$, has a cost of installation denoted by $\mathbf{c_j}$. We denote by $\mathbf{u_i}$ the required number of simultaneously active connections for a TP of index i (TP_i). In this section we will need the following notations:

$P_{\text{reçue}}$	received power
P_{target}	target power
P_{max}	maximum power
SIR	The signal-to-Interference Ratio
H_b	height of the base (in meters)
H_m	height of the mobile station (in meter)
F	signal frequency (in megahertz)
dB	decibels
dBm	power ratio in dB of the measured power referenced to 1 milliwatt
CS	candidate site
TP	test points
BS	base station
g_{ij}	propagation factor of the radio link between TP_i and CS_j
d_{ij}	distance between TP_i and candidate site CS_j
SF	the ratio between the spread signal rate and the user rate

2.1 Mechanism of Power Control (PC)

In UMTS networks, different users can transmit in the same frequency band. Assume that a mobile terminal communicates with a base station, and emits at a too high power. Then there is a risk to prevent the other mobile terminals of the cell to communicate with this base station. To remedy this problem a fast power control is essential. On the other hand, each user can be a source of interference to the others. Thus it is important to implement a mechanism which allows to a mobile terminal to adjust its power of emission while guaranteeing a good reception of the base station. This power problem also arises for the

power emitted by the base station to limit the intercellular interferences. Two PC mechanisms are commonly considered [1]:

1. *PC mechanism based on the received power:* The transmitted power is adjusted so that the power received on each channel is equal to a given target value P_{target}.
2. *PC mechanism based on the estimated SIR:* The transmitted power is set so that the SIR is equal to a target value SIR_{target}

2.2 Radio Propagation

The propagation channel in a mobile radio environment is mainly related to the type of environment to be considered, urban, rural, indoor, etc.; and to physical phenomena that the wave undergoes during the propagation namely reflection, diffraction and diffusion. In this paper we consider the Hata's propagation model presented in [19], which gives the attenuation due to the signal propagation. In particular, the attenuation between a BS_j and TP_i for urban areas, measured in decibels (dB) and denoted by L_u, is given by (see [19]):

$$\begin{aligned} L_u(d_{ij}) = 69.55 + 26.16\log(F) - 13.82\log(H_b) \\ -a(H_m) + [44.9 - 6.55\log(H_b)]\log(d_{ij}), \end{aligned} \tag{1}$$

where the parameter $a(H_m)$ is a correction factor depending on the height of the antenna of the mobile station and on the environment. The value of $a(H_m)$ is given by:

– For a medium sized city:

$$a(H_m) = (1.1\log(F) - 0.7)H_m - (1.56\log(F) - 0.8) \tag{2}$$

– For a large city:

$$a(H_m) = 3.2(\log(11.75H_m)]^2 - 4.97 \tag{3}$$

2.3 Model Presentation

Let $S = \{1, ..., m\}$ be a set of candidate sites (CS) where a base station can be installed and $I = \{1, ..., n\}$ a set of test points. Each base station BS_j has a cost of installation denoted by c_j. Let u_i be the required number of simultaneously active connections for a TP_i. Let us define the two following classes of decision variables:

$$y_j = \begin{cases} 1 & \text{if a } BS \text{ is installed in a site } j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } j \in S, \tag{4}$$

and

$$x_{ij} = \begin{cases} 1 & \text{if a } TP_i \text{ is assigned to a } BS_j, \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } i \in I \text{ and } j \in S. \tag{5}$$

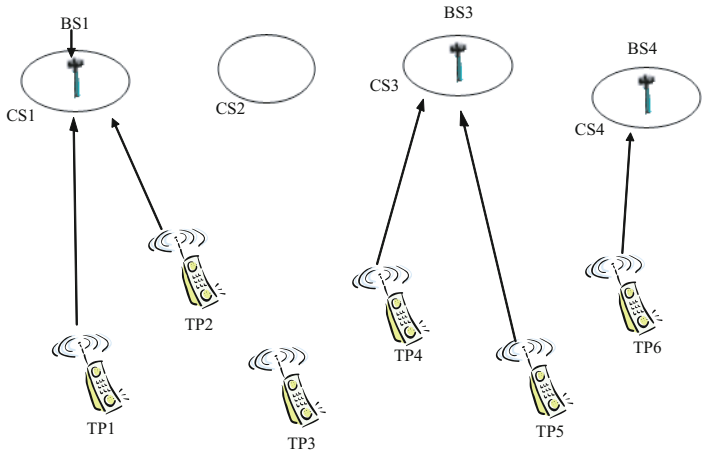


Fig. 1. Illustration: problem with 6 TPs and 4 CSs

We see that the variable x depends on y. An illustrative example is presented in Figure 1. In this example we have four CSs and six TPs. We see that the BS_2 is not installed and the TP_3 is not covered; TP_1 and TP_2 are assigned to BS_1; TP_4 and TP_5 are assigned to BS_3 and TP_6 is assigned to BS_4. We consider a power-based PC mechanism. Suppose we have directive BSs with three identical 120 degree sectors and with an omnidirectional antenna diagram along the horizontal axis. Let the index set $I_j^\sigma \subseteq I$ denotes the set of all TPs that fall within the sector σ of the BS installed in the candidate site CS_j. Since we wish to maximize the total trafic covered and minimize the total installation cost subjected to some constraints, then the problem can be expressed as [1,18]:

$$\begin{cases} \text{Maximize } f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} u_i x_{ij}, \\ \text{Minimize } g(y) = \sum_{j=1}^{m} c_j y_j, \end{cases} \tag{6}$$

subject to:

$$\sum_{j=1}^{m} x_{ij} \leq 1 \ , \quad i \in I, \tag{7}$$

$$x_{ij} \leq \min\{1, \frac{g_{ij} P_{max}}{P_{target}}\} y_j, \quad i \in I \ , j \in S, \tag{8}$$

$$y_j \sum_{i \in I_j^\sigma} \sum_{t=1}^{m} (\frac{u_i g_{ij}}{g_{it}} x_{it} - 1) \leq \frac{SF}{SIR_{min}}, \quad j \in S, \sigma \in \Sigma, \tag{9}$$

$$x_{ij} \ , \ y_j \in \{0,1\}, \quad i \in I, j \in S. \tag{10}$$

Where the propagation factor of the radio link between a TP_i and a candidate site CS_j is given by:

$$g_{ij} = (10^{\frac{L_u(d_{ij})}{10}})^{-1},$$

where the attenuation L_u is calculated by relation (1).
In [1,18] authors have transformed the multi-objective problem (6) into a mono-objective one as follows:

$$\text{Maximize} \sum_{i=1}^{n} \sum_{j=1}^{m} u_i x_{ij} - \lambda \sum_{j=1}^{m} c_j y_j, \tag{11}$$

Subject to the constraints (7), (8), (9) and (10), where $\lambda > 0$ is a trade-off parameter between maximizing coverage and minimizing costs. Here the parameter λ is taken a constant. In Section 3, we show that, when using *GA* to solve problem (11), is not always appropriate to take λ fixed. Instead, we introduce a dynamic trade-off parameter.

3 *GA* and Dynamic Trade-Off Parameter

Consider the optimization model (11): $h = f + \lambda g$. Assume that f is much greater than g. When applying GA to maximize the objective function h, if we take the trade-off parameter λ constant, then there is a great risk that the procedure of selection of the GA chooses only solutions which improve f by neglecting g, since the function f dominates g.

3.1 Limits of Choosing λ Fixed

Let $[a, b]$ be a real interval and f and g be two real functions satisfying:

$$10^2 \leq f(x) \leq 10^4 \quad \text{and} \quad 0 \leq g(x) \leq 1, \quad x \in [a, b].$$

The problem of maximizing f and maximizing g can be transformed into the maximization of the function h defined by:

$$h(x) = f(x) + \lambda g(x), \quad x \in [a, b].$$

For example, let λ be chosen equal to 1000. Then we have

$$10^2 \leq h(x) \leq 10^4 + 10^3, \quad x \in [a, b].$$

Suppose that in the iteration k of the GA, the solution x_k satisfying: $f(x_k) = 10^3$ and $g(x_k) = 0.1$ is retained as a solution for h, i.e. the best among solutions of the current population. Then we have:

$$h(x_k) = f(x_k) + \lambda g(x_k) = 10^3 + 10^2 = 1100.$$

Assume that in the next iteration of the GA, we find two solutions $x_{k+1}^{(1)}$ and $x_{k+1}^{(2)}$ satisfying:

$$\begin{cases} f(x_{k+1}^{(1)}) = 5 \times f(x_k) \\ \\ g(x_{k+1}^{(1)}) \approx g(x_k) \end{cases}$$

$$\begin{cases} f(x_{k+1}^{(2)}) \approx f(x_k) \\ \\ g(x_{k+1}^{(2)}) = 5 \times g(x_k) \end{cases}$$

Then we have:

$$h(x_{k+1}^{(1)}) = f(x_{k+1}^{(1)}) + \lambda g(x_{k+1}^{(1)}) \approx 5 \times 10^3 + 10^2 \approx 5100$$

and

$$h(x_{k+1}^{(2)}) = f(x_{k+1}^{(2)}) + \lambda g(x_{k+1}^{(2)}) \approx 10^3 + 5 \times 10^2 \approx 1500.$$

Therefore

$$\frac{h(x_{k+1}^{(1)})}{h(x_k)} \approx 5100/1100 \approx 5 \quad \text{and} \quad \frac{h(x_{k+1}^{(2)})}{h(x_k)} \approx 1500/1100 \approx 1$$

It is clear that although $x_{k+1}^{(1)}$ and $x_{k+1}^{(2)}$ improve, respectively, the objective functions f and g in the same way, the probability that GA selects $x_{k+1}^{(2)}$ is weak compared to the probability of selecting $x_{k+1}^{(1)}$. When one takes λ fixed, the fact that an objective function dominates another is extremely probable. In this example, a solution which improves the value of g does not have the same influence on maximizing h as one which improves the value of f.
To remedy this problem we should not take the value of λ fixed, but rather this value must be dynamic and it changes in each iteration of the *GA*.

3.2 Algorithm

Now, we present a *GA* algorithm, applied to the optimization model (11), using a dynamic trade-off parameter λ. Let $h = f + \lambda g$ be the fitness function of the GA. In each iteration i of the GA we take:

$$\lambda_i = |\frac{f(x_{i-1})}{g(x_{i-1})}|,$$

where x_{i-1} is the best individual among solutions of the current population $P(i-1)$, with respect to the fitness function h. Then the algorithm is outlined as follows:

begin
  At the initialization step of the GA assign a positif integer to λ;
  repeat
    Run an iteration of the GA;
    Let x^* be the best solution among solutions of the current population;
    Calculate $f(x^*)$ and $g(x^*)$;
    if $g(x^*) \neq 0$ then
      Take $\lambda = |f(x^*)/g(x^*)|$.
    endif
  until a stopping criterion is satisfied
end.

Our algorithm has two immediate advantages:

1. It automates the choice of the trade-off parameter λ. The utility resides, therefore in the fact that we do not need to define this factor in advance. This task turns out to be a very delicate question.
2. It ensures an equitable treatment of each objective function, so we have an equitable chance to maximize both functions f and g. Indeed, the function $h(x) = f(x) + \lambda g(x)$ becomes:

$$h(x) = f(x) + |\frac{f(x^*)}{g(x^*)}|g(x).$$

Where x is a solution of h for the current iteration (i) and x^* is the best solution of h selected for the previous iteration $(i-1)$.
When using the method of scaling for example, the difference between $f(x)$ and $f(x^*)$, also between $g(x)$ and $g(x^*)$ is reduced, therefore the two terms of the function h will be close to each other.

3.3 Application of the Algorithm

Consider the example of the objective functions f and g presented in Section 3.1. We apply the proposed algorithm to maximize $h = f + \lambda g$.

– Let x^* be the best solution among solutions of the population in the current iteration of the GA. Assume that the solution x^* satisfies $f(x^*) = 10^3$ and $g(x^*) = 0.1$. Then the trade-off is defined as:

$$\lambda = \frac{f(x^*)}{g(x^*)} = \frac{1000}{0.1} = 10000.$$

$$h(x^*) = f(x^*) + \lambda g(x^*) = 10^3 + 10^3 = 2 \times 10^3.$$

– Then, in the next iteration of the GA we have to maximize the function $h(x) = f(x) + 10^4 g(x)$. Let x_1 and x_2 be two solutions such as:

$$\begin{cases} f(x_1) = 5 \times f(x^*) \\ g(x_1) \approx g(x^*) \end{cases}$$

$$\begin{cases} f(x_2) \approx f(x^*) \\ g(x_2) = 5 \times g(x^*) \end{cases}$$

Then we have:

$$h(x_1) = f(x_1) + \lambda g(x_1) = 5 \times 10^3 + 10^3 = 6 \times 10^3$$

$$h(x_2) = f(x_2) + \lambda g(x_2) = 10^3 + 5 \times 10^3 = 6 \times 10^3$$

It is clear that the probabilities to select x_1 and x_2 are nearly equal. The solution x_2 thus has almost the same chances, as x_1, to be selected in the next generation of the GA, which is not the case with the choice of λ fixed.

4 GA Approach

4.1 Chromosome Representation

To code the base stations we can successfully use the binary encoding. For m base stations we introduce a sequence of m binary digits, where each digit in a position i, d_i, from left to right, indicates that the BS_i is installed if $d_i = 1$, otherwise the BS_i in not installed. For example, if $m = 7$, the code 0111001 means that we have to install BSs numbered $2, 3, 4$ and 7. As for encoding the TPs, the binary coding is not appropriate any more. To remedy this problem we used integer coding. For n test points we introduce a sequence of n digits, where each digits is an integer taking values between 0 and m. If the digit in a position j, takes a value k, $d_j = k$, that means the TP_j is assigned to the BS_k; if $k = 0$ the TP_j is not assigned to any base station. For example, if $n = 13$, the code 4;3;1;0;1;7;4;7;2;3;2;5;6 means that the TP_1 is assigned to the BS_4, the

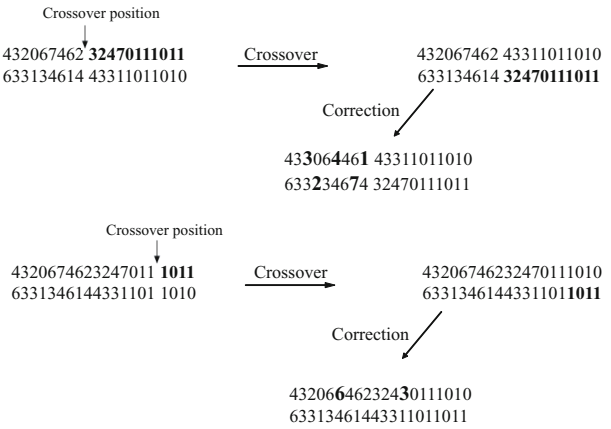


Fig. 2. Illustration of crossover

TP_2 assigned to the BS_3, $\cdots$, the TP_{13} assigned to the BS_6. We see that the TP_4 is not assigned to any BS, this means that the TP_4 is not covered.

To encode each chromosome we use a combination of the two above codes. If we have m base stations and n test points, the chromosome will have $m + n$ genes, where the first m genes are formed by the code of the base stations and the remanning digits are formed by the test points code. Hence each chromosome will indicate if a TP is covered or not, and to which BS is assigned. For example, if we have $m = 7$ and $n = 13$ the chromosome can be encoded as: 4;3;1;0;1;7;4;7;2;3;2;5;6;0111001. Finally, we must always take care not to fall in the case where a TP is assigned to a BS that is not installed. To do this we use a small correction which reassigns the TP to another BS.

4.2 Initial Population, Crossover and Mutation

- *Initial population.* Suppose we have n TPs and m BSs. To define each chromosome of the population we generate $n+m$ random genes; the first n genes are integers in the set $\{0, \ldots, m\}$ and the remaining m genes are binary digits. Then, we use the correction procedure defined above.
- *Crossover.* We use the usual crossover followed by the procedure of correction. Figure 2 illustrates this operation.
- *Mutation.* We used the usual mutation followed by the procedure of correction. If the gene to mutate is a TP, we replace it by an integer chosen randomly from the set $\{1, 2, \ldots, m\}$, otherwise we transform the 0 to 1 and the 1 to 0 in the selected gene. Figure 3 illustrates these two cases.

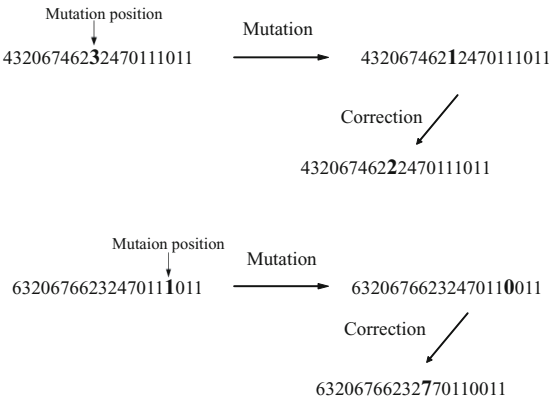


Fig. 3. Illustration of mutation

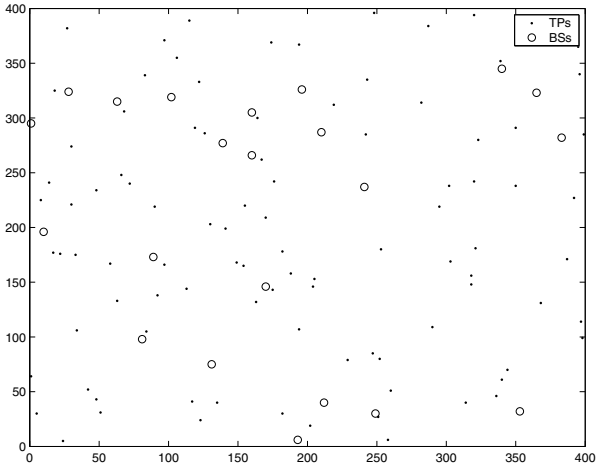


Fig. 4. First instance: Location of 95 TPs and 22 BSs in a service area of $0.4 \times 0.4(Km)$

5 Application

5.1 Data Description

To evaluate the performance of the proposed algorithm, we consider uplink instances generated by using Hata's propagation model. For each instance, we consider a rectangular service area, a number of candidate sites in which to locate omnidirectional antennas, and a number of TPs. Using a pseudorandom number generator each candidate site and each TP is assigned a position with uniform distribution in the service area. We considered two families instances of a urban environment. The simulation parameters are:

- Size of the service area (in km): 0.4×0.4 in the first instance and 1×1 in the second instance;
- Number of TPs: 95 in the first instance and 400 in the second instance;
- Number of BSs: 22 in the first instance and 120 in the second instance;
- $\mathbf{u_i} = 1$, the required number of simultaneously active connections for a TP_i;
- $F = 2000\ MHz$;
- $H_m = 1\ meter$;
- $H_b = 10\ meters$;
- $P_{target} = -100\ dBm$(about $10^{-10}\ MilliWatt$);
- $P_{max} = 30 dBm$;
- $SF = 128$;
- $SIR_{min} = 0.03125 dB$;
- $SIR_{target} = 6\ dB$;
- Costs $\mathbf{c}_i$: are taken randomly between 1 and 20 units.

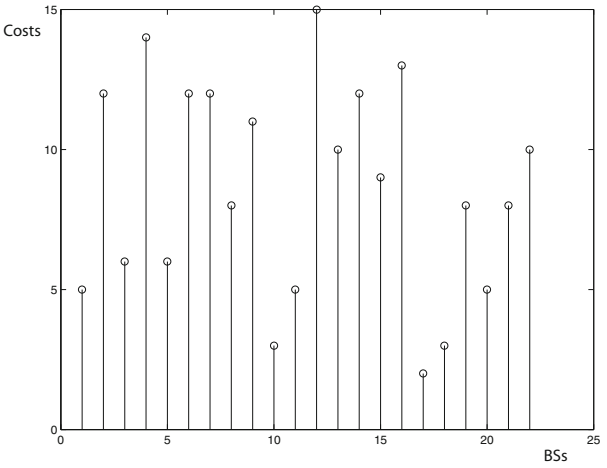


Fig. 5. First instance: Costs of each BS

Figures 4 and 6 illustrate the distribution of the TPs and BSs in the area service of the two instances, respectively. Figures 5 and 7 show the cost of installation of each BS in the two instances, respectively. We note that total cost of installing all BSs is 189 in the first instance problem and 1282 in the second instance.

5.2 Computational Results

For each instance of problem, we consider two cases: (i) λ is taken constant $\lambda = 1$ and $\lambda = 4$ as in [20]; (ii) λ is a dynamic parameter introduced in Section 3.2.

In the GA approaches we have used three selection methods; roulette, scaling and sharing. The parameters of GA are set as follows: crossover probability $p_c = 0.4$, mutation probability $p_m = 0.01$, population size $ps = 30$, and maximum number of generations 5000. In the sharing selection method, the threshold of dissimilarity between two parents is taken as $\sigma_s = ps/2$, and $\alpha = 1$.

Each experiment were conducted on ten times. Tables 1 and 2 show number of TPs covered, number of BSs installed and costs for the two instances problem, respectively. Now, we comment results of each experiment.

First Instance Problem: 95 TPs and 22 BSs.
The good results are obtained by taking the trade-off parameter λ dynamic, see line 6 in Table 1. The best solution consists of installing 20 BSs instead of 22, which cover 93 TPs among 95, with a cost equal to 176. Then we have a gain of 13 (approximately 7% of costs of BSs), since the cost of installing all BSs is 189.

Second Instance Problem: 400 TPs and 95 BSs.
This time also, the good results are obtained by taking the trade-off parameter λ dynamic, see last line in Table 2. The best solution consists of installing 116

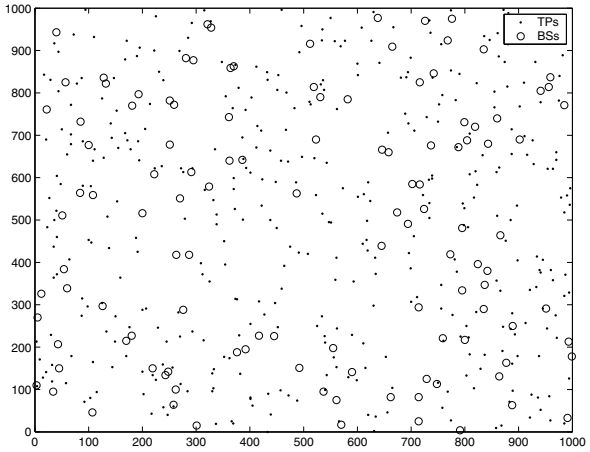


Fig. 6. Second instance: Location of 400 TPs and 120 BSs in a service area of $1 \times 1(Km)$

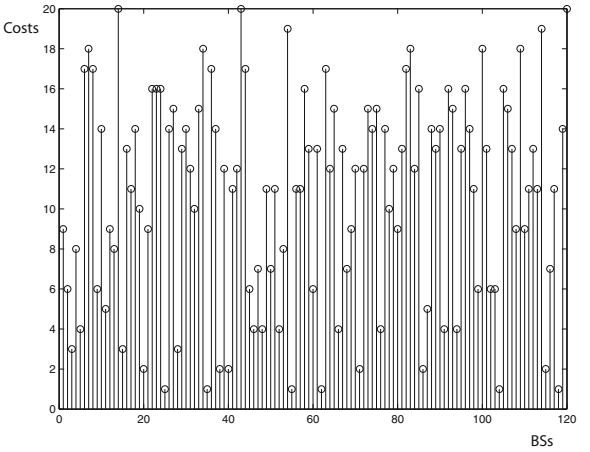


Fig. 7. Second instance: Costs of each BS

Table 1. Number of TPs covered and BSs installed for n = 95 and m = 22

Choice of λ	Selection method	Served TPs	BSs not installed	Cost	Time in S
$\lambda = 1$ (fixed)	Roulette	92	1	187	776
	Scaling	90	1	183	778
	Sharing	93	1	179	777
Dynamic	Roulette	91	1	174	779
	Scaling	**93**	**2**	**176**	780
	Sharing	92	1	184	780

Table 2. Number of TPs covered and BSs installed for n = 400 and m = 120

Choice of λ	Selection method	Served TPs	BSs not installed	Cost	Time in S
$\lambda = 4$ (fixed)	Roulette	398	3	1259	481
	Scaling	397	2	1256	483
	Sharing	398	5	1235	482
Dynamic	Roulette	398	3	1252	484
	Scaling	397	4	1226	486
	Sharing	**398**	**4**	**1231**	485

BSs instead of 120, which cover 398 TPs among 400, with a cost equal to 1231. Then we have a gain of 51, since the cost of installing all BSs is 1282. We realized a gain of approximately 4% of cost of BSs.

6 Conclusion

In this paper we have considered an UMTS base station (BS) location planning problem based on propagation models with power control. The problem, which is formulated as optimizing two objectives functions f and g subjected to some constraints, is transformed into mono-objective problem $f + \lambda g$, where λ is a trade-off parameter, see [1,18]. We note that the choice of λ is not an easy task for both the decision maker and the system analyser. In order to solve the formulated problem we have proposed a GA approach using a dynamic trade-off parameter λ which varies at each iteration of the algorithm. The utility of choosing λ dynamic lies mainly in the two following points:

- It automates the choice of the parameter λ. Therefore we do not need to define this factor a priori.
- It ensures an equitable treatment of each objective function.

To code the solutions of the problem, we have proposed an encoding method which combines binary and integer coding, then we have described the crossover and mutation operators. We have applied our approach to two instances problem. In future research, we will use fuzzy logic to deal with the imprecise and uncertain information of the cost.

References

1. Amaldi, E., Capone, A., Malucelli, F.: Planning UMTS Base Station Location: Optimization Models With Power Control and Algorithms. IEEE Transactions on wireless communications 2, 939–952 (2003)
2. Berruto, E., Gudmundson, M., Menolascino, R., Mohr, W., Pizarroso, M.: Research ctivities on UMTS radio interface, network architectures, and planning. IEEE Communications Magazine 36, 82–95 (1998)

3. Naghshineh, M., Katzela, I.: Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. IEEE Personal Communications 3, 10–31 (1996)
4. St-Hilaire, M., Chamberland, S., Pierre, S.: Uplink UMTS network design-an integrated approach. Computer Networks 50, 2747–2761 (2006)
5. Juttner, A., Orban, A., Fiala, Z.: Two new algorithms for UMTS access network topology design. European Journal of Operational Research 164, 456–474 (2005)
6. Hashemi, S.M., Moradi, A., Rezapour, M.: An ACO algorithm to design UMTS access network using divided and conquer technique. Engineering Applications of Artificial Intelligence 21, 931–940 (2008)
7. Meunier, H.: Algorithmes évolutionnaires parallèles pour l'optimisation multi objectif de réseaux de télécommunications mobiles. PhD thesis, University of Sciences and Technologies, Lille (2002)
8. Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: Métaheuristiques pour l'optimisation difficile. Eyrolles, Paris (2003)
9. Talbi, E.G., Basseur, M., Nebro, A.G., Alba, E.: Multi-objective optimization using metaheuristics: non-standard algorithms. International Transactions in Operational Research 19, 283–306 (2012)
10. Talbi, E.G.: Metaheuristics: From Design to Implementation. John Wiley and Sons (2009)
11. Schaffer, J.D.: Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of an International Conference on Genetic Algorithms and their Applications, pp. 93–100 (1985)
12. Fonseca, C.M., Fleming, P.J.: Multiobjective genetic algorithms. IEE Colloquium on Genetic Algorithms for Control Systems Engineering 6(1-5) (1993)
13. Horn, J., Nafpliotis, N., Goldberg, D.E.: A niched Pareto genetic algorithm for multiobjective optimization. IEEE World Congress on Computational Intelligence, pp. 82–87 (1994)
14. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation, 257–271 (1999)
15. Nakibe, A.: Conception de métaheuristiques d'optimisation pour la segmentation d'images. Application des images biomédicales. PhD thesis, UFR of Sciences and Technology, University PARIS 12-VAL DE MARNE (2007)
16. Fonseca, C.M., Fleming, P.J.: Multiobjective optimization. IOP Publishing, Bristol (2000)
17. Jin, Y., Okabe, T., Sendhoff, B.: Adapting Weighted Aggregation for Multiobjective Evolution Strategies. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 96–110. Springer, Heidelberg (2001)
18. Amaldi, E., Capone, A., Malucelli, F., Signori, F.: Radio Planning and Optimization of W-CDMA Systems. Personal Wireless Communications, 437–447 (2003)
19. Hata, M.: Empirical Formula for Propagation Loss in Land Mobile Radio Services. IEEE Transactions on Vehicular Technology 29, 317–325 (1980)
20. Amaldi, E., Capone, A., Malucelli, F.: Radio planning and coverage optimization of 3G cellular networks. Wireless Networks 14, 435–447 (2008)

Video Encryption Based on the Permutation of the ($\mathbb{Z}/p\mathbb{Z}$) Fields

Younes Benlcouiri[1], Mohammed Benabdellah[2],
Moulay Chrif Ismaili[1], and Abdelmalek Azizi[1]

[1] Laboratory of Arithmetic, Scientific Computing and Applications, Faculty of Science, Mohamed First University, Oujda, Morocco
{benlcouiriy,mcismaili,abdelmalekazizi}@yahoo.fr
[2] Laboratory of Economic and Management of Organizations, Faculty of Law, Economics and Social Sciences. Mohamed First University, Oujda, Morocco
med_benabdellah@yahoo.fr

Abstract. Advances in digital content transmission have increased in recent years. Issues of security and confidentiality of transmitted data have become an important concern in multimedia technology. In this paper, we propose an efficient algorithm to encrypt video sequences in real time without additional dedicated hardware. Our method is based on the principle of modified affine encryption to achieve transposition and to perform some manipulations such as puzzle to reduce the amount of bits to be processed on the image. Testing this approach provides a stronger security and a good encryption time on MPEG sequences.

Keywords: Affine Encryption, MPEG, $\mathbb{Z}/p\mathbb{Z}$ fields, Congruence, Puzzle, Extended Euclidean algorithm.

1 Introduction

During the last few years, interest in multimedia and in particular diffusion of the audio-visual content involved a great amount of research in the field of video signal coding, which led to several standards such as H-263, H.26L and MPEG-4 [8]. These standards consist essentially of toolboxes for video signal processing which can be adapted to the context and desired result. In such applications, confidentiality of the video data during transmission is extremely important. This necessitates secure video encryption algorithms. [4]

In the naive approach for video encryption, the MPEG stream (bit sequence) is treated as text data, and encrypted using standard encryption algorithms like DES (Data Encryption Standard), RC5 (Rivest Cipher), AES (Advanced Encryption Standard), etc. Though this approach is supposedly the most secure for video encryption, it is computationally infeasible for real-time applications. [1]

For real-time applications, light-weight encryption algorithms were also proposed. These methods encrypt using simple XOR or encrypt selected bits of the video data (for example, sign bits of I frames, motion vectors, etc.). These encryption algorithms

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 135–143, 2013.

are much faster than selective algorithms. Also, they add less overhead on the codec. (Note that if encryption modifies the syntax of the MPEG bit stream, it adds overhead to the MPEG codecs.) Another category of algorithms is based on scramble (permutation) only methods, where the DCT coefficients are permuted to provide confusion. However, in most of these methods, computational efficiency comes at the cost of security. [2]

Choon [1] proposed a light-weight and cost effective encryption algorithm based on the Shannon principle of diffusion and confusion. These principles can be achieved by permutation of macro blocks followed by XOR operation on the permuted macro block. Choo [2] proposed another light-weight encryption algorithm on the uncompressed raw MPEG data named Secure Real-time Media Transmission (SRMT), which uses two block transpositions and a XOR operation. Tang [3] proposed a scramble based encryption algorithm using permutation of the DCT coefficients. The basic idea is to use a random permutation list to replace the zig-zag order of the DCT coefficients of a block to a 1×64 vector. Zeng and Lie [10] extended Tang's permutation range from a block to a segment, where each segment consists of several macro blocks. Within each segment, the DCT coefficients of the same frequency band are randomly shuffled within the same band. Apart from shuffling of the I frames, they also permute the motion vectors of P and B frames. However, light-weight encryption and scramble-only methods provide less security than the naive encryption.

In this sense, and in order to optimize and secure transposition and storage of movie, we propose a new transposition approach to implement the principles of using affine encryption. The main advantages of this approach are flexibility and reduction of processing time, which is proportional to the number of pixels during the operation of encryption and decryption. Indeed, through this method we can vary the processing time depending on the desired level of security.

In a first step, we will discuss the affine encryption procedure and some notions of congruence. Then, we describe the subdivision process image in puzzle. After, we will describe in detail the principles of our approach and results obtained after its implementation. Finally, we conclude our article by introducing some perspectives.

2 Methods

2.1 Structure of Coding MPEG Format

The techniques used to compress a video signal use space redundancy. The objective is to reduce the flow of the video sequence to be compressed, while minimizing the visible errors (MSE and PSNR) [5]. To do this, there are two principal techniques, lossless compression and lossy compression. The former makes it possible to find the initial information after decompression, while the latter will restore only an approximation of it. In the case of natural images, the lossless compression is insufficient, and the introduction of losses in the compression process makes it possible to obtain better results without preventing the interpretation of the visual content. Current video standards use a hybrid coding system with compensation for movement based on blocks and a reduction of entropy by a transformer. The MPEG standard defines a set of coding stages that transform a video signal (digitized in standardized format) into a

binary stream (a bit stream) intended to be stored or transmitted through a network. The binary stream is described according to a syntax coded in a standardized way that can be restored easily by any decoder that recognizes the MPEG standard. The coding algorithm defines a hierarchical structure containing the levels described in the following figure 1. [6]

Sequence	Heading	GOP	.....	GOP
GOP	Heading	Image	.....	Image
Image	Heading	Slice	.....	Slice
Slice	Heading	Macro-Block	.....	Macro-Blok
Macro-Block	Heading	Block	.....	Block
Block	Heading	Coeficient	.....	Coeficient

Fig. 1. Hierarchical structure of MPEG coding

The group of pictures or GOP consists of a periodic continuation of compressed images. There are three types of the compressed images: Intra image (I) compressed using JPEG for the fixed images, Predicted image (P) coded using a prediction of a previous image of type I or P, and Bidirectional image (B) coded by double prediction (or Interpolation) by using a previous image of type I or P and a future Intra image or Predicted image as references. A GOP starts with an image I, contains a periodic continuation of the images P separated by a constant number of images B as in the following figure 2. The structure of GOP is thus defined by two parameters: the number of images of GOP and the distance between Intra images and Predicted image [7].

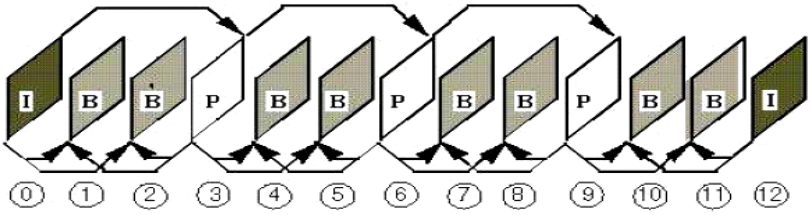


Fig. 2. Structure of GOP

2.2 Cryptography

The most methods of encryption based on two principles: substitution and transposition. Substitution means that replacing some letters by symbols or others. Transposition means that permuting the letters of the message to make it unintelligible. Over the centuries, many systems of cryptographic Telecare Medicine Information System (TMIS) have developed more perfection more clever. [9]

2.3 Congruence

For $\boldsymbol{u}, \boldsymbol{v}$ in $\mathbb{Z}$ and n an integer ≥ 2, the notation $\boldsymbol{u} \equiv \boldsymbol{v}\ \boldsymbol{mod}(\boldsymbol{n})$ reads u congruent to v modulo n and means that $(\boldsymbol{u} - \boldsymbol{v})$ is divisible by n which is equivalent to say that u and v have the same retained when divided by n, for example 17 = 5 (3) but also -1 = 1 (2). [9]

u, v, r, s belong to $\mathbb{Z}$ and n an integer ≥ 2,
if u ≡ v (n) and r ≡ s (n) so:
u+r ≡ v+s (n); u-r ≡ v-s (n); u×r ≡ v×s (n);
and for all k ∈ $\mathbb{Z}$ we have u ≡ v+kn (n)
if u and v are two integers belonging to {0; 1; 2;… n-1}
so u ≡ v(n) implies u ≡ v

2.4 The Extended Euclidean Algorithm

The extended Euclidean algorithm permits to calculate the inverse of b modulo n if it exists. Remembering that the inverse of modulo n of b is the whole number b^{-1} such that $b \times b^{-1} = 1 \pmod n$ for example 7 is the inverse modulo 9 of 4 because $4 \times 7 = 28 = 1 \pmod 9$

Algorithm.

n_0 := n
b_0 := b
t_0 := 0
t := 1
q := an integer less than or equal to n_0/b_0
r := $n_0 - q \times b_0$
while r > 0 do
start
temp := $t_0 - q \times t$
if temp >= 0 then
temp := temp mod n,
else temp := n - ((-temp) mod n)
t_0 := t
t := temp
n_0 := b_0
b_0 := r
q := an integer less or equal to n_0/b_0
r := $n_0 - q \times b_0$
end while ;
if $b_0 \neq 1$ then b has no inverse modulo n,
else b^{-1} mod n =t

This algorithm can also calculate the Bezout coefficients of a and b (called extended Euclidean algorithm). Recalling that if d is the Greatest Common Divisor (GCD) of a and b, there exists the whole u and v such that $au + bv = d$. The Euclidean algorithm

allows calculating these u and v coefficients. Simply go up the calculations by expressing the GCD d as a function of other numbers. [9]

2.5 Affine Encryption

Noting that E = {0; 1; 2;…;25} a and b are integers selected from E. Coding is affine, after numbered from 0 to 25 letters of the alphabet, to encode a letter (called source) number x by the letter number y, where y is the remainder of the division of $ax + b$ by 26. The encoding function associated affine associated with the coefficients a and b is the function f from E to E in which x matches to $f(x) = y$. $f(x)$ is the only element of the set E = {0, 1, 2... 25} which is congruent to $ax + b$ modulo 26, $f(x) \equiv ax + b\ mod(26)$.[9]

3 Proposed Method

Our method is based on encrypting only Intra pictures in each GOP of MPEG sequence while keeping the motion vectors responsible for reconstruction of the P images and the B images.

It is to use the principles of an affine encryption on field type ($\mathbb{Z}/p\mathbb{Z}$) married to the results of modular arithmetic and the problem of the puzzle to complete the transposition on these elements.

$$\begin{array}{lll} f:(\mathbb{Z}/p\mathbb{Z})^* & \rightarrow & (\mathbb{Z}/p\mathbb{Z})^* \\ : \quad x & \rightarrow & \begin{cases} b\ if\ x = a^{-1}(-b)\ mod(p) \\ else \qquad ax + b\ mod(p) \end{cases} \end{array} \tag{1}$$

Note that: $f_{a1,b1} \circ f_{a2,b2} \neq f_{a2*a1,a2*b1+b2}$ when $a_2 b_1 \neq -b_2$.

For our method, the keys generation is divided into five parts:

- Choose a prime number p to work on the ($\mathbb{Z}/p\mathbb{Z}$)field, in which all elements are invertible.
- Subdivide the image processing into $(p-1)$ puzzle pieces (square, triangle…) with $(n \times m)$ blocks size according to the required security level.
- Let M a k pair of elements (a_i, b_i)in ($(\mathbb{Z}/p\mathbb{Z})^*$, $(\mathbb{Z}/p\mathbb{Z})$) such as :

$$\begin{cases} a_1 \neq a_{\mathrm{k}}^{-1} \text{and } a_i \neq a_{\mathrm{i+1}}^{-1} \text{ for each element of } i = 1,2 \ldots k-1 \\ -b_{i-1} \neq a_{\mathrm{i}}^{-1} b_i \text{ for } 1 < i \leq k \text{ and } -b_k \neq a_1^{-1} b_1 \end{cases}$$

  then $M = \{(a_1, b_1), (a_2, b_2), \ldots (a_k, b_k)\}$.
- Calculate the number of cycles N_c of M on the function f defined in (1).
- Choose a number N_b less than the number of cycles N_c ($N_b < N_c$).

The encryption key is:$\{p, M, N_b, (n, m)\}$

The decryption key is the same as the encryption key.

Encryption Step: We divide each image Intra of each GOP of the MPEG video sequence in $(\boldsymbol{p-1})$ macro-blocks as shown in the following figure:

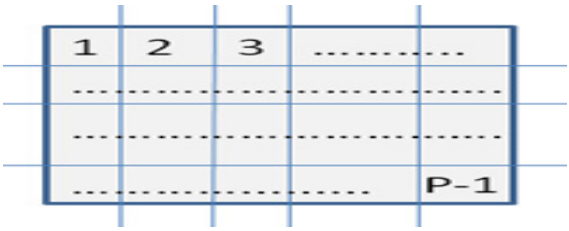


Fig. 3. Original Intra image divided into (p-1) macro-blocks

After subdividing the Intra image in $(\boldsymbol{p-1})$ macro-blocks numbered from 1 $(\boldsymbol{p-1})$, we proceed as following:

We repeat the application of the function f for each pair $(\boldsymbol{a_i, b_i})$ of M on the results obtained after application of this function with the parameters $(\boldsymbol{a_{i-1}, b_{i-1}})$when $\boldsymbol{i} > 1$, then, repeat the process $\boldsymbol{N_b}$ times for multiplying the number of lost blocks.

The encryption function can be defined as follows:

$$g_{(M,N_b)} = f_M^{N_b} = \left(f_{(a1,b1)} \circ f_{(a2,b2)} \dots \circ f_{(ak,bk)}\right)^{N_b} \tag{2}$$

To change the location of each macro-block of image processing, we apply the function $\boldsymbol{g}$ to its indices.

$$\{\boldsymbol{g_{(M,N_b)}(1)};\ \boldsymbol{g_{(M,N_b)}(2)}; \dots . \boldsymbol{g_{(M,N_b)}(p-1)}\} = \{\mathbf{5}; \mathbf{10}; \dots; \boldsymbol{(p-1)}; \dots \mathbf{3}\}_{\boldsymbol{transpose}}$$

Reorganize the results after applying the function $\boldsymbol{f}$ on the indices of macro-blocks conducted a transposition. The encrypted image is shown in the figure below:

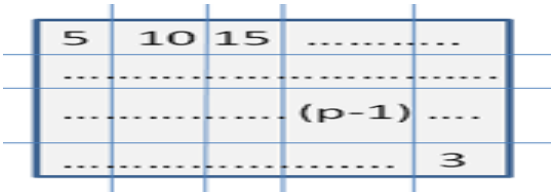


Fig. 4. Encrypted image after transposition of (p-1) macro-blocks

Reconstruct the video sequence by replacing the original Intra image by the Intra image after processing.

Decryption Step: The decryption key is the $\{\boldsymbol{p, M, N_b, (n, m)}\}$.We performs the extraction of intra images of the video sequence encrypted then remake the same calculation performed in the encryption phase to find the order of transposition:

$$\{\mathbf{5}; \mathbf{10}; \dots; (\boldsymbol{p}-\mathbf{1}); \dots \mathbf{3}\}_{\boldsymbol{transpose}}$$

We rearrange Intra encrypted image according to the result obtained. Then we apply the MPEG algorithm for the decoded video sequence.

4 Application and Results

We apply the proposed method to news sequence. We proceed as follows:

1) We choose a prime number $p = 257$ to work on the fields$\mathbb{Z}/p\mathbb{Z} = \mathbb{Z}/257\mathbb{Z}$.
2) Then we divide the image into $(p-1) = 256$ elements each of size $(n \times m)$ where $n = 8$ n = 8 and $m = 8$.
3) We selected M from $\mathbb{Z}/257\mathbb{Z}$ composed with three pair $\{(a_1, b_1), (a_2, b_2), (a_k, b_k)\} = \{(19,6), (1,1), (21,90))\}$, when $a_2^{-1} \times b_2 \neq -b_1, a_3^{-1} \times b_3 \neq -b_2$ and $a_1^{-1} \times b_1 \neq -b_3$.
4) Calculate the number of cycles N_c=105819 of M on the function$f$.
5) We choose $N_b = 24$ an integer $< N_c$

We use the element pairs in M successively by applying the function f N_b times to provide the greatest disorder possible.

The decryption key is the same as the encryption key and it is given as follows:
$\{p = 257; M = \{(19,6), (1,1), (21,90))\}; N_b = 24; (n \times m) = (8 \times 8)\}$

Table 1. Results after applying our method on Intra image. E.O.I: Entropy of Original Image, E.E.I: Entropy of Encrypted Image, E.T: Encryption Time (ms).

GOP(1:12)	Our method		
	E.O.I	E.E.I	E.T(ms)
Intra image	7.0005	7.0005	5.2
Predicted image (P1)	6.9285	7.1651	0
Bidirectional image (B1)	6.9199	7.2486	0
Bidirectional image (B2)	6.9263	7.2489	0

The subdivision of the image into 256 macro-blocks allows us, on encryption, to make a calculation of codes into 8 bits and obtain a minimum processing time. The entropy of the original Intra image is equal to that of the encrypted Intra image which provides a high degree of security according to law of security measure that is introduced by Shannon in information theory.

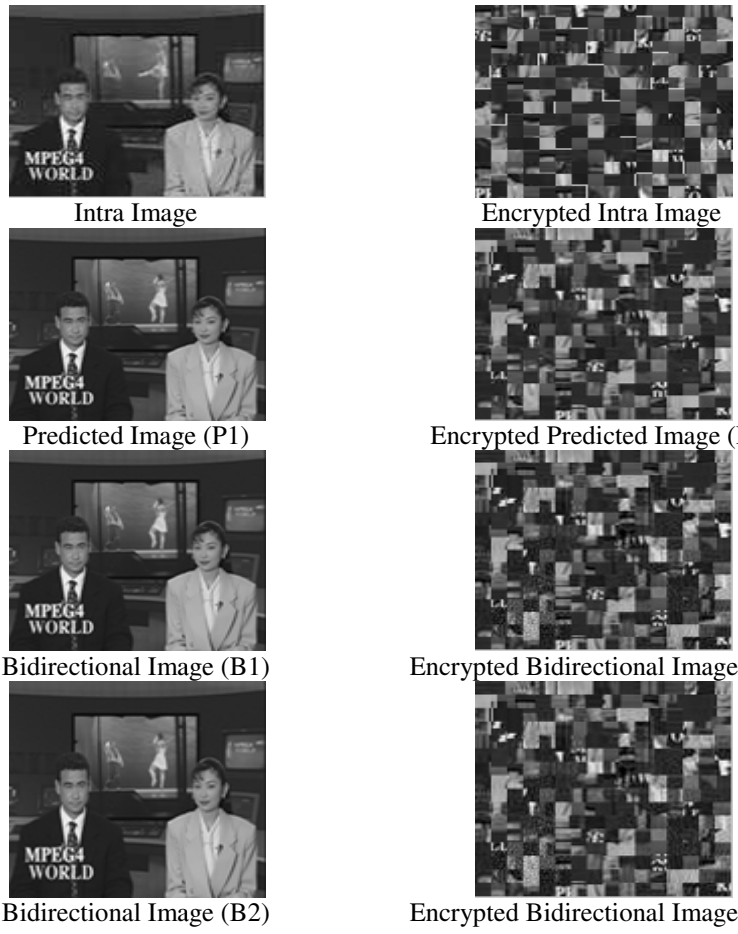


Fig. 5. Encryption intra image and its effect on P and B images of news sequence(GOP(1: 12))

The attack of our method with $p \times (p-1)$ possible keys does not lead to the reconstruction of the original image since the $f_{(a1,b1)} \circ f_{(a2,b2)}$ which is different from the $f_{(a2*a1),(a2*b1+b2)}$ lost at least one macro block. Knowing that the combination used for the transposition is outside of $p \times (p-1)$ cases directly accessible by all the keys.

So the problem is more complicated for brute force cryptanalysis since it must grope on the combination of M and the number of repetitions. We have a complex $p \times (p-1)^{Nc}$check images. Thus, an attacker who uses brute force will be forced to tackle the puzzle.

5 Conclusion

In this paper, we introduced a novel approach to encrypt video streams in real-time by using the affine cipher method and puzzle principle. This "hybrid" method is based on

a modified Affine Cipher to realize permutations on $\mathbb{Z}/p\mathbb{Z}$ fields and on puzzle techniques to reduce the number of bits to be processed. The basic idea is to encrypt only I-frames in each GOP of an MPEG sequence.

The advantage of our method is to increase a number of accessible cases. Indeed, the Affine Cipher allowed reaching only $p \times (p-1)$cases among $p \times (p-1)!$. However, by using the method presented in this work, one can access to a number of cases which is bigger than $p \times (p-1)$. By working in $\mathbb{Z}/p\mathbb{Z}$ field we can choose randomly keys.

Based on its encryption speed and degree of security, we think make hybridizations between this encryption method and other compression methods to produce crypto-compression methods of still and moving images. We are currently working on producing a random crypto-system based on affine encryption for future use on IPTV diffusion protocol.

Acknowledgements. Supported by the Hassan II Academy of science and technology of Morocco.

References

1. Choon, L.S.: Lightweight and cost-effective MPEG video encryption. In: International Conference on Information and Communication Technologies: From Theory to Application, pp. 525–526 (2004)
2. Choo, E., Lee, J., Lee, H., Nam, G.: SRMT: A Lightweight Encryption Scheme for Secure Real-time Multimedia Transmission. In: International Conference on Multimedia and Ubiquitous Engineering (MUE 2007) (2007)
3. Tang, L.: Methods for encrypting and decrypting MPEG video data efficiently. In: ACM International Multimedia Conference, Boston, MA (1996)
4. Benabdellah, M., Gharbi, M., Regragui, F., Bouyakhf, E.H.: A method for choosing reference images based on edge detection for video compression. In: Second IEEE-EURASIP International Symposium on Control, Communications, and Signal Processing (IEEE-ISCCSP), Marrakech-Morocco (2006)
5. Benabdellah, M., Gharbi, M., Regragui, F., Bouyakhf, E.H.: A method for choosing reference images in video compression. In: Fifth IEEE-EURASIP International Symposium on Communication Systems, Networks and Digital Signal Processing (IEEE-CSNDSP 2006), Patras-Greece (2006)
6. Benabdellah, M., Gharbi, M., Regragui, F., Bouyakhf, E.H.: An approach for choosing the reference images based on FMT in video compression process. In: 1st International Conference on Digital Communications and Computer Applications (DCCA), Irbid, Jordan (2007)
7. Benabdellah, M., Gharbi, M., Regragui, F., Bouyakhf, E.H.: Choice of reference images for video compression. International Journal of Applied Mathematical Sciences 1(44), 2187–2201 (2007)
8. Benabdellah, M., Benabdellah, Z., Essofi, S.: Compression des scènes vidéosurveillance basée sur la détection des contours avec analyse juridique. In: Colloque TELECOM 2009 & 6èmes JFMMA, Agadir- Maroc (2009)
9. Benlcouiri, Y., Benabdellah, M., Ismaili, M.C., Azizi, A.: Encryption of ultrasound images using the permutation on the (Z/pZ) fields 7(42), 2093–2102 (2013)
10. Zeng, W., Lei, S.: efficient frequency domain selective scrambling of digital video. IEEE Transaction on Multimedia, 118–219 (2002)

Improving Resource Location with Locally Precomputed Partial Random Walks*

Víctor M. López Millán[1], Vicent Cholvi[2],
Luis López[3], and Antonio Fernández Anta[4]

[1] Universidad CEU San Pablo, Spain
vmlopez.eps@ceu.es
[2] Universitat Jaume I, Spain
vcholvi@uji.es
[3] Universidad Rey Juan Carlos, Spain
llopez@gsyc.es
[4] Institute IMDEA Networks, Spain
antonio.fernandez@imdea.org

Abstract. Random walks can be used to search complex networks for a desired resource. To reduce search lengths, we propose a mechanism based on building random walks connecting together partial walks (PW) previously computed at each network node. Resources found in each PW are registered. Searches can then jump over PWs where the resource is not located. However, we assume that perfect recording of resources may be costly, and hence, probabilistic structures like Bloom filters are used. Then, unnecessary hops may come from false positives at the Bloom filters. Two variations of this mechanism have been considered, depending on whether we first choose a PW in the current node and then check it for the resource, or we first check all PWs and then choose one. In addition, PWs can be either simple random walks or self-avoiding random walks. Analytical models are provided to predict expected search lengths and other magnitudes of the resulting four mechanisms. Simulation experiments validate these predictions and allow us to compare these techniques with simple random walk searches, finding very large reductions of expected search lengths.

Keywords: Random walks, self-avoiding random walks, network search, resource location, search length.

1 Introduction

A *random walk* in a network is a routing mechanism that chooses the next node to visit at random among the neighbors of the current node. Random walks have been extensively studied in mathematics, and have been used in a wide range of

* This research was supported in part by Comunidad de Madrid grant S2009TIC-1692, Spanish MICINN grant TEC2011-29688-C02-01, Spanish MEC grant TIN2011-28347-C02-01 and Bancaixa grant P11B2010-28.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 144–158, 2013.

applications such as statistic physics, population dynamics, bioinformatics, etc. When applied to communication networks, random walks have had a profound impact on algorithms and complexity theory. Some of the advantages of random walks are their simplicity, their small processing power consumption at the nodes, and the fact that they need only local information, avoiding the communication overhead necessary in other routing mechanisms. An important application of random walks has been the search for resources held in the nodes of a network, also known as the *resource location problem*. Roughly speaking, the problem consists of finding a node that holds the resource, starting at some *source node*. Random walks can be used to perform such a search as follows. It is checked first if the source node holds the resource. If it does not, the search hops to a random neighbor, that repeats the process. The search proceeds through the network in this way until a node that holds the resource is found. Due to the random nature of the walk, some nodes may be visited more than once (unnecessarily from the search standpoint), while other nodes may remain unvisited for a long time. The number of hops taken to find the resource is called the *search length* of that walk. The performance of this direct application of random walks to network search has been studied in [1,2,3,4,5].

The use of random walks for resource location has several clear applications, like unstructured peer-to-peer (P2P) file sharing systems or content-centric networks (CCN) [6]. The latter are networks in which the key elements are named content chunks, which are requested by users using the content name. Content chunks have to be efficiently located and transferred to be consumed by the user. The techniques described in this paper could be used in the context of CCN to locate content chunks.

Contributions. This paper proposes an application to resource location of the technique of concatenating partial walks (PW) available at each node to build random walks. A PW is a precomputed random walk of fixed length. Two variations are considered, depending on whether the search mechanism first randomly chooses one of the PWs in the current node and then checks its associated information for the desired resource, or it first checks all PWs in the node and then randomly chooses among those with a positive result. Both of these variations may use PWs that are simple random walks (RW) or self-avoiding random-walks (SAW), resulting in four mechanisms referred to as *choose-first* PW-RW or PW-SAW, and *check-first* PW-RW or PW-SAW, respectively. Our mechanisms assume the use of Bloom filters [7] to efficiently store the set of resources (not their owners) held by the nodes in each partial walk. The compactness of Bloom filters comes at the price of possible *false positives* when checking if a given resource is in the partial walk. False positives occur with a probability p, which is taken into account in our analyses. These assumptions provide generality to our model, since a probability of $p = 0$ models the case in which the full list of resources found are stored (instead of using a Bloom filter).

We provide an analytical model for the choose-first PW-RW technique, with expressions for the *expected search length*, the *optimal length of the partial walks*, and for the *optimal expected search length*. We found that, when the probability

of false positives in Bloom filters is small, the optimal expected search length is proportional to the square root of the expected search length achieved by simple random walks, in agreement with the results in [8]. Another interesting finding is that the optimal length of the partial walks does not depend on the probability of false positives of the Bloom filters. We also provide analytical models for the choose-first PW-SAW mechanism as well as for the check-first variations, which predict their expected search length. Then, the predictions of the models are validated by simulation experiments in three types of randomly built networks: regular, Erdős-Rényi, and scale-free. These experiments are also used to compare the performance of the four mechanisms, and to investigate the influence of parameters as the false positive probability and the number of partial walks per node. Finally, we have compared the performance of the four search mechanisms with respect to simple random walk searches. For choose-first PW-RW we have found a reduction in the average search length ranging from around 98% to 88%. For choose-first PW-SAW such a reduction is even bigger, ranging from 12% to 5% with respect to PW-RW. Check-first PW-RW and PW-SAW can achieve still larger reductions increasing the number of PWs available at each node.

Related Work. Das Sarma et al. [8] proposed a distributed algorithm to obtain a random walk of a specified length ℓ in a number of rounds[1] proportional to $\sqrt{\ell}$. In the first phase, every node in the network prepares a number of *short (random) walks* departing from itself. The second phase takes place when a random walk of a given length starting from a given source node is requested. One of the short walks of the source node is randomly chosen to be the first part of the requested random walk. Then, the last node of that short walk is processed. One of its short walks is randomly chosen, and it is *connected* to the previous short walk. The process continues until the desired length is reached.

Hieungmany and Shioda [9] proposed a *random-walk-based* file search for P2P networks. A search is conducted along the concatenation of hop-limited shortest path trees. To find a file, a node first checks its *file list* (i.e., an index of files owned by neighbor nodes). If the requested file is found in the list, the node sends the file request message to the file owner. Otherwise, it randomly selects a leaf node of the hop-limited shortest path tree, and the search follows that path, checking the *file list* of each node in it.

The use of partial random walks in resource location has been proposed in [10] for networks with dynamic resoures. Our work in this paper incorporates efficient storage by means of Bloom filters, in the context of static resources. The use of SAWs as PWs is also proposed and compared with simple RWs.

Structure. The next section presents a model for the four search mechanisms proposed. Then, the choose-first PW-RW is evaluated in Section 3. For the

[1] A *round* is a unit of discrete time in which every node is allowed to send a message to one of its neighbors. According to this definition, a simple random walk of length ℓ would then take ℓ rounds to be computed.

sake of clarity, the choose-first PW-SAW mechanism is covered separately in Section 4, which includes the corresponding analysis together with performance results. Similarly, the check-first PW-RW/PW-SAW mechanisms are presented in Section 5.

2 Model

Let us consider a randomly built network of N nodes and arbitrary topology, whose nodes hold resources randomly placed in them. Resources are unique, i.e., there is a single instance of each resource in the network. The resource location problem is defined as visiting the node that holds the resource, starting from a certain node (the *source* node). For each search, the source node is chosen uniformly at random among all nodes in the network.

The search mechanisms proposed in this paper exploit the idea of efficiently building *total random walks* from *partial random walks* available at each node of the network. This process comprises two stages:

(1) Partial Walks Construction. Every node i in the network precomputes a set W_i of w random walks in an initial stage before the searches take place. Each of these partial walks has length s, starting at i and finishing at a node reached after s hops. In the PW-RW mechanism, the partial walks computed in this stage are simple random walks. During the computation of each partial walk in W_i, node i registers the resources held by the s first nodes in the partial walk (from i to the one before the last node). As mentioned, for generality, we assume that the resources found are stored in a Bloom filter. This information will be used in Stage 2. Bloom filters are space-efficient randomized data structures to store sets, supporting membership queries. Thus, the Bloom filter of a partial walk can be queried for a given resource. If the result is negative, the resource is not in any of the nodes of the partial walk. If the result is positive, the resource is in one of the nodes of the partial walk, unless the result was a *false positive*, which occurs with a certain probability p.[2] The size of the Bloom filters can be designed for a target (small) p considered appropriate. A variation of the partial walk construction mechanism consists of using PWs that are *self-avoiding* walks (SAW). The resulting mechanism, called PW-SAW, is analyzed in Section 4.

(2) The Searches. After the PWs are constructed, searches are performed in the following fashion when the choose-first PW-RW/PW-SAW mechanisms are used. When a search starts at a node A, a PW in W_A is chosen uniformly at random. Its Bloom filter is then queried for the desired resource. If the result is negative, the search *jumps* to node B, the last node of that partial walk. The process is then repeated at B, so that the search keeps jumping in this way while the results of the queries are negative. When at a node C, the query to the Bloom filter (of the PW randomly chosen from W_C) gives a positive result,

[2] More concretely, p is the probability of obtaining a positive result conditioned on the desired resource not being in the filter.

the search *traverses* that partial walk looking for the resource until the resource is found or the partial walk is finished. If the resource is found, the search stops. If the search reaches the last node D of the partial walk without having found the resource in the previous nodes, it means that the result of the Bloom filter query was a false positive. The search then randomly chooses a partial walk in W_D and decides whether to jump over it or to traverse it depending on the result of the query to its Bloom filter, as described above. A variation of this behavior consists of first checking all PWs of the node for the desired resource, and then randomly choosing among the ones with a positive result. The resulting mechanisms, called check-first PW-RW/PW-SAW are analyzed in Section 5.

In this work, we are interested in the number of *hops* to find a resource (when PWs of length s are used), which is defined as the *search length* and denoted L_s. Some of these hops are *jumps* (over PWs) and other are *steps* (traversing PWs). In turn, we distinguish between *trailing steps*, if they are the ones taken when the resource is found, and *unnecessary steps*, if they are taken when the resource is not found. The search length is a random variable that takes different values when independent searches are performed. The *search length distribution* is defined as the probability distribution of the search length random variable. We are interested in finding the *expected search length*, denoted $\overline{L}_s$. Figure 1 summarizes the behavior of the search mechanisms.

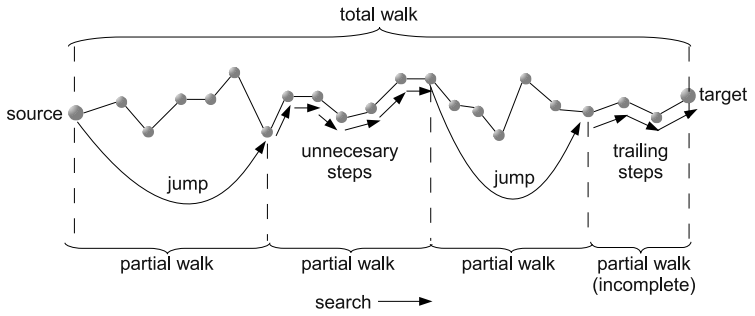


Fig. 1. An example of search, using PWs of length $s = 6$

At this point, we emphasize the difference between the *search* just defined and the *total walk* that supports it, consisting of the concatenation of *partial walks* as defined above. Searches are shorter in length than their corresponding total walks because of the number of steps saved in jumps over partial walks in which we know that the resource is not located (although these saving may be reduced by the unnecessary steps due to Bloom filter false positives).

3 Choose-First PW-RW

3.1 Analysis of Choose-First PW-RW

We make an additional assumption in order to simplify this analysis. Once a PW has been used in the total walk of a search, it is never reused again in that

total walk or in any other searches. Thus we guarantee that the total walks are true random walks. This implies that in practice each node needs to have a large number of precomputed partial walks (w), assumption that would compromise the benefits of the proposed mechanism in practice. Simulations in Section 3.3 show that real cases with small w behave very similarly to the base case provided by this analysis.

Let L_s be the random variable representing the number of hops in the search (i.e., its length) when PWs of length s are used. The expected search length is denoted by $\overline{L}_s$. Let L be the random variable representing the number of hops of the corresponding total walk. Its expected search length is denoted $\overline{L}$. Making use of the assumption that partial walks are never reused, L can be viewed as the length of a search based on a simple random walk in the considered network, and $\overline{L}$ as the expected search length of random walks in that network. Then, we can state the following theorem:

Theorem 1. *If the expected number of trailing steps is assumed to be uniformly distributed in* $[0, s-1]$[3]*, then the expected search length is:*

$$\overline{L}_s = \left(\frac{s}{2} + \frac{2\overline{L}+1}{2s} - 1\right) \cdot (1-p) + \overline{L} \cdot p. \tag{1}$$

Proof. Let P, J, U and T be random variables representing the number of partial walks, jumps, unnecessary steps and trailing steps in a search, respectively. Their expectations are denoted as $\overline{P}$, $\overline{J}$, $\overline{U}$ and $\overline{T}$. Since hops in a search can be jumps, unnecessary steps or trailing steps, it follows that, $L_s = J + U + T$. Then, the expected search length for partial walks of size s is[4] $\overline{L}_s = \overline{J} + \overline{U} + \overline{T}$.

The expected number of jumps can be obtained from the expected number of partial walks in the search ($\overline{P}$) and from the probability of false positive (p) as $\overline{J} = \overline{P} \cdot (1-p)$, since J follows a binomial distribution $B(P, 1-p)$, where the number of experiments is the random variable representing the number of partial walks in a search (P) and the success probability is the probability of obtaining a negative result in a Bloom filter query $(1-p)$.[5]

For the expected number of unnecessary steps, $\overline{U} = \overline{P} \cdot p \cdot s$, since $\overline{P} \cdot p$ is the expected number of false positives in the search and each of them contributes with s unnecesary steps. The number of partial walks in a search can be obtained dividing the length of the total walk by the size of a partial walk: $P = \left\lfloor \frac{L}{s} \right\rfloor = \frac{L-T}{s}$. Then, the expected number of partial walks in a search is $\overline{P} = \frac{\overline{L}-\overline{T}}{s}$.

[3] This is, in fact, a pessimistic assumption. The distribution of trailing steps is approximately uniform, but shorter walks have a slightly higher probability than longer ones. This can be shown analytically and has been confirmed in our experiments (see Appendix A in [11]). Therefore, the expected value in our analysis, derived from a perfectly uniform distribution, is slightly higher than the real average value.

[4] In the following, we make implicit use of the linearity properties of expectations of random variables.

[5] If Y is a random variable with a binomial distribution with success probability p, in which the number of experiments is in turn the random variable X, it can be easily shown that $\overline{Y} = \overline{X} \cdot p$ (see Appendix B in [11]).

Since we assume that the expected number of trailing steps is uniformly distributed between 0 and $(s-1)$, its expectation is $\overline{T} = \frac{s-1}{2}$.

Using the previous equations we have:

$$\overline{L}_s = \left(\frac{s}{2} + \frac{2\overline{L}+1}{2s} - 1\right) + p \cdot \left(\overline{L} - \left(\frac{s}{2} + \frac{2\overline{L}+1}{2s} - 1\right)\right), \tag{2}$$

where the first term is the expectation of the search length for a "perfect" Bloom filter (one that never returns a false positive) and the second term is the expectation of the additional search length due to false positives.

Another interpretation of this expression is obtained if we reorganize it to make explicit the contributions of a perfect filter and of a "broken" filter (one that always returns a false positive result when the resource is not in the filter, i.e., $p = 1$) as

$$\overline{L}_s = \left(\frac{s}{2} + \frac{2\overline{L}+1}{2s} - 1\right) \cdot (1-p) + \overline{L} \cdot p. \tag{3}$$

From this theorem and using calculus, we have the following corollary.

Corollary 1. *The optimal length of the partial walks, i.e., the length of the partial walks that minimizes the expected search length, is:*

$$s_{opt} = \sqrt{2\overline{L}+1}. \tag{4}$$

The obtained value needs to be rounded to an integer, which is omitted in the notation. Observe that *the optimal length of the partial walks is independent from the probability of false positives in the Bloom filters*, while the expected search length ($\overline{L}_s$) does of course depend on it.

Corollary 2. *The optimal expected search length, i.e., the expected search length when partial walks of optimal length are used, is:*

$$\overline{L}_{opt} = \left(\sqrt{2\overline{L}+1} - 1\right)(1-p) + \overline{L}\,p = (s_{opt}-1)(1-p) + \overline{L}\,p. \tag{5}$$

This result is an interesting relation between the optimal length of the search and the optimal length of the PWs. If we consider perfect Bloom filters ($p = 0$), we have $\overline{L}_{opt} = s_{opt} - 1$, which for large $\overline{L}$ (e.g. for large networks) becomes $\overline{L}_{opt} \approx s_{opt}$. Therefore, we have found that, for large N and $p = 0$, *the optimal expected search length approximately equals the optimal length of the partial walks.* For arbitrary values of p, Equation 5 shows that $\overline{L}_{opt}$ is linear in p.

This completes the analysis of choose-first PW-RW. Appendix D in [11] provides an alternative analysis using a different approach. Instead of assuming that the total walk is a random walk, it considers that it is built using the w PWs available at each node, which avoids the need of $\overline{L}$. On the other hand, the alternative model does not provide expressions for $\overline{L}_{opt}$ or s_{opt}.

3.2 Cost of Precomputing PWs

Since searches use the partial walks precomputed by each of the nodes of the network, the cost of this computation must be taken into account. We measure this cost as the number of messages C_p that need to be sent to compute all the PWs in the network. This quantity has been chosen to be consistent with our measure of the performance of the searches. Indeed, each *hop* taken by a search can be alternatively considered as a *message* sent. In addition, C_p is independent from other factors like the processing power of nodes, the bandwidth of links and the load of the network. The cost of precomputing a set of PWs can be simply obtained as $C_p = Nw(s+1)$, since each of the N nodes in the network computes w partial walks, sending s messages to build each of them plus one extra message to get back to its source node.

Let's suppose that each node starts on the average b searches that are processed by the network with the set of PWs precomputed initially. We define C_s to be the total number of messages needed to complete those searches. If the expected number of messages of a search is $\overline{L}_s + 1$ (counting the message to get back to the source node), we have that $C_s = Nb(\overline{L}_s + 1)$. Now, defining C_t as the *average total cost per search*, we can write:

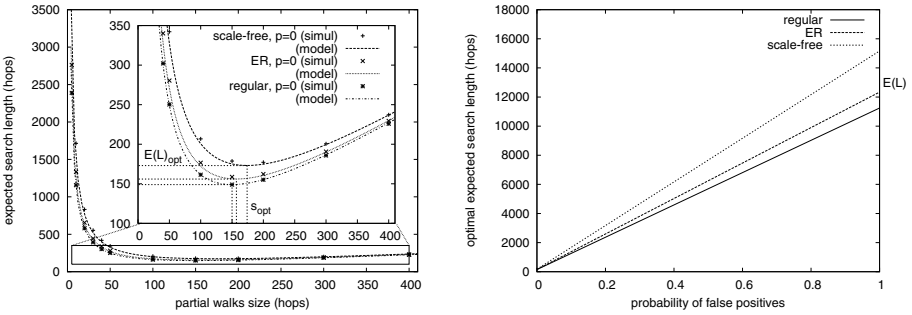
$$C_t = \frac{C_s + C_p}{Nb} = (\overline{L}_s + 1) + \frac{w}{b}(s + 1). \tag{6}$$

The second term in Equation 6 is the contribution to the cost of the precomputation of the PWs. This contribution will remain small provided that the number of searches per node in the interval is large enough.

3.3 Performance Evaluation

The goal of this section is to apply the model for choose-first PW-RW presented in the previous section to real networks, and to validate its predictions with data obtained from simulations. Three types of networks have been chosen for the experiments: regular networks (constant node degree), Erdős-Rényi (ER) networks and scale-free networks (with power law on the node degree). A network of each type and size $N = 10^4$ has been randomly built with the method proposed by Newman et al. [12] for networks with arbitrary degree distribution, setting their average node degree to $\overline{k} = 10$. Each network is constructed in three steps: (1) a preliminary network is constructed according to its type; (2) its degree distribution is extracted, and (3) the final (random) network is obtained feeding the Newman method with that degree distribution. For each experiment, 10^6 searches have been performed, with the source node chosen uniformly at random among the N nodes. Likewise, the resource has been placed in a node chosen uniformly at random for each experiment.

Optimal PW Size and Expected Search Length in Choose-First PW-RW. We start by applying Theorem 1 to the networks described above



(a) **(b)**

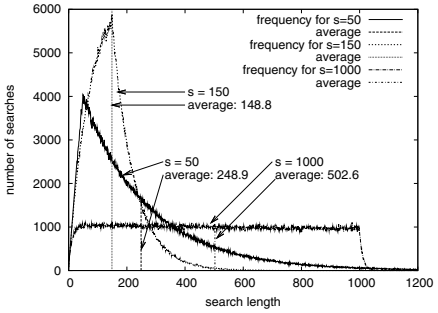
Fig. 2. (a) Expected search length ($\overline{L}_s$) as a function of s when $p = 0$ in a regular network, an ER network and a scale-free network. The optimal points $(s_{opt}, \overline{L}_{opt})$ for each network are $(150, 149)$, $(157, 156)$, and $(174, 173)$. (b) Optimal expected search length ($\overline{L}_{opt}$) as a function of p.

to obtain the expected search length as a function of the size of the PWs.[6] Figure 2(a) provides plots of the expected search lengths ($\overline{L}_s$) given by Equation 1 as a function of the size of the PWs (s), when the probability of a false positive in the Bloom filter is set to $p = 0$, for the three types of networks considered. Results from the analytical model are shown as curves while simulation data are shown as points. The curves for the three networks show a minimum point $(s_{opt}, \overline{L}_{opt})$. This behavior is due to the fact that, when s is small, the number of jumps needed to reach a PW containing the chosen resource grows, therefore increasing the value of $\overline{L}$. In turn, for larger values of s, the number of trailing steps within the last PW grows, also increasing the value of $\overline{L}$.

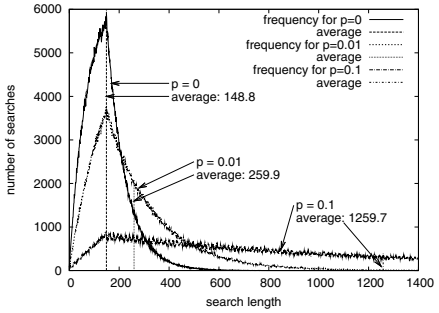
Figure 2(b) shows the linear relation between $\overline{L}_{opt}$ and p (Equation 5). The regular network exhibits the smallest slope, followed by the ER network and then by the scale-free network. For $p = 0$, Equation 5 degenerates to $\overline{L}_{opt} = \overline{L}$, since the search performs all the hops of the total walk (i.e., it is a RW). In fact, Equation 1 also degenerates to $\overline{L}_s = \overline{L}$ in this case, meaning that the expected search length is that of random walk searches regardless the size of the PWs (s).

Distributions of Search Lengths in Choose-First PW-RW. The aim of this section is to experimentally explore how the use of PWs affects the statistical distribution of search lengths.

[6] For each network, the expected length of a random walk search ($\overline{L}$) is needed. We estimate these simulating 10^6 simple random walk searches and averaging their lengths for each network. Average search lengths are denoted in lowercase ($\overline{l}$) to distinguish them from the actual expected value ($\overline{L}$) in the model. The values obtained are: $\overline{l}_{reg} = 11246$, $\overline{l}_{ER} = 12338$, and $\overline{l}_{sf} = 15166$). These results agree with the approximate analytical method in [13] (a modification of the one provided in [5]), which produces the following results: $\overline{l}_{reg} = 11095$, $\overline{l}_{ER} = 12191$, and $\overline{l}_{sf} = 14920$.



(a) Search lengths for $p = 0$ and for $s = s_{opt} = 150$, $s = 50$ and $s = 1000$.



(b) Search lengths for s_{opt} and for $p = 0, 0.01, 0.1$.

Fig. 3. Distributions of search lengths (histograms) with PWs that are not reused in the regular network.

Length Distributions. We first obtain the lengths distributions of searches using PWs that are never reused. Later in this section we will discuss the effect of having a limited number of partial random walks that are reused. We consider each random walk to be the total walk of a search based on PWs. For each original random walk, we break it in pieces of size s, which are taken as the PWs that make up the total walk. Then we consider a search that uses those PWs and count the number of hops (jumps plus trailing steps plus unnecessary steps). This gives the length of the search if it had been constructed using those (precomputed) PWs. Note that the PWs are not reused because they are obtained from independent (real) random walks.

The search length distributions in the regular network for $p = 0$ and for several values of s are shown in Figure 3(a). The average search lengths of each distribution are also shown as vertical bars. These values are very close to the expected values calculated with Equation 1 ($\overline{L}_{50} = 248.9$, $\overline{L}_{150} = 149.0$ and $\overline{L}_{1000} = 510.2$). Therefore, our model accurately predicts average lengths of searches based on PWs of size s in the three types of networks considered.

The shape of the distributions is such that for low s ($s = 50$ in Figure 3(a)) search lengths are dominated by the number of jumps, which is proportional to the length of the total walk. For high s ($s = 1000$ in Figure 3(a)) the distribution adopts a rather uniform shape since search lengths are dominated by the number of trailing steps, assumed to have an approximately uniform distribution between 0 and $s - 1$. The optimal length for the PWs, s_{opt} ($s = 150$ in Figure 3(a)), represents a transition point between these two effects. The shape is such that the values around the average search length (which approximately equals s_{opt}, according to Equation 5) are also the most frequent.

Once it has been found the optimal length for the PWs s_{opt} (known to be independent of p), we investigate the effect of the probability of false positive of Bloom filters in these distributions. Figure 3(b) shows the distributions of search

lengths (histograms) for the regular network when $s = s_{opt}$ and for several values of p. It can be seen that the distributions get wider and lower as p grows, pushing average search lengths to higher values, in accordance with Figure 2(b). However, we observe that the most frequent lengths remain the same regardless of the value of p. For $p = 0$, the most frequent value for each network approximately equals the average search length which, in turn, approximately equals the optimal length of the PWs ($s_{opt} = 150$ for the regular network). For greater values of p, the average search length grows while the most frequent value stays the same. Distributions for the ER and the scale-free networks have similar shapes and are omitted here. However, they have been used in Table 1(a) (explained below).

Effect of Reusing PWs. At this point, we note that we have been assuming that PWs are never reused. However, in practical scenarios it seems quite reasonable to consider a limited number of partial random walks that are reused. In Appendix F of [11] we have explored the distributions of search lengths when the total walks are built reusing a limited number w of PWs precomputed in each node. As it can be readily seen there, we conclude that, for the types of networks in our experiment, just two precomputed PWs per node are enough to obtain searches whose lengths are statistically similar to those that would be obtained with PWs that are not reused. So, we can say that our results using not reused PWs are also valid when using a limited number of PWs that are reused.

Comparison of Performance with Respect to Random Searches. Finally, in Table 1(a) we compare the performance of the proposed mechanism and that of random walk searches. The reduction in the average search length that PW-RW achieves with respect to simple random walks is lower for higher p, ranging from around 98% in the case when $p = 0$ to 88% when $p = 0.1$. Furthermore, we also see that the achieved reductions are independent of the network type.

Table 1. Reductions of average search lengths

(a) PW-RW with respect to random walk searches

	Reduction of $\overline{l}$ (%)		
Network type	$p = 0$	$p = 0.01$	$p = 0.1$
Regular	98.67	97.68	88.73
ER	98.71	97.68	88.42
Scale-free	98.83	97.79	88.43

(b) PW-SAW with respect to PW-RW

	Reduction of $\overline{l}$ (%)		
Network type	$p = 0$	$p = 0.01$	$p = 0.1$
Regular	5.67	8.22	11.24
ER	6.25	9.10	11.88
Scale-free	6.53	9.75	12.65

4 Choose-First PW-SAW

As it was pointed in Section 2 when describing the PW construction mechanism, a possible variation consists of using self-avoiding walks (SAW) instead of RWs.

The resulting mechanism is called PW-SAW. The aim is to revisit less nodes, increasing the chances of locating the resource. In short, a SAW chooses the next node to visit uniformly at random among the neighbors that have not been visited so far by the walk. If all neighbors have already been visited, it chooses uniformly at random among all neighbors, like a simple random walk.

Analysis of Choose-First PW-SAW. When PWs are self-avoiding walks, their concatenation is not a random walk, and hence Theorem 1 is no longer valid. We state a new theorem here for the choose-first PW-SAW mechanism and prove it in Appendix C of [11] using a different approach.

Theorem 2. *If the expected number of trailing steps is assumed to be uniformly distributed in* $[0, s-1]$*, then the expected search length of PW-SAW is*

$$\overline{L}_s = \frac{1}{N} \sum_k n_k \left(\frac{1}{p_{tp}(k)} \cdot (p_n(k) + s \cdot p_{fp}(k)) + \frac{s-1}{2} \right). \tag{7}$$

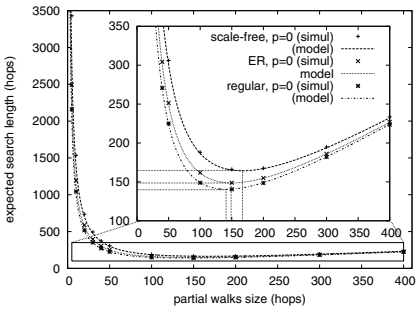
In the above theorem, p_n, p_{tp}, and p_{fp} are the probabilities that the query of the Bloom filter of the chosen PW in the current node returns a (true) negative, a true positive, and a false positive result, respectively, as a funcion of k, the degree of the node holding the resource. The proof in Appendix C of [11] gives expressions for these probabilities.

Expected Search Length in PW-SAW. In this section, we compare the analytic results from the model with experimental data from simulations. Figure 4(a) shows the expected search length ($\overline{L}_s$) as a function of the size of PWs (s) in a regular network, an ER network and a scale-free network, for $p = 0$. The curves in this graph are plotted using the equations in Appendix C of [11]).

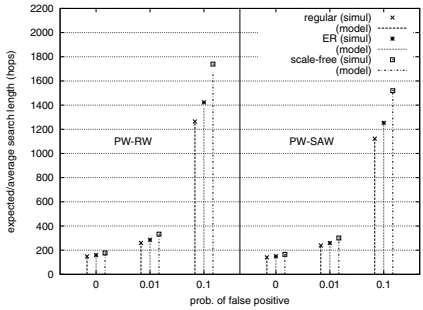
According to the results computed using the PW-SAW model, the minimum search lengths occur for values around $s = 141$, $s = 149$ and $s = 167$ for the regular, ER and scale-free networks, respectively. These values are slightly lower than the ones predicted by the PW-RW model (Figure 2(a)), which were $s_{opt} = 150, 157$ and 174, respectively.

Both the model curves and the simulation experiments have been computed for $w = 5$, chosen as a reference value. However, it has been observed that very similar results are obtained if we change the value of w. Furthermore, plots of the model equations for different values of w are coincident. This behavior was also observed for PW-RW (Section 3.3), where we found that the average search length remained almost constant as we increased w. The reason for this is that the probability of the resource being in the chosen PW does not depend on the number of PWs in the node.

We now compare the results of the PW-RW and PW-SAW mechanisms. Figure 4(b) shows results for PW-RW (left part) and for PW-SAW (right part), in the three networks considered in our study, and for values of $p = 0, 0.01$ and 0.1. Expected search lengths from the analytical models are shown as vertical bars, while average search lengths from the simulations experiments are shown



(a) Vs. s for $p = 0$. Optimal points $(s_{opt}, \overline{L}_{opt})$ are $(141, 139.92)$, $(149, 148.55)$, and $(167, 164.75)$.



(b) Comparison with PW-RW for $p = 0, 0.01, 0.1$.

Fig. 4. Expected search length of PW-SAW in a regular network, an ER network and a scale-free network

as points. The size of the PWs has been set to $s = 150, 157$ and 174 for the regular, ER and scale-free networks, respectively, which are the optimal values predicted by the PW-RW model. For all the networks, we have found a very good correspondence between model predictions and simulation results.

Comparison of Performance with Choose-First PW-RW. The reduction in the average search length that PW-SAW achieves with respect to PW-RW for a given p is largest for the scale-free network, followed by the ER network and then by the regular network. For each network type, the reduction is larger for higher p. Actual values can be found in Table 1(b).

5 Check-First PW-RW and PW-SAW

We now present the check-first versions of the PW-RW and PW-SAW mechanisms, introduced in Section 2. Suppose the search is currently in a node and it needs to pick one of the PWs in that node to decide whether to traverse it or to jump over it. In the check-first mechanisms, it first *checks* the associated resource information of *all* the PWs, and then randomly *chooses* among those with a positive result, if any (otherwise, it chooses among all PWs, as the choose-first version). Performance is improved since the probability of choosing a PW with the resource increases. This comes at the expense of slightly incrementing the processing power used since several PWs need to be checked, but without incurring extra storage space costs.

A minor additional difference between the algorithms is that in the check-first version, the resource information is registered from the *first* node (the node next to the current node) to the *last* node in the PW. This change slightly improves the performance of the new version, since the probability of choosing a PW with the resource increases also in the cases where the resource is held by the last

node of the PW. We have adapted the analysis presented in Section 4 to reflect the new behavior of the check-first mechanisms (see Appendix E of [11]).

Expected Search Length in Check-First PW-RW/PW-SAW. Figure 5 shows the expected search length ($\overline{L}_s$) vs. the size of PWs (s) in a regular network for the four mechanisms presented, for $p = 0.01$ and $w = 5$. The check-first mechanisms achieve a lower minimum expected search length than the original choose-first mechanisms, as expected. In fact, the expected search length can be lowered further by increasing w, the number of PWs per node, clearly at the expense of increasing the cost of the PWs construction stage. In addition, the minimum expected search length occurs for significantly lower s (s_{opt} falls from 150 to about 50), meaning shorter PWs in the nodes, which in turn decreases the cost of the PWs construction stage. As for the PW-SAW mechanisms, we note that both versions achieve a slight decrease in the expected search length with respect to their PW-RW counterparts (which was already observed in Table 1). Results for the ER and scale-free networks are similar and are omitted here.

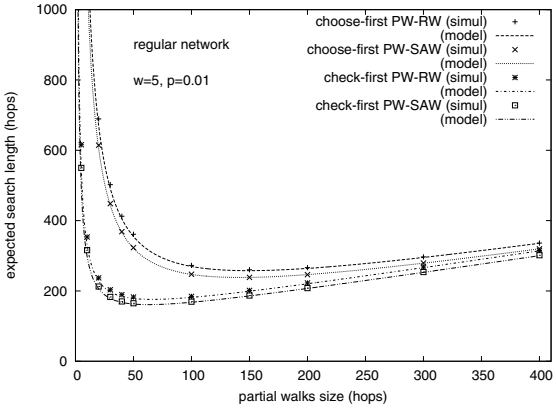


Fig. 5. Expected search length of choose-first and check-first versions of PW-RW and PW-SAW as a function of s in a regular network for $p = 0.01$ and $w = 5$

6 Future Work

The proposed mechanisms could be improved with new strategies to choose from the PWs at the nodes. Smarter variants of RWs could be used as PWs. It would be interesting to compare their application to unstructured P2P networks with algorithms for structured overlays like DHT or quorum systems.

References

1. Lada, A., Adamic, R.M., Lukose, A.R.: Puniyani, and Bernardo A. Huberman. Search in power-law networks. Physical Review E 64(046135) (2001)
2. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th International Conference on Supercomputing, ICS 2002, pp. 84–95. ACM, New York (2002)
3. Yang, S.-J.: Exploring complex networks by walking on them. Physical Review E 71(016107) (2005)
4. Gkantsidis, C., Mihail, M., Saberi, A.: Random-walks in peer-to-peer networks: algorithms and evaluation. Performance Evaluation 63, 241–263 (2006)
5. Rodero-Merino, L., Fernández Anta, A., López, L., Cholvi, V.: Performance of random walks in one-hop replication networks. Computer Networks 54(5), 781–796 (2010)
6. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N., Braynard, R.: Networking named content. Commun. ACM 55(1), 117–124 (2012)
7. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet Mathematics 1(4), 485–509 (2004)
8. Sarma, A.D., Nanongkai, D., Pandurangan, G., Tetali, P.: Distributed random walks. J. ACM 2, 2:1–2:31 (2013)
9. Hieungmany, P., Shioda, S.: Characteristics of random walk search on embedded tree structure for unstructured p2ps. In: International Conference on Parallel and Distributed Systems, pp. 782–787 (2010)
10. Millán, V.M.L., Cholvi, V., López, L., Fernández Anta, A.: Resource location based on partial random walks in networks with resource dynamics. In: Proceedings of the 4th International Workshop on Theoretical Aspects of Dynamic Distributed Systems, TADDS 2012, pp. 26–31. ACM, New York (2012)
11. Millán, V.M.L., Cholvi, V., López, L., Fernández Anta, A.: Improving resource location with locally precomputed partial random walks. arXiv:1304.5100 (2013)
12. Newman, M.E.J., Strogatz, S.H., Watts, D.J.: Random graphs with arbitrary degree distributions and their applications. Physical Review E 64(026118) (2001)
13. Millán, V.M.L., Cholvi, V., López, L., Fernández Anta, A.: A model of self-avoiding random walks for searching complex networks. Networks 60(2), 71–85 (2012)

Distributed B-Tree with Weak Consistency

Gregor V. Bochmann and Shah Asaduzzaman*

School of Electrical Engineering and Computer Science, University of Ottawa, Canada
bochmann@eecs.uottawa.ca
shah.asaduzzaman@gmail.com

Abstract. B-tree is a widely used data-structure indexing data for efficient Retrieval. We consider a decentralized B-tree, were parts of the structure are distributed among different processors and some parts are replicated, thus providing a decentralized indexing structure and parallel operations as desired by modern-day cloud computing platforms. To accommodate the dynamic changes due to data insertion/deletion and changes of the retrieval load, the state of the B-tree is updated by splitting and merging tree-nodes. The traditional update algorithms maintain strong consistency among the replicated states and possibly involve very many tree-nodes. We show in this paper that data retrieval and update can be performed correctly with much weaker consistency criteria. This allows to decompose the necessary updates into smaller update operations that involve only a limited number of tree-nodes, each. We show by analytical models and simulations that with weak consistency the average number of tree-nodes that require updating is reduced compared to the traditional B-tree update algorithms.

Keywords: B-tree, peer-to-peer systems, distributed data retrieval, weak consistency, distributed update operations, distributed databases.

1 Introduction

Massive-scale computing platforms such as computing clouds frequently operate on huge volumes of data. Highly parallel operations are desired by such platforms due to the large number of processing units they have. Consequently, appropriate organization of the data is required such that the high-volume and highly dynamic data set is efficiently accessed and updated without any performance bottleneck.

B-tree is a widely used and well-understood data-structure to index data for efficient retrieval. Highly parallel operations are desired by modern-day cloud computing platforms on high-volume and highly dynamic sets of data. This motivates decentralized indexing structures for data-organization.

In fact, the biggest concern for the cloud computing model, identified in the discussion on the cloud computing research agenda [5] and afterwards, is the enormous overhead and the resulting infeasibility of the strong consistency model assumed in many well-known operations in distributed systems. Thus, it is desired that distributed

* Now with Telenav, Inc., USA.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 159–174, 2013.

and replicated-state data structures be designed in a way that they can tolerate some degree of inconsistency and still function appropriately. This motivates us to design a distributed implementation of the B-tree data structure that works with weak consistency among its replicated components but provides strong consistency in terms of search semantics.

In this paper we identify the consistency conditions that are sufficient for correct and efficient search operation on the distributed B-tree indexing data structure (Section 3). We then define algorithms for updating the data structure keeping these consistency conditions maintained (Section 4). The data structure is generalized for key-spaces of arbitrary dimensions. The system model, assumptions and the particular way of distributing the B-tree structure are introduced in Section 2.

2 System Model and Assumptions

2.1 B-Tree Structure

We consider the B+ -tree variant of the B-tree, which is possibly the most widely used variant of the data structure. In a B+ -tree, all nodes have the same structure. Each of the leaf nodes maintains data-keys pertaining to a certain range in the key-space. Each internal node effectively maintains a list of entries, each containing a key-range and a pointer to some other node corresponding to this range. B-trees were designed for indexing one-dimensional data-spaces. So, the ranges were effectively expressed by integer data-keys, or points in the linear key-space.

Among the design goals of B-trees were (a) efficient use of disk blocks, and (b) keeping the search tree balanced while growing or shrinking. For keeping the tree balanced, a global parameter d is introduced which defines the maximum number of entries to be held by a node. The root node of the tree describes the whole key-space or key-universe and each of the other nodes describes a portion or sub-range of the data-universe. Describing a range means dividing the range into sub-ranges and maintaining pointers to the child nodes that describe each of the sub-ranges. If n is the number of child pointers or sub-ranges described by a node, one normally maintains the relation $\lceil d/2 \rceil \leq n \leq d$ in order to balance the amount of information stored in each node. In the case of a one-dimensional key-space (as used in our examples), a sub-range is characterized by two key values, the minimum and maximum key values of the sub-range.

In the following general discussion and the presented algorithms, we assume a generalization of B-trees for key-spaces of arbitrary dimensions, instead of a single dimension. Thus, we avoid any particular way of expressing the division of ranges, such as by points for one dimension as in a B-tree, or by lines or rectangles for two-dimensions as in Rtree [8] or Quad-tree [7]. We assume that each tree-node maintains the definitions of N sub-ranges of the whole range it describes, along with one pointer to another tree-node for each of the sub-ranges. Figure 1(a) shows an example of such B-tree. In all our examples in this paper, we use a one-dimensional key-space with consecutive sub-ranges. In the rest of the paper, the term **B-tree** will be used to denote a centralized implementation of such a generalized tree-based indexing structure.

2.2 Distributed Implementations of B-Tree

When the number of data records is huge and/or the access load becomes too large for a single computer, a distributed B-tree must be considered. Simply replicating the whole data structure on several computers is not practical because of the difficulty of the update operations. In this paper we consider what we call "decentralized distribution". For disambiguation between tree-nodes and processing nodes, we denote the latter as processor, while node refers to tree-nodes.

Centralized Distribution. The intuitive method for distributing the tree data structure is to place each tree-node on one processor. The scalable distributed B-tree proposed by Aguilera et al. [2] and the tablet hierarchy in the internal representation of Google's Bigtable [6] structure use such representations. Although this allows the update algorithms on the structure for data insertion/deletion to be similar to the centralized version, the processors holding the root or the higher level tree-nodes get overburdened with search traffic. A typical solution to this problem, used in both [2] and [6], is caching or replicating the higher level nodes of the tree in the user or client computers, such that traversing higher level nodes can be avoided. However, this involves additional overhead for maintaining consistency among the replicas, and may not be suitable for highly dynamic data sets.

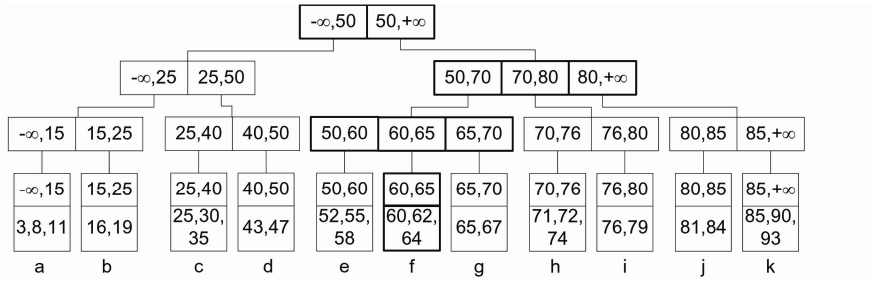
Decentralized Distribution. An alternative distribution of the tree structure is possible, following the decentralized design philosophy, assigning equal workload and the same role to each processor node. So, instead of assigning the responsibility of one tree-node to one processor, one branch of the tree, i.e. the path from root to a leaf node, is assigned to one processor. Thus, the higher level tree-nodes are, in a sense, replicated in proportion to their usage, and hence, the workload due to traversal operations is equally distributed among the processors.

To represent a branch of the tree, each processor i maintains a routing table data structure RT_i with multiple levels, each level representing one node of the branch. Level l of RT_i , denoted as RT^l_i, corresponds to a level-l node of the B- tree. RT^l_i is a set of entries or tuples c together describing a range LR^l_i in the key-space. Each r is a sub-range of LR^l_i and the corresponding j refers to some processor j (may be i itself) that holds the level l-1 node of the B-tree describing r, that is, $RT^{l-1}{}_i$ represents the child node and $LR^{l-1}{}_i$ = r. Representation of one branch to the leaf-node f for the example B-tree in Figure 1(a) is shown in Figure 1(b).

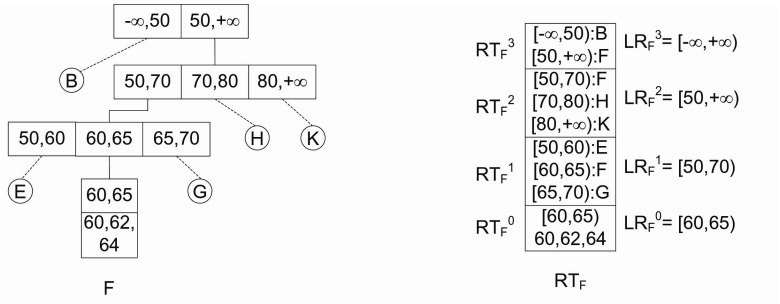
Because non-leaf nodes are replicated in multiple processors, one for each branch, there are multiple options for j if l -1 is a non-leaf level, and any one of them may be chosen for the entry <r, j>. Also, the range r for different entries in an RT^l_i are non-overlapping and the union of these ranges constitutes LR^l_i (this is the same as in a normal B-tree). The lowest level, RT^0_i corresponds to a leaf node of the B-tree, and stores the set of keys in the range LR^0_i delegated to processor i, and the pointers to corresponding data items. Note that the size of the tree state maintained at each processor is O(logN), where N is the total number of keys in the whole structure.

A similar distributed implementation of a tree structure has been proposed in [10], called DPTree. Although a DPTree builds the tree-structure on top of a distributed hash table used to name and discover the tree nodes, such decentralized structure can be maintained without such overlay, as shown in [3] (see also [14]). In this paper we do not consider distributed hash tables because we want to support range queries.

The above references consider that a single (global) B-tree is distributed and partially replicated over all the processors. We call this situation global consistency. It implies that updates of nodes in the upper part of the B-tree, which are replicated in many processors, require complex update operations. We consider in this paper a situation with weak consistency where an update of any node in the B-tree involves only a few processors.



(a) An example of a B-Tree



(b) Consistent decentralized implementation of the B-tree. The view of the tree from processor F (left). The routing table maintained by processor F is shown (right)

Fig. 1. A *B*-tree and its consistent decentralized implementation. The leaf-level tree- nodes are referred by small letters (a, b, c, ...) and the processors holding the corresponding leaf-nodes are referred by capital letters (A, B, C, ...)

2.3 Assumptions

We assume in this paper a decentralized distributed implementation of a B-tree as described above. We assume that the processors can use an asynchronous message-passing system [4], where each processor contains its own local memory (or persistent

storage), the processors communicate among them through messages, all processors run the same program and there is no master clock to synchronize the events in the processors.

We follow a peer-to-peer model, where the search operation can be initiated from any processor. Thus, the client application may consider any of the processors in the distributed B-tree as a portal to the search service.

For fault-tolerance, a processor in our model may be realized by a small cluster of computers, replicating the state of one processor. Details of implementing a fault-tolerant processor from faulty processing nodes may be found at [11]. We assume that a message sent to another processor is eventually received by that processor in finite amount of time, although messages may be delivered out of order. The message channels may be made reliable through use of an end-to-end transport protocol [1]. We assume a complete network model, where any processor is able to send messages to any other processor as long as the address of that processor is known.

3 Search and Updates in Decentralized B-Tree

3.1 Search Algorithm

To search a target key d_t (or a range r_t) in the decentralized B-tree, the primary goal is to find the processor i (or a set of processors P) such that $d_t \in LR^0_i$ (or $\text{union}_i \in P$ includes r_t). The search can be initiated from any processor. Navigation of the request from the initiator to the target processor is performed by Algorithm 1. The initiator processor calls the Algorithm 1 with level l parameter equal to the topmost level of its own routing table.

Algorithm 1: Search(*i, d_t , l*)
1: **Initiator:** processor i
2: **Condition:** a query received to resolve d_t at level l
3: **Action:**
4: **if** $l = 0$ **then**
5: Result is processor i
6: **else**
7: Find <r,j> $\in RT^l_i$, such that $d_t \in$ r
8: Forward the query to j as *Search(j, d_t , l -1)*
9: **end if**

For range search, instead of finding one <r,j> $\in RT^l_i$, all {<r,j> | r $\cap$ $r_t \neq \phi$} are looked up and the navigation proceeds next level to all the j's in parallel. The time complexity of both point and range search algorithms are clearly O(logN), although the message complexity is higher for the range search (O(N) in the worst case, if all the processors are included in r_t).

3.2 Updates in a Globally Consistent Decentralize B-tree

The data structure needs to be updated as keys are inserted or deleted. The B- tree data structure grows with key-insertion by splitting a node when the number of entries overflows, and shrinks with key-deletion by merging two sibling nodes. In the decentralized B-tree, leaf level split and merger are simple. However, because non-leaf nodes are replicated in many processors, split/merge operations in non-leaf levels require a large number of nodes to be updated atomically, which may require the updates to be coordinated by a single master processor. In the worst case, when the state of the root node is changed, the update needs to be atomically propagated to all the processors.

Figure 2(a) illustrates the split of the tree node f after insertion of data element 63 by maintaining a single consistent view of the tree at each processors. Splitting at the leaf level is relatively simple. Part of the data-keys at processor F is now moved to a new processor F2. Because the level-1 tree node is modified, level-1 at processors E and G need to be updated. Also, whichever processor held F responsible for its level-0 range [60, 65) need to be updated about the change.

When the level-1 tree-node, containing the range [50, 70) needs to be split (Figure 2(b)), it involves splitting the level-1 of processors E, F , F2 and G. This causes the level-2 tree-node to have one new entry, which requires all the processors E through K to add an entry at their level-2. Finally, whichever processors held any of E, F , F2 or G responsible for its level-1 range now need to update their pointers. Thus even a level-1 split for consistent B-tree with fan-out of only 2 - 4 involves atomic update of the states at 10 - 12 processors.

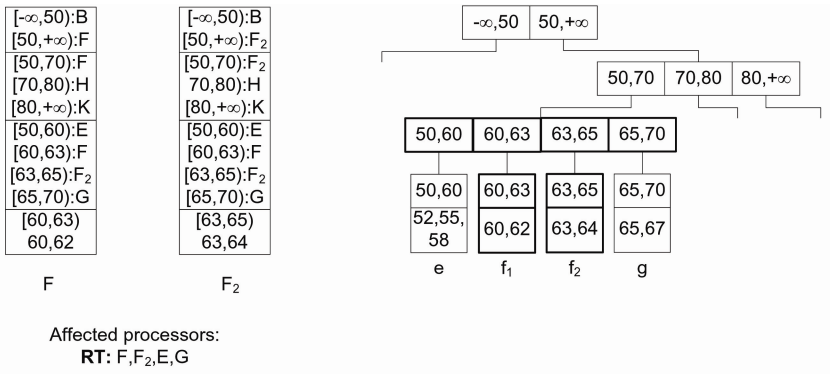
The huge overhead of large-scale atomic updates in the consistent decentralized B-tree structure motivates us to look for weaker consistency conditions that are easy to maintain through much smaller-scale updates, and yet sufficient for correct search operations.

3.3 How Much Consistency Is Needed?

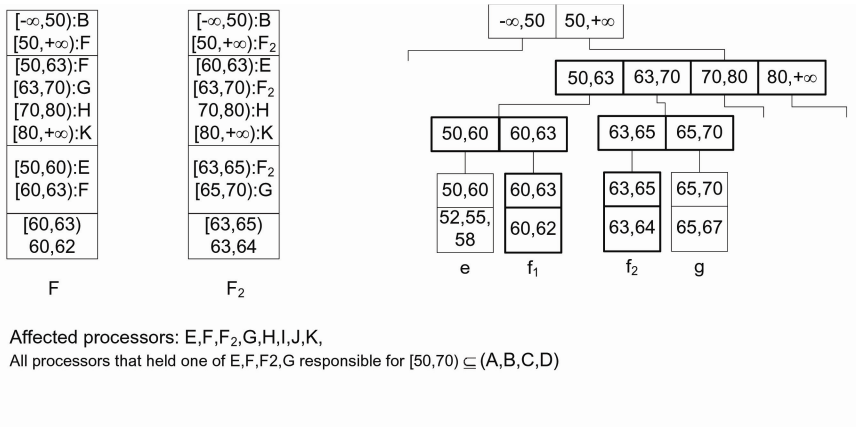
Here we define consistency conditions among the components of the decentralized B-tree structure maintained by different processors that are sufficient for ensuring the correctness of the search operation through Algorithm 1, but weaker than the constraint that all component-states are consistent with a single global B-tree.

First, any processor should be able to initiate the search, so every processor should maintain a description of the key-space universe (U) at the topmost level of its routing table. We call this condition **invariant of universal coverage:**

AU: for all i : $LR^{m}_{i} = U$, where m is the highest level in RT_i



(a) Splitting leve-0 range [60,65)



(b) Splitting level-1 range [50, 70)

Fig. 2. Updates in a consistent decentralized B-tree. The original tree is shown in Figure 1

For correct navigation, if an entry <r, j> is in RT^l_i , then its target j must describe at least the range r at level *l - 1*. Formally, this defines the **invariant of navigability:**

AN: for all *i* and *l* : <r, j> ϵ RT^l_i implies r is included in LR^{l-1}_j

Another condition is necessary depending on the semantics of the search operation. If we allow different processors to have overlapping local ranges at the leaf level, then for a search query for d_t , where $d_t \in LR^0_i$ and $d_t \in LR^0_j$ with $i \neq j$, Algorithm 1 ensures delivery of the query to at least one of i and j. This result is correct, if all keys in the intersection of LR^0_i and LR^0_j are available in both TR^0_i and TR^0_j . If overlapping coverage of ranges by different processor does not imply such exact replication of all keys in the common range, then the usual semantics of search requires the query to reach all such processors. To keep things simple, we impose the following **invariant of disjoint local ranges:**

ALR: for all *i* and *l* : i ≠ j implies LR^0_i and LR^0_j are disjoint.

Theorem 3.1: *AU, AN and ALR are sufficient conditions for the correctness of ex- act and range search operations in the decentralized B-tree using Algorithm 1.*

Proof. Line 8 of Algorithm 1 ensures that the algorithm proceeds at least one level towards level 0 at each hop. Thus the algorithm terminates in a number of steps not larger than the maximum number of levels in the routing table of any processor.

At each hop in the navigation, an entry <r, j> ∈ RT^l_i , d_t ∈ r must always be found at Line 7. AU ensures that, if such an $<r_1, p>$ is found at level *l* of the current processor *i*, an entry $<r_2, q>$, $d_t \in r_2$ can be found at level *l* - 1 of the next hop processor p. So, by induction, we observe that the query is finally forwarded to a processor p such that d_t ∈ LR^0_p . ALR ensures that only one such processor exists. The proof can be easily extended to show the correctness of the range-search algorithm.

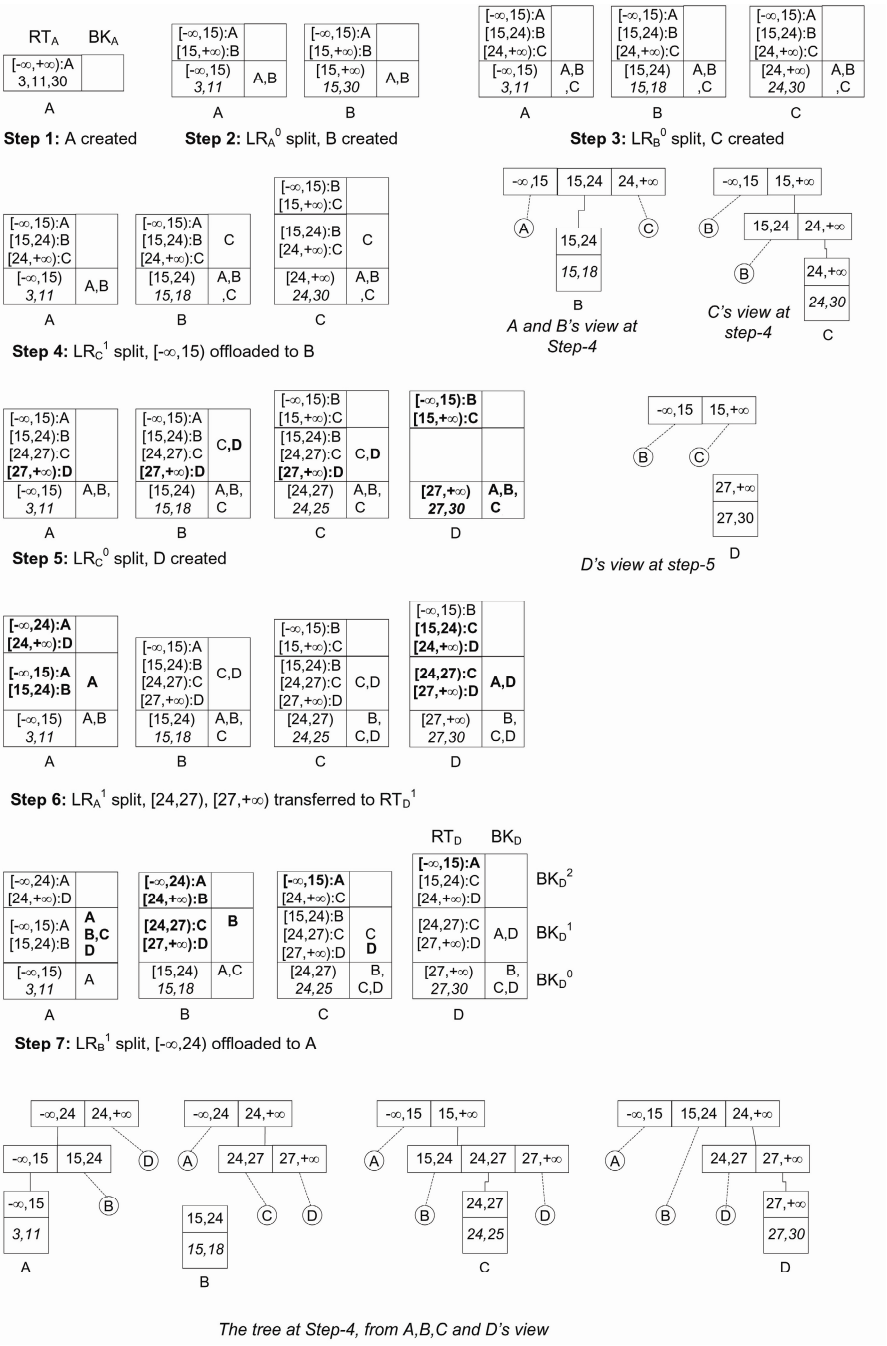
The decentralized B-tree structure that maintains the conditions AU , AN and AR, in general, is a weakly-consistent structure, because several conditions valid in the consistent decentralized B-tree structure have been relaxed. For example, in the normal B-Tree structure, we have a stronger invariant of navigablity

AN(strong): for all *i* and *l* : <r, j> ∈ RT^l_i implies r = LR^{l-1}_i
where r is equal to LR^{l-1}_i instead of included. Also, in the consistent structure, each level of the routing table contains a self-pointer, i.e. the condition

Self-pointer: for all *i* and *l* : there exist <r, j> ∈ RT^l_i
is valid, but this is not maintained in the weakly-consistent structure. In addition, the number of levels of the routing table may be different for different processors. The condition that each node in the tree must maintain a number of entries n such that $\lceil d/2 \rceil \leq n \leq d$, is also relaxed. The lower and upper limits are now rather soft-limits. As a result, the cascaded split or merge operations are treated as separate update operations.

Figure 3 shows how a weakly-consistent B-tree structure may grow through insertion of data-keys. Initially, through the first three steps, the view of the tree remains consistent for all three processors A, B and C . From step-4 onwards, different processors may have different views of the tree. It may be noted that with such weak-consistent updates, the view of the data structure at some processors may no longer remain a single connected tree. Rather, the view may be of several disconnected segments of the tree. Nevertheless, each processor maintains sufficient information to route any search query originated at any processor.

The update operations are initiated independently and asynchronously by individual processors. Compared to the updates in a consistent B-tree shown in Figure 2, which, even for a level-1 split, require updating the states of a large number of processors atomically, updates here are much less invasive. For example, starting from the same states as in Figure 2, weak-consistent updates at level-1 could be initiated independently by the processors E, F, F2 and G, and each of them would involve the states of 3 - 4 processors known to them by routing table entries. The algorithms presented in the next section will explain these asynchronous updates.



The tree at Step-4, from A,B,C and D's view

Fig. 3. Evolution of a weak-consistent B-tree with asynchronous updates. To facilitate asynchronous update, each processor i maintains a table BK_i in addition to RT_i . BK^l_i holds the names of all processors that hold RT^l_i responsible for the whole or some part of LR^l_i

The update operations are initiated independently and asynchronously by individual processors. Compared to the updates in a consistent B-tree shown in Figure 2, which, even for a level-1 split, require updating the states of a large number of processors atomically, updates here are much less invasive. For example, starting from the same states as in Figure 2, weak-consistent updates at level-1 could be initiated independently by the processors E, F, F2 and G, and each of them would involve the states of 3 - 4 processors known to them by routing table entries. The algorithms presented in the next section will explain these asynchronous updates.

Although the weak consistency leads different processors to maintain different views of the tree, the search operation remains correct and can be initiated from any processor. The search always progresses one level at each hop, and thus, terminates after a number of hops equal to the maximum height of the tree (i.e. the maximum level in the routing table) in any of the views.

4 Updates with Weak Consistency

In this section we describe the atomic update operations needed to adapt the decentralized B-tree structure when data keys are inserted or deleted, or when some processor is overloaded. The basic insertion and deletion operations works similarly as in a traditional B-tree, i.e. first the target key (or its position) is searched, and then the deletion or insertion is performed. Insertion of keys may cause overflow in a leaf level node, which then splits, and the split may be cascaded to higher level nodes. Similarly deletion of keys cause underflow and triggers merger of nodes. Because lower and upper limits in the number of entries are now soft-limits, the cascading splits (or mergers) are treated as separate atomic operations. Here we define the atomic split and merge update operations for leaf level and non-leaf level separately. The update algorithms assume that the three consistency conditions AU , AN and ALR are satisfied when the update is started, and assure that the conditions will be satisfied again when the update completes.

The update algorithms are triggered independently by any processor. One principle followed in the design of the update algorithms is to modify the states of a minimal number of processors. Specifically, the modification is limited to the neighbour processors only, i.e. the processors known to the local routing table of the initiating processor. To facilitate the updates, an additional table called backward pointer table is maintained by each processor (the table of processor *i* is denoted as BK_i). For any processor *i*, BK_i has the same number of levels as RT_i . $j \in BK_i$ if and only if $<r, i> \in RT^{l+1}_j$.

Each update operation may need to modify the states (routing tables) of a few neighbouring processors. To ensure correctness in the presence of concurrent updates, some concurrency control mechanism must ensure atomicity of each update. To allow a higher degree of parallelism, a version-number-based optimistic transaction protocols may be used [9]. In this method, a counter or version number is maintained for

the state of each processor. The version number is incremented whenever the state is successfully modified. The initiating processor that executes the update algorithm reads the necessary states along with their version numbers. After computing the modified state locally, it then attempts to commit the new states to appropriate processors. The update transaction is aborted if any of the states in the write-set has a different version number than the one that was read initially. Aborted transactions are retried at a later time. While describing the update algorithms, we clearly mention which processor initiates it (initiator), and which state in which processors are read (Readset) and updated (Writeset). Version control may be applied at different granularities on the states. Each row of RT and BK tables at each processor may be separately versioned for maximum parallelism.

4.1 Split Algorithms

Algorithm 2 describes the procedure to split the local range LR^0_i of processor *i* into 2 disjoint ranges LR_1 and LR_2 , and offloading LR_2 to a newly recruited processor *j*. Because *i* loses part of LR^0_i , for all $p \in BK^0_i$, RT^1_p need to add entries pointing to the new processor *j* instead of *i* for the lost part of the range. BK^0_i may include *i* if RT^1_i has a self-entry (Lines 6-13). In addition to the leaf level, the topmost level of the new processor j's routing table, RT^m_j is initialized by a replica of RT^m_i (Line 13). Mid-levels of RT_j remains empty. For the nodes newly pointed to by *j* at level m, their backward pointers are updated (Line 14).

Algorithm 3 is executed when processor *i* wants to offload some entries from its routing table RT^l_i at level $l > 0$. Unlike the case of leaf-level split, no new processor is recruited here. So, the major challenge here is to find an existing processor *j*, whose routing table at the same level, RT^l_j , either already contains some entries covering some common range with LR^l_i, or, have some space to take few entries from RT^l_i. In a consistent distributed B-tree, we have for all j : if $<r, j> \in RT^l_j$ then $RT^l_j = RT^l_i$. Thus, neighbours in RT^l_i are natural target for offloading part of RT^l_i. In the weak-consistent structure, it is not certain that such a *j* will be found in RT^l_j , so, other neighbours are searched including all backward pointers. Also, in the leaf-level split, the mid-levels of the new processor's routing table are kept empty. Non-leaf level splits are initiated for the lowest overloaded level. So, there is high possibility of finding a *j* in RT^l_i with empty space in RT^l_j .

Once a suitable *j* is found, the update procedure is straightforward. The entries are transferred from RT^l_i to RT^l_j and BK^{l-1}_p are updated for the processors corresponding to the transferred entries (Lines 6-8). Then for the processors in BK^l_i, i.e. those who held *i* responsible for some part of LR^l_i , now need to update for the range shifted to *j*, by adding a new entry in the level l + 1 of their routing tables. Backward pointers of *i* and *j* are also updated accordingly (Lines 9-13). Finally, if the topmost level of RT^l_i is split, one additional level is added to hold the pointer to the transferred range, such that the whole universe is described.

Algorithm 2: SplitLeafNode(*i*)

1: **Initiator:** processor i
2: **Condition:** RT^0_i is overloaded, in terms of storage or access load
3: **Action:**
4: Partition RT^0_i into two disjoint sets of keys D_1, D_2
and LR^0_i into two corresponding ranges LR_1, LR_2
5: Find a new processor *j*
6: $RT^0_i := D_1$; $RT^0_j := D_2$;
7: $LR^0_i := LR_1$; $LR^0_j := LR_2$;
8: **for all** p ∈ BK^0_i **do**
9: there must exist <x,i> ∈ RT^1_p with x = LR^0_i
10: $RT^1_p := RT^1_p \setminus$ <x,i> union {< LR_1, i> , < LR_2, j> }
11: add {p} to BK^0_j
12: **end-for**
13: $RT^m_j := RT^m_i$ where m is the topmost level of RT_i
14: **for all** p **such that** <r, p> ∈ RT^m_i **do** add {j} to BK^{m-1}_p

Algorithm 2: SplitNonLeafNode(*i, l*)

1: **Initiator:** processor i
2: **Condition:** RT^l_i has too many entries or is causing too much routing load
3: **Action:**
4: find some existing processor j such that RT^l_j is empty
or has some overlap with LR^l_i
5: partition LR^l_i into two subranges R_s and R_x
where R_x is equal to the overlap (if there is one)
and partition RT^l_i into two corresponding sets E_s and E_x
6: $RT^l_i := RT^l_i \setminus E_x$
7: if there is no overlap do $RT^l_j := E_x$
8: **for all** p **such that** <r, p> ∈ E_x **do** delete {i} from BK^{l-1}_p and add {j}
9: **for all** p ∈ BK^l_i **do**
10: there must exist <x,i> ∈ RT^{l+1}_p where x is included in LR^l_i
11: remove <x, i> from RT^{l+1}_p and add < R_s, i>
12: add < R_x, j> to RT^{l+1}_p
13: add {p} to BK^l_j ; **if** x ∩ R_s = ϕ **do** remove {p} from BK^l_i
14: **if** *l* is the topmost level of the routing table RT_i **do**
15: RT^{l+1}_i := {< R_s, i> , < R_x, j> } ; add {i} to BK^l_i

For finding an existing processor j that can be used for off-loading some of the entries from RT^l_i (see line 4 of the SplitNonLeafNode algorithm), we have explored two algorithms by simulation [12] : (1) The Ping algorithm checks the descendants of RT^l_i in the distributed B-Tree to check whether their processor contains at level *l* en empty routing table or a table that has an overlap with RT^l_i . If no suitable processor j is among them, then the algorithm checks the next-lower descendents of RT^l_i in the tree, possibly until level 0 is reached. (2) The Ping-Pong algorithm goes down one level (like the Ping algorithm) but then follows the back-pointers that point to routing tables

at level *l*. Because of the two steps, a larger number of processors is reached. Again, if no suitable processor j is found, the Ping-Pong process is repeated by going down two levels and going up two levels, and so on.

Our simulation studies confirmed that both of these algorithms always find a suitable processor j in reasonable time on average [12].

4.2 Merge Algorithms

When there are too few data items in a processor *i*, it decides to release itself by merging its items and routing table with those of another processor. Algorithm 4, described in detail in an earlier version of this paper [13], describes the update procedure for such a merger. First a suitable partner *j* for the merger is selected among the processors backpointed at level 0, such that RT^1_j points to *i* for some range x included in LR^0_i. If RT^1_i is pointing to *j* for some other range y, then after the merger, the two entries <x, *i*> and <y, *j*> can be merged into <x union y, *j*>. Because processor *i* is being released, all levels of its routing table are merged with the corresponding level of *j*'s routing table. Accordingly, for all *l* and p ∈ BK^l_j , RT^{l+1}_p are updated.

Similar to the level-0 merger, if any other level *l* of the routing table of a processor *i* is found under-loaded, the entries of that level can be merged with the same-level entries in another processor. The merging partner, *j*, is found in a similar way as before, among the p ∈ RT^{l+1}_p , so that after the merger one entry is eliminated there. If RT^{l+1}_p is the topmost level, and contains only one entry after the merger, then that level may potentially be eliminated. This procedure, called Algorithm 5, is also described in [13].

4.3 Proving the Invariants

Theorem 4.1: *The update algorithms, Algorithms 2, 3, 4 and 5 maintain the invariants AU , AN and ALR .*

AU: Algorithm 2 maintains AU in the newly joined processor *j* by copying the top level of the routing table of *i* (Line 14). In Algorithm 3, the range LR^{l+1}_p in all p ∈ BK^l_i remains unchanged by the modification following Line 9. If RT^l_i is the topmost level in RT_i then the additional update in Line 14 ensures AU for processor *i*. Concerning Algorithms 4 and 5, see [13].

AN: AN can be violated only when LR^l_i for some processor *i* and some level *l* is reduced. In Algorithm 2, LR^1_i is reduced, and so, RT^1_p is updated for all p ∈ BK^0_i to maintain AN (Line 10). A similar update is performed in Line 13 of Algorithm 3, for the reduction in LR^l_i.

ALR: Violation of ALR is possible only when RT^0_i is created or extended for any *i*. In Algorithm 2, LR^0 is modified for processors *i* and *j* only, and no overlap is formed (Line 7). In Algorithm 4, LR^0_i and LR^0_j are merged into LR^0_j , and then processor *i* is removed. So no overlap is created. Algorithms 3 does not modify LR^0 of any processor.

In an elementary state of the decentralized B-tree structure, when there is only one processor having only one level in its routing table, all the invariants AU , AN and ALR are valid. So, by induction over successive updates, it can be proved using Theorem 4.1 that all three invariants are always maintained for the structure. Also, all four update algorithms work assuming the three invariants only. Validity of the backward pointers is also maintained in these algorithms whenever a forward pointer is updated.

5 Discussion

As mentioned in Section 2.2, we use a decentralized distributed implementation of the B-Tree similar to the BPTree [10] or as in [3]. The main difference is that we relax in this paper the requirement for global consistency of the B-Tree data structure. Through the use of the weak consistency the tree update operations (split and merge) can be performed much more effectively. In the decentralized B-tree architecture, the average number of processors involved in a routing table split operation at level *l* (the SplitNonLeafNode algorithm described above) is (2 + b) which is independent of the level. Here the number 2 accounts for the processor *i* of the node being split and the node *j* to which some of the entries are transferred. b represents the average number of back-pointers of the split node. This value is equal to the average fan-out of a node, which is ¾ d (if we assume that the relation $\lceil d/2 \rceil \leq \backslash \leq d$ is maintained).

In the case of a B-Tree with global consistency, all copies of the routing table at level *l* must be updated concurrently in a single transaction. Since the number of copies of a routing table at level *l* is b^l, this becomes a very big number when the level is close to the root, and if the root table must be updated, this involves all processors.

This shows the main advantage of weak consistency. The main disadvantage of weak consistency is the fact that it is more difficult to find a suitable processor for node splitting or merging due to the irregular structure of the B-tree after repeated data insertions and deletions. We note, however, that the average number of tree nodes to be updated due to a single data object insertion is the same for strong and weak consistency. If we assume that the number of data objects per processor is also limited by d, then the probability that a data insertion leads to the splitting of a leaf node is equal to 2/d (assuming that the number m of data objects is in the range $\lceil d/2 \rceil \leq m \leq d$). And such a split will lead to the update of b routing tables at level 1, where again, for each of these updates, there is the probability of 2/d that the routing table at level 1 will be split (i.e. b * 2/d = 1.5 splits at level 1), and so on. Therefore, the average number of routing tables to be split after one data object insertion is given by 2/d $(1 + 1.5 + 1.5^2 + \ldots + 1.5^N)$. In the case of the B-Tree with weak consistency, these splits can be performed as separate transactions, each involving only (2+b) processors (as mentioned above). In the case of strong consistency, the probability of having a split at level *l* after a data object insertion is equal to $(2/d)^{l+1}$, however, the number of processors involved for an split at level *l* would be $(\frac{3}{4} d)^l$, which becomes prohibitive for the root node.

6 Conclusion

We have demonstrated that it is possible to distribute a B-tree for data retrieval over a large number of processors with partial replication of the interior nodes of the tree over the different processors without full consistency. Enforcing only weak consistency conditions necessary for the correct operation of the retrieval function, it is possible to define tree update operations that can be initiated by one of the processors and would involve only the local state of the tree and the state in a few neighbour nodes, without requiring simultaneous updates in all processors that have a replica of the state being updated.

We have proved that the new update algorithms maintain our weak consistency conditions and that these conditions guarantee correct operation of the data retrieval algorithm that requires L steps where L is the depth of the B-tree. Through our simulation studies, we have shown that the depth of the tree can be maintained over a long period of tree update operations at the optimal level of L = log(N), where N is the number of processors in the system.

References

1. Afek, Y., Gafni, E.: End-to-end communication in unreliable networks. In: Proc. 7th ACM Symposium on Principles of Distributed Computing, PODC 1988, pp. 131–148 (1988)
2. Aguilera, M.K., Golab, W., Shah, M.S.: A practical scalable distributed B-tree. Proc. VLDB Endow. 1(1), 598–609 (2008)
3. Asaduzzaman, S., Bochmann, G.V.: GeoP2P: An adaptive peer-to-peer overlay for efficient search and update of spatial information. Unpublished document (2009), http://arxiv.org/abs/0903.3759
4. Bar-Noy, A., Dolev, D.: A partial equivalence between shared-memory and message-passing in an asyn. fail-stop distr. env. Mathematical Systems Theory 26, 21–39 (1993)
5. Birman, K., Chockler, G., Renesse, R.V.: Toward a cloud computing research agenda. ACM SIGACT News 40(2), 68–80 (2009)
6. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. ACM Transactions on Computing Systems 26(2), 1–26 (2008)
7. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. Acta Informatica 4(1), 1–9 (1974)
8. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD 84, 47–57 (1984)
9. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Trans. Database Syst. 6(2), 213–226 (1981)
10. Li, M., Lee, W., Sivasubramaniam, A.: DPTree: A Balanced Tree Based IndexingFramework for Peer-to-Peer Systems. In: 14th IEEE ICNP, pp. 12–21 (November 2006)
11. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys 22(4), 299–319 (1990)

12. Hafaiedh, K.B.: Studying the properties of a distributed decentralized B+ tree with weak consistency. Master Thesis, University of Ottawa (Oct. 2012)
13. Asaduzzaman, S., Bochmann, G.V.: Distributed B-tree with weak consistency, unpublished report, see http://www.site.uottawa.ca/~bochmann/Curriculum/Pub/2010%20-%20Distributed%20B-tree%20with%20Weak%20Consistency.pdf
14. Asaduzzaman, S., Bochmann, G.V.: A locality preserving routing overlay using geographic coordinates. In: IEEE Intern. Conf. on Internet Multimedia Systems Architecture and Application, Bangalore, India (December 2009)

Consistency in Distributed Storage Systems

An Overview of Models, Metrics and Measurement Approaches

David Bermbach and Jörn Kuhlenkamp

Karlsruhe Institute of Technology,
Karlsruhe, Germany
firstname.lastname@kit.edu

Abstract. Due to the advent of eventually consistent storage systems, consistency has become a focus of research. Still, a clear overview of consistency in distributed systems is missing. In this work, we define and describe consistency, show how different consistency models and perspectives are related and briefly discuss how concrete consistency guarantees of a distributed storage system can be measured.

Keywords: Consistency, Distributed Systems.

1 Introduction

In distributed storage systems replication can be used to increase durability and availability of data as well as to enable fault tolerance and low latencies for distributed clients. This comes with a price, though, as multiple copies add the burden of keeping replicas identical. With the advent of the Internet and lately Cloud Computing, replication has become the number one mechanism to deal with scalability issues, load variance and large numbers of parallel requests. This in turn has brought *consistency* to the attention of both businesses and researchers as there is now a multitude of storage systems each offering different consistency guarantees which cannot be easily measured. As the CAP theorem and the PACELC model[18,1] mandate, there are direct trade-offs between consistency and availability as well as consistency and request latency of a replicated storage system and each system chooses a different spot on the continuum of consistency guarantees between strict consistency and no consistency.

There is much work on consistency models, their implementations and measurements, but the relationships between models and measurement approaches is not always clear. Furthermore, researchers from the database community have an entirely different understanding of consistency than researchers from the distributed systems community.

In this work, we try to shed some light on these issues. We start with a brief description of what consistency means in both research communities before taking the distributed systems view and analyzing different perspectives on consistency as well as the relationship between various consistency models. Next, we

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 175–189, 2013.

combine this with our previous work on consistency benchmarking and describe continuous metrics and setups for experimental consistency measurements.

The main contribution of this work is, hence:

- A comprehensive overview of consistency models and guarantees, including discussions why particular guarantees are useful in different use cases.
- An analysis how consistency models can be ordered by their strictness and how client-centric guarantees relate to data-centric models.
- A discussion of different metrics to describe consistency guarantees as well as a brief analysis of existing benchmarking approaches.

2 Definitions

The term consistency is derived from the latin word *consistere* which means "standing together" or also "stopping together". Hence, consistency generally describes relationships between items that are somehow connected. When considering consistency of data, a consistent state requires that all relationships between data items and replicas are as they should be, i.e., that the data representation is correct. This focus on the correctness can be seen in both the database as well as the distributed systems community – but on different levels.

2.1 Database Systems

Researchers from the (relational) database community focus on consistency in the context of ACID guarantees. A set of operations is abstracted into a transaction that will execute entirely or not at all (atomicity). If it executes, its changes will be permanently visible (durability). Multiple transactions may execute concurrently but use suitable mechanisms to show the same behavior as if all transactions were executed according to some global serializable order. This can be done in various degrees of isolation[1] ranging from true *serializability* (which is rarely offered in actual products [5]) to *read uncommitted*. Transactions guarantee that the database always adheres to a global schema (independent of whether the database is replicated or not) where all integrity constraints, a set of conditions and requirements, are observed [15] (consistency).

Hence, the consistency focus of database systems is on the relationships between data items and the overall correctness of the entire database. They can be guaranteed in a distributed setting but it is expensive to do so as consistency and isolation are typically guaranteed via locking mechanisms which create an extensive communication overhead in a distributed setting. This is also caused by the fact, that transactions were intended as "a simple programming abstraction to cope with concurrent executions, rather than to address the challenges of a distributed setting." [33].

[1] Isolation describes the degree to which concurrent transactions are aware of each other, e.g., by accessing the same data items.

2.2 Distributed Systems

Researchers from the distributed systems community investigate state shared by multiple replicas, i.e., several copies of a datum exist which may or may not be identical. Executions of operations on these replicas may read or change the state at one or more replicas. Essentially, "a consistency criterion [or consistency model] defines which executions of a distributed system are considered correct" [21], i.e. which order of operations leaves the data in a correct state.

In the following, we will focus on the distributed systems perspective on consistency which can be defined as follows:

> A system is in a consistent state, if all replicas are identical and the ordering guarantees[2] of the specific consistency model are not violated.

3 Perspectives and Consistency Models

3.1 Perspectives on Consistency

In a distributed storage system there are two perspectives on consistency [34]: the provider (i.e., the entity responsible for the deployment and operation of a storage system) views the internal state of the system. His focus is on the synchronization processes among replicas and the ordering of operations. Hence, this perspective is called data-centric. The other perspective is the one of a client of the storage system. Here, a client refers to the process that interacts with the storage system which can be any kind of application, middleware or even software running on the end user's machine or mobile device. This client-centric perspective views the system from the outside as a black box. Hence, its focus is on the guarantees of the distributed storage system that could also be captured as part of a service level agreement (SLA). Based on these two perspectives, there are various consistency models either taking a client-centric or data-centric perspective. Still, there is a relation between those models so that some models and combinations thereof mean exactly the same thing while still bearing different names.

Both perspectives have advantages and disadvantages for the analysis of consistency guarantees – depending on the issue of interest. While data-centric consistency models do not address concrete implementations or algorithms, they certainly describe ordering properties that allow to develop a corresponding synchronization protocol. The downside is that data-centric consistency models are not really helpful to application developers. Client-centric consistency models describe the effects of such a synchronization protocol. While this is very helpful to an application developer, it ignores completely how this could be implemented, i.e., what internal synchronization protocols might deliver such a guarantee.

[2] Ordering guarantees in this context describe how requests may be reordered on different replicas.

3.2 Consistency Models and Implementations

Both data-centric and client-centric consistency guarantees have two dimensions: ordering and staleness[3]. Staleness describes how much a given replica is lagging behind, either expressed in terms of time (t-visibility) or versions (k-staleness)[4]. Again, k-staleness is a function of t-visibility and the update patterns of the application so that, for application-independent information, t-visibility suffices to characterize the staleness guarantees of a storage system. Low, bounded staleness values can often be tolerated by applications as long as the corresponding real-world events would have the same or higher staleness values without an IT system. For example, when person A wires money to person B, the account of A will be charged right away. Person B in contrast might not be credited for some time. In the EU this time window is limited to three days which is far longer than any replica synchronization protocol might take. Hence, small staleness values will often not be noticed.

Ordering on the other hand is more critical. In a setting with strict consistency, all requests must be executed on all replicas in their chronological order which is hard to implement in distributed databases due to clock synchronization issues as the replica servers might disagree on the actual chronological order of events. The standard database mechanism of locking offers poor performance levels in a distributed setting. Based on this, data-centric consistency models exist that relax certain ordering requirements while keeping those that are essential to applications. These models can be ordered by the "strictness" of their guarantees. Client-centric consistency models take a different approach: While there will almost certainly be cross effects between the models, the guarantees itself are disjunct in their promises and complement each other. We will start by describing client-centric consistency models before continuing to data-centric models and how those two are related.

Client-centric Consistency. The first model, *Monotonic Read Consistency (MRC)*, guarantees that a client that has read a version n will thereafter always read versions $\geq n$ [34,37]. This is helpful as from an application perspective data visibility might not be instantaneous but versions at least become visible in chronological order, i.e., the system never "goes backward" in time. For example, imagine person B from our bank scenario above. If this person first sees the credited amount on his bank account statement and then tries to transfer the money to a person C which fails due to "insufficient funds", this will at least cause severe customer irritation if not more.

Read Your Writes Consistency (RYWC) guarantees that a client that has written a version n will thereafter always be able to read a version that is at least as new as n, i.e., $\geq n$ [34,37]. This helps, for example, to avoid user irritation when person A checks his bank account statement, does not see the

[3] Yu and Vahdat[39] propose an additional dimension *numerical error* to describe replica differences based on the semantics of the respective data item. From our point of view, this is first not always applicable and second a numerical error is essentially a function of ordering, staleness and application access patterns.

transaction and consequently wires the same amount of money again. Generally, RYWC avoids situations where a user or application issues the same request several times because it gets the impression that the request failed the first time. For idempotent operations reissuing requests causes only additional load on the system, while reissuing other requests will create severe inconsistencies.

Monotonic Writes Consistency (MWC) guarantees that two updates by the same client will be serialized in the order that they arrive at the storage system [34,37]. This is useful to avoid seemingly lost updates when an application first writes and then updates a datum but the update is executed before the initial write and is, thus, overwritten. In the bank scenario above, person A might have corrected the account number of person B before finalizing the transfer. If MWC is not guaranteed, the money might end up in the wrong account.

Write Follows Read Consistency (WFRC) guarantees that an update following a read of version n will only execute on replicas that are at least of version n [34]. This, also, helps against seemingly lost updates where the update is overwritten by a delayed update request for versions $\leq n$. This model essentially extends MWC guarantees to updates by other clients that have at least been seen.

In NoSQL and Cloud storage systems, these client-centric properties are typically not guaranteed explicitly. Benchmarks can be used to determine the probability of violations or to measure the second dimension staleness [8,38].

Data-centric Consistency. In this section, we will present data-centric consistency models ordered by the strictness of their guarantees and discuss for each model how it can be translated into a client-centric consistency model. As already discussed, there are two consistency dimensions: staleness and ordering. The following consistency models (apart from Linearizability) do *not* consider staleness [34]. In fact, increasing strictness of ordering guarantees often leads to higher staleness values as updates may not be applied directly but are required to fulfill dependencies first (e.g.,[3]).

The lowest possible ordering guarantee is typically described as *Weak Consistency* [34,37]. As the name states, guarantees are very weak in that they do not really exist. Essentially, weak consistency translates to a colloquial "replicas might by chance become consistent". While an implementation may or may not have a protocol to synchronize replicas, a typical usecase can be found in the context of a browser cache: it is updated from time to time but replicas will rarely (if ever) be consistent. As Weak Consistency does not provide any ordering guarantees at all, there is no relation to client-centric consistency models.

Eventual Consistency (EC) is a little stricter. It requires convergence of replicas, i.e., in the absence of updates and failures the system converges towards a consistent state. Updates may be reordered in any way possible and a consistent state is simply defined as all replicas being identical [34,37]. EC is very vague in terms of concrete guarantees but is very popular for web-based services. Most NoSQL systems implement EC [16,11,26,17].

In terms of client-centric consistency guarantees, EC often fulfills these guarantees for a majority of requests but does not guarantee to do so. As an example,

Amazon S3[4] currently delivers MRC for about 95% of all requests whereas it still violated MRC in about 12% of all requests in 2011 [8].

While there are certainly some usecases where EC cannot be applied, it often suffices as the real world itself is inherently eventually consistent. The difference is, that more conflict resolution is necessary at the application layer [16] requiring a higher skill set from application developers. Instead of pessimistically locking data items "guesses and apologies" are used [22].

Causal Consistency (CC) is the strictest level of consistency that can be achieved in an always available storage system [30] based on the tradeoffs of the CAP theorem [18]. In a causally consistent storage system, all requests that have a causal relationship to another request must be serialized (i.e., executed) in the same order on all replicas while unrelated requests may be serialized in arbitrary order. A request $r2$ causally depends on a request $r1$

- if both requests are issued by the same client and $r1$ was received at the storage system before $r2$,
- if $r2$ is a read that returns the result of $r1$ which is an update or
- if there is a transitive relation between both requests [34,37,9].

Of course, CC captures potential causality so that systems like COPS [29] have to evaluate large dependency trees before applying an update. This both adds an overhead and increases staleness as updates cannot become visible right away. Bailis et al. [3] propose to minimize this impact by having the application explicitly define dependencies that need to be considered. A typical implementation uses vector clocks to identify (potential) causal dependencies.

CC can also be defined via the client-centric guarantees discussed above: If all four are fulfilled, the system is causally consistent [9]. It is also possible to create the client-side illusion of CC with the combination of version caching and vector clocks [7].

As Guerraoui and Hari point out, CC does not require replica convergence [21]. Convergence is only asserted when the latest update is causally dependent on all previous writes since the last idempotent replace-update[5] and staleness is bounded.

Sequential Consistency (SC) is a very strict consistency model and cannot be achieved in always available systems[6]. It requires that all requests are serialized in the same order on all replicas and that requests by the same client are executed in the order that they are received by the storage system [34]. While this model does not guarantee anything about the recentness of values read by clients, it mandates that all updates become visible to clients in the same order. Often,

[4] `aws.amazon.com/s3`

[5] i.e., some request like $x := 5$ which does not depend on any previous value.

[6] In CC only requests with causal dependencies must be executed in the same order on all replicas. For SC, this extends to all requests so that replicas need to agree on the ordering of requests for non-causally related requests. This is not possible in the presence of failures so that the system either becomes unavailable or violates its consistency model.

SC is described as strict consistency which is not entirely true as staleness is not addressed. But since real-world staleness values are often very small SC usually suffices even for applications seemingly requiring strict consistency.

SC could, for example, be implemented using the Paxos algorithm [27]. Generally, vector clocks that define causal relationships can be in conflict (e.g., for unrelated concurrent updates). If vector clocks are used for request ordering and an approach exists that defines a transitive, global order for all conflicting vector clocks, then a causally consistent system becomes sequentially consistent.

When focusing on client-centric consistency guarantees, the main difference between CC and SC is that WFRC becomes global in so far as reads by all clients are considered. This means that as soon as a client has seen a particular version n, all updates by other clients will only be executed on replicas that have already processed the update to version n. This guarantee can be provided as SC guarantees that all replicas execute all updates in the same order. So, once a version n has been read, it is guaranteed to have been finally serialized as that version so that any updates will be serialized with a higher version number.

Linearizability (LIN) describes what is typically meant with strict consistency. It does not only consider ordering but also staleness, i.e., it requires that all requests are ordered chronologically by their arrival time in the system and that all requests always see the effects of the preceding request. This can be visualized as all operations happening instantaneously at a single point in time and not during an interval of time [23].

LIN is hard to implement in distributed systems as there is always the issue of clock synchronization (which is necessary to determine a chronological order of requests). In practice, however, sufficiently high precision is achieved to guarantee that violations are highly improbable to occur. Furthermore, in case of violations LIN becomes SC between which applications will rarely notice a difference. While Consensus protocols can guarantee that all replicas serialize requests in the same order, they cannot guarantee that all replicas execute requests in the actual chronological order of arrival in the system. An implementation using distributed locking is likely to show poor performance.

Expressed in terms of client-centric consistency guarantees, the difference between SC and LIN is that both RYWC and MWC become global guarantees. This means that a client will always see all updates and that all writes will be executed in the (global) chronological order. MRC then also becomes global as a side effect.

Beyond the data-centric consistency models discussed here, there are a few other models (e.g., PRAM consistency [10]) which we leave out as no implementations exist and space within this paper is limited. Table 1 gives an overview of the relationship between different client-centric and data-centric consistency models. Entries "N/A" mean that the guarantee may be reached for single requests from time to time but only based on chance. In contrast, "Often" specifies that such a guarantee is fulfilled for a large number of requests. "Single Client" describes that the guarantees from section 3.2 are fulfilled, whereas we

Table 1. Relationship Between Data-centric and Client-centric Consistency Models Ordered by the Strictness of their Guarantees

Data-centric Model	MRC	RYWC	MWC	WFRC
Weak Consistency	N/A	N/A	N/A	N/A
Eventual Consistency	Often	Often	Often	Often
Causal Consistency	Single Client	Single Client	Single Client	Single Client
Sequential Consistency	Single Client	Single Client	Single Client	Global
Linearizability	Global	Global	Global	Global

use "Global" to describe when such a guarantee is extended to all clients at the same time.

Other Consistency Models. Beyond the models already discussed, there are also a few other consistency models that do not quite fit the categorization used so far.

Multi-dimensional Consistency: Yu and Vahdat [39] introduce the concept of a conit, a consistency unit, which is a three dimensional vector that describes tolerable deviations from LIN along the dimensions staleness, order error and numerical error. As already mentioned, numerical error is often not applicable and semantically overlaps with staleness and order error. When ignoring numerical error, their work becomes comparable to the work of Torres-Rojas et al.(e.g., [36,35]) who coined the term *timed consistency*. Timed consistency models are also sometimes known as delta consistency and essentially describe a combination of ordering and staleness in that the inconsistency window (defined by the time period between the commit of an update and reaching a consistent state) is bound. This means that the guarantees of a particular consistency model are not reached right away but rather after a fixed period of time Δt. If replicas fail to synchronize during that period of time, the item becomes unavailable until consistency has been reached. This is particular useful for guaranteeing Service Level Agreements (SLAs) and increases the transparency of the consistency availability trade-off.

Sadly, to our knowledge no implementations of timed consistency models exist apart from TACT [39] and the work of Krishnamurthy et al. [25] who guarantee bounds on k-staleness (based on version count). It is possible, though, to specify a *timed* version for each of the data-centric consistency models where the guarantees become visible before the specified time window is over. In that case, the models discussed above become a special case of their timed equivalent (i.e., with a time window of infinity) which also affects the timeliness of client-centric guarantees.

Coherence: In their original definition, data-centric consistency models provide ordering guarantees for all data items, i.e., in CC, for example, two updates by the same client on two different data items must be serialized in correct

order. This also implies that an eventually consistent datastore can only be in a consistent state if all replicas of all data items are identical. Depending on the size of the datastore deployment this may never be the case and it is also more difficult to coordinate updates on large numbers of servers than for just a few. So, for reasons of scalability it often makes sense to provide the guarantees of the consistency model only per key. In the case of our example above, those two updates could then be executed in arbitrary order, thus, granting more flexibility to the storage system. Guarantees per key often suffice as it is then up to the application developer to persist all items, which need guarantees amongst each other, under the same key.

Those models are named coherence, i.e., eventual coherence, causal coherence, sequential coherence. It is common practice, though, to use consistency for both coherence and consistency models alike. To add some clarity, we propose to add a "per key" prefix if coherence is meant, i.e., per key CC instead of causal coherence.

Ramakrishnan [33] argues that the "unit of consistency" should also be considered as a continuum where guarantees are not only provided either for the entire data set or for just one key but also for groups of keys like, e.g., the entity groups in Google's Megastore [6].

Adaptable Consistency: Kraska et al. [24] propose Consistency Rationing where data items are in a first step clustered based on importance (e.g., for a web shop credit card numbers vs. comments on reviews) into types A, B and C. While types A and C are always handled at LIN or EC respectively, B data continuously changes its consistency requirements based on an external cost function. This means that B data is handled at LIN whenever the costs of inconsistencies exceed the cost of opportunity caused by unavailability or high latencies. Consistency Rationing could, for example, be implemented via the much older GARF library [20].

Chihoub et al. [12,13] present approaches that allow the user to specify maximum stale read rates or a consistency cost efficiency level as part of SLAs. The system then dynamically uses different consistency levels in Apache Cassandra [26] while guaranteeing the SLAs.

Li et al. [28] propose the concept of RedBlue Consistency where operations are broken down into very small commutative suboperations that are then categorized as either red or blue meaning that they are either synchronously or asynchronously replicated while guaranteeing dependencies between suboperations. While Consistency Rationing uses different consistency levels based on the data type, RedBlue Consistency adaptively tunes the consistency level based on the kind of operation.

4 Measuring Consistency Guarantees

4.1 Continuous Consistency Metrics

According to thefreedictionary.com, a metric is "A standard of measurement". When measuring a certain aspect, a measurement always comprises a value and

a corresponding unit (e.g., for the height of a building this could be the value "5" and the unit "meter"). If it is not possible to find two values which do not have any value in between them, the metric is continuous. Otherwise the metric is discrete. An example for a continuous metric would be the height of a person, whereas clothing sizes are an example for a discrete metric.

When the ultimate goal is to compare consistency guarantees of two storage systems, it is desirable to either use a continuous metric or at least use a discrete metric with a large number of potential measurement values. Otherwise, it might not be possible to rank systems according to their consistency guarantees. In the following, we will discuss metrics for data-centric and client-centric consistency. Depending on the perspective (storage system provider or application developer), different metrics may be the best fit.

Data-centric Consistency Metrics. Zellag and Kemme [40] extend their previous work on transactional datastores to non-transactional datastores. They propose to build a global dependency graph based on operation logs and count cycles in the graph as a metric for "consistency anomalies". This is a discrete metric and one of their main assumptions is that the storage system guarantees at least CC which is very restrictive and does not allow to analyze consistency guarantees of most NoSQL systems which only offer EC. Table 2 lists their approach as "Anomalies".

Rahman et al. [32], Golab et al. [19] and Anderson et al. [2] at Hewlett Packard Labs also propose to build dependency graphs based on operation logs and to count cycles in the graph as a metric for consistency guarantees. They distinguish the three properties safeness, regularity and atomicity for which they each count violations. A storage system that has no cycles in its atomicity graph fulfills LIN. The other two properties also consider staleness as well as ordering. Regularity is, thus, stricter than SC whereas Safety cannot be compared to existing consistency models. Regularity mandates that "a read not concurrent with any writes returns the value of the most recent write, and a read concurrent with some writes returns either the value of the most recent write, or the value of one of the concurrent writes" [2]. Safeness in contrast relaxes the last requirement so that reads concurrent with writes may return arbitrary values. We do not believe that the latter guarantee is very helpful as it basically requires LIN for non-concurrent requests and Weak Consistency for concurrent requests. Chockler et al. [14] seem to share that opinion. Furthermore, real-world systems may or may not return the value of the most recent write but, to our knowledge, no system exists that may return values that have never been written. All three metrics can also be expressed as k-property or Δ-property (e.g., k-atomicity and Δ-atomicity) which describes the maximum number of time units or versions a particular system has been found to lag behind during a violation. This is a rather coarse-grained discrete metric. Table 2 lists their approach as "k-Atomicity", "Δ'-Atomicity" etc.

We propose to again distinguish the two consistency dimensions ordering and staleness and measure them separately. Staleness can be expressed either based on time (t-Visibility) or operation count (k-Staleness) [4]. We believe that these two (continuous) metrics are best suitable to describe data-centric staleness.

Table 2. Overview of Data-centric Consistency Metrics

Metric	Staleness	Ordering	Continuous	Discrete	Unit & Description
Anomalies	X	X	-	X	Number of cycles
k-Atomicity	X	X	-	X	Max. version lag in violation
Δ-Atomicity	X	X	-	X	Max. time lag in violation
k-Regularity	X	X	-	X	Max. version lag in violation
Δ-Regularity	X	X	-	X	Max. time lag in violation
k-Safeness	X	X	-	X	Max. version lag in violation
Δ-Safeness	X	X	-	X	Max. time lag in violation
t-Visibility	X	-	X	-	Prob. distr. of time lag
k-Staleness	X	-	X	-	Prob. distr. of version lag
Violations	-	X	X	-	No. of violations per time unit

It probably makes sense to aggregate them into a distribution function, i.e., a function describing the probability of a particular staleness "level". Staleness can be measured independent of concrete application workloads. For ordering on the other hand, it makes sense to mine the replicas' operation logs to determine the number of violations for each of the consistency models; i.e., in a SC system violations of LIN will be counted, in a CC system violations of SC will be measured and in an EC system violations of CC could be counted. This, obviously, highly depends on the distribution of requests regarding time, target key, originator and kind (read, insert, update, delete). Hence, for a comparison of two systems' consistency guarantees it is a hard requirement to replay exactly the same client workload which will often be problematic[7]. Ordering can then be reported as number of violations of consistency model X per unit of time. Table 2 gives an overview of data-centric consistency metrics.

Client-centric Consistency Metrics. Wada et al. [38] as well as Bermbach and Tai [8] propose to take a client-centric perspective for measuring consistency. This is of particular interest for application developers who can this way get concrete information to act upon. For client-centric consistency, there are again the two consistency dimensions ordering and staleness which both papers consider. Patil et al. [31] also propose to measure client-centric staleness in terms of time.

Staleness is best expressed either in terms of time (t-Visibility) or version lag (k-Staleness) in both cases the corresponding data-centric value is an upper bound for the client-centric one, as a system may employ additional mechanisms to hide staleness from the application. For example, in a quorum system with an (N,R,W) configuration of (5,2,2) data-centric t-Visibility will be determined by the time difference between the start of the update in replica 1 (or the commit timestamp – this depends on when updates become visible: right away or upon commit) and the end of the update in replica 5. The client-centric t-Visibility, in contrast, is determined again by the same start timestamp but

[7] This is a common problem for consistency metrics: Ordering cannot be considered properly without analysis of the request workload.

Table 3. Overview of Client-centric Consistency Metrics

Metric	Staleness	Ordering	Continuous	Discrete	Unit & Description
MRC Violations	-	X	X	-	Prob. distr. of violation
MWC Violations	-	X	X	-	Prob. distr. of violation
RYWC Violations	-	X	X	-	Prob. distr. of violation
WFRC Violations	-	X	X	-	Prob. distr. of violation
t-Visibility	X	-	X	-	Prob. distr. of time lag
k-Staleness	X	-	X	-	Prob. distr. of version lag

ends when replica 4 completes the write as afterwards no request will ever again return the old value. Hence, data-centric staleness values are an upper bound for client-centric staleness values. Staleness can either be expressed as a density function (probability distribution of inconsistency window sizes) or as a cumulative density function (probability of reading fresh data Δt time units after the last update).

Ordering is best expressed in terms of the client-centric consistency models, i.e., the likelihood of a request violating a particular guarantee. Table 3 gives an overview of client-centric consistency metrics.

4.2 Consistency Benchmarking Approaches

After identifying the metrics most useful for measuring consistency in the last section, we will now describe benchmarking approaches for these metrics.

Data-centric Consistency. All data-centric metrics require access to the actual replicas of the storage system. A test application creates load on the system. Results are then achieved by mining replica logs which should for each request contain the following information: start and end timestamp at each replica, some unique request id and the request type (read, write). Based on this, it is then possible to calculate t-Visibility, k-Staleness as well as the number of ordering violations and the corresponding consistency model[8].

Client-centric Staleness and MRC Violations. Both t-Visibility as well as k-Staleness can be benchmarked via the approach of [38] and its extension by [8]: Several geographically distributed machines interact with a storage system. A single writer periodically writes a timestamp and a version number to the storage system. The remaining machines continuously read the same data item from the storage system. Based on this the distribution of staleness (both based on time and version lag) can be calculated. The probability of MRC violations can be calculated by analyzing the results of each individual reader machine.

[8] Some additional information like the (N,R,W) configuration for a quorum system may be necessary.

MWC Violations. A single machine inserts a value into the storage system and directly updates it afterwards. After waiting for a sufficiently long period (all replicas need to synchronize) the key is read again and the result is compared to the updated value. If this is repeated for a large number of keys, the probability distribution for violations of MWC can be calculated.

RYWC Violations. A single machine writes a value into the storage system and directly starts to continuously read it afterwards and logs the time difference to the end of the update as well as whether it was possible to read the new value or not. If this is repeated a statistically significant number of times, then it is possible to calculate the probability distributions for violations of RYWC as a function of the duration since the last update.

WFRC Violations. So far, no benchmarking approach exists for WFRC violations. This can be explained by the fact that a violation cannot be directly observed by a client. One approach could be to use the replica logs of the storage system to identify if and how often WFRC has been violated.

Another approach could rely on the fact that WFRC violations mainly cause the effect that a delayed update message of an older version replaces the update that was executed on an older replica. If, for example, a client reads version $n+10$ and then issues an update which executes on a replica still at version n, then (depending on the storage system's implementation) either a delayed update message for version $n+10$ may replace the client's update (which leads to a lost update) or a conflicting version will be created which needs to be reconciled at a later point in time. If neither effect becomes visible, it still does *not* imply that WFRC is always guaranteed.

Finally, a third approach might work for storage systems which offer update operations beyond a CRUD interface. For example, a record append operation like in the Google File System [17] could be used followed by an analysis of the update order within the file.

5 Conclusion

In this work, we have provided an comprehensive overview of consistency in distributed systems. We started with a brief comparison of consistency in databases and distributed systems before focusing on the two perspectives on consistency in distributed systems. Next, we continued with a detailed discussion of data-centric and client-centric consistency models, their usecases and the relationships between those models before describing metrics and benchmarking approaches that help to determine consistency guarantees of distributed storage systems.

References

1. Abadi, D.: Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. Computer 45(2), 37–42 (2012)
2. Anderson, E., Li, X., Shah, M., Tucek, J., Wylie, J.: What consistency does your key-value store actually provide. In: HotDep (2010)
3. Bailis, P., Fekete, A., Ghodsi, A., Hellerstein, J., Stoica, I.: The potential dangers of causal consistency and an explicit solution. In: Proceedings of the Third ACM Symposium on Cloud Computing, p. 22. ACM (2012)
4. Bailis, P., Venkataraman, S., Hellerstein, J., Stoica, I.: Probabilistically bounded staleness for practical partial quorums. VLDB Endowment (2012)
5. Bailis, P.: When is "acid" acid? rarely, `http://www.bailis.org/blog/when-is-acid-acid-rarely` (accessed January 28, 2013)
6. Baker, J., Bond, C., Corbett, J., Furman, J., Khorlin, A., Larson, J., Léon, J., Li, Y., Lloyd, A., Yushprakh, V.: Megastore: providing scalable, highly available storage for interactive services. In: Proceedings of Conference on Innovative Data Systems Research
7. Bermbach, D., Kuhlenkamp, J., Derre, B., Klems, M., Tai, S.: A middleware guaranteeing client-centric consistency on top of eventually consistent datastores. In: IC2E. IEEE (2013)
8. Bermbach, D., Tai, S.: Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. In: Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, p. 1. ACM (2011)
9. Brzezinski, J., Sobaniec, C., Wawrzyniak, D.: From session causality to causal consistency. In: PDP (2004)
10. Brzeziński, J., Sobaniec, C., Wawrzyniak, D.: Session guarantees to achieve PRAM consistency of replicated shared objects. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 1–8. Springer, Heidelberg (2004)
11. Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 26(2), 1–26 (2008)
12. Chihoub, H., Ibrahim, S., Antoniu, G., Pérez, M., et al.: Consistency in the cloud: When money does matter! (2012)
13. Chihoub, H., Ibrahim, S., Antoniu, G., Pérez, M., et al.: Harmony: Towards automated self-adaptive consistency in cloud storage. In: IEEE CLUSTER (2012)
14. Chockler, G., Guerraoui, R., Keidar, I., Vukolic, M.: Reliable distributed storage. Computer 42(4), 60–67 (2009)
15. Codd, E.F.: The relational model for database management: Version 2. Addison-Wesley, Reading (1990)
16. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: Proc. SOSP (2007)
17. Ghemawat, S., Gobioff, H., Leung, S.: The Google file system. ACM SIGOPS Operating Systems Review 37(5), 29–43 (2003)
18. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 33(2), 59 (2002)
19. Golab, W., Li, X., Shah, M.: Analyzing consistency properties for fun and profit. In: Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 197–206. ACM (2011)

20. Guerraoui, R., Garbinato, B., Mazouni, K.: The garf library of dsm consistency models. In: Proceedings of the 6th Workshop on ACM SIGOPS European Workshop: Matching Operating Systems to Application Needs, pp. 51–56. ACM (1994)
21. Guerraoui, R., Hari, C.: On the consistency problem in mobile distributed computing. In: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, pp. 51–57. ACM (2002)
22. Helland, P., Campbell, D.: Building on quicksand. In: CIDR (2009)
23. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. 12(3), 463–492 (1990)
24. Kraska, T., Hentschel, M., Alonso, G., Kossmann, D.: Consistency rationing in the cloud: Pay only when it matters. In: Proceedings of the VLDB Endowment (2009)
25. Krishnamurthy, S., Sanders, W., Cukier, M.: An adaptive framework for tunable consistency and timeliness using replication. In: DSN. IEEE (2002)
26. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review 44(2), 35–40 (2010)
27. Lamport, L.: Paxos made simple. ACM SIGACT News 32(4), 18–25 (2001)
28. Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N., Rodrigues, R.: Making geo-replicated systems fast as possible, consistent when necessary. Tech. rep., Technical report, MPI-SWS (2012), `http://www.mpi-sws.org/chengli/rbTR.pdf`
29. Lloyd, W., Freedman, M., Kaminsky, M., Andersen, D.: Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In: SOSP. ACM (2011)
30. Mahajan, P., Alvisi, L., Dahlin, M.: Consistency, availability, and convergence. Technical Report TR-11-22, University of Texas at Austin (2011)
31. Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., Gibson, G., Fuchs, A., Rinaldi, B.: Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. In: SOCC. ACM (2011)
32. Rahman, M., Golab, W., AuYoung, A., Keeton, K., Wylie, J.: Toward a principled framework for benchmarking consistency. In: Proceedings of the 8th Workshop on Hot Topics in System Dependability (2012)
33. Ramakrishnan, R.: Cap and cloud data management. Computer (2012)
34. Tanenbaum, A.S., Steen, M.V.: Distributed systems: principles and paradigms, 2nd edn. Pearson, Prentice Hall, Upper Saddle River, NJ (2007)
35. Torres-Rojas, F., Ahamad, M., Raynal, M.: Timed consistency for shared distributed objects. In: Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 163–172. ACM (1999)
36. Torres-Rojas, F., Meneses, E.: Convergence through a weak consistency model: Timed causal consistency. CLEI Electronic Journal 8(2) (2005)
37. Vogels, W.: Eventually consistent. Queue 6, 14–19 (2008)
38. Wada, H., Fekete, A., Zhao, L., Lee, K., Liu, A.: Data consistency properties and the trade offs in commercial cloud storages: the consumers' perspective. In: 5th Biennial Conference on Innovative Data Systems Research, CIDR, vol. 11 (2011)
39. Yu, H., Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services. ACM TOCS (2002)
40. Zellag, K., Kemme, B.: How consistent is your cloud application? In: Proceedings of the Third ACM Symposium on Cloud Computing, p. 6. ACM (2012)

Request Complexity of VNet Topology Extraction: Dictionary-Based Attacks*

Yvonne-Anne Pignolet[1], Stefan Schmid[2], and Gilles Tredan[3]

[1] ABB Corporate Research, Switzerland
[2] Telekom Innovation Laboratories & TU Berlin, Germany
[3] CNRS-LAAS, France

Abstract. The network virtualization paradigm envisions an Internet where arbitrary virtual networks (VNets) can be specified and embedded over a shared substrate (e.g., the physical infrastructure). As VNets can be requested at short notice and for a desired time period only, the paradigm enables a flexible service deployment and an efficient resource utilization.

This paper investigates the security implications of such an architecture. We consider a simple model where an attacker seeks to extract secret information about the substrate topology, by issuing repeated VNet embedding requests. We present a general framework that exploits basic properties of the VNet embedding relation to infer the entire topology. Our framework is based on a graph motif dictionary applicable for various graph classes. Moreover, we provide upper bounds on the *request complexity*, the number of requests needed by the attacker to succeed.

1 Introduction

While network virtualization enables a flexible resource sharing, opening the infrastructure for automated virtual network (VNet) embeddings or service deployments may introduce new kinds of security threats. For example, by virtualizing its network infrastructure (e.g., the links in the aggregation or backbone network, or the computational or storage resources at the points-of-presence), an Internet Service Provider (ISP) may lose control over how its network is used. Even if the ISP manages the allocation and migration of VNet slices and services itself and only provides a very rudimentary interface to interact with customers (e.g., service or content providers), an attacker may infer information about the network topology (and state) by generating VNet requests.

This paper builds upon the model introduced in [12] and studies complexity of the *topology extraction problem*: How many VNet requests are required to infer the full topology of the infrastructure network? While algorithms for trees and cactus graphs with request complexity $O(n)$ and a lower bound for general graphs of $\Omega(n^2)$ have been shown in [12], graph classes between these extremes have not been studied.

Contribution. This paper presents a general framework to solve the topology extraction problem. We first describe necessary and sufficient conditions which facilitate the

* This project was partly funded by the Secured Virtual Cloud (SVC) project.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 190–205, 2013.

"greedy" exploration of the substrate topology (the *host graph* H) by iteratively extending the requested VNet graph (the *guest graph* G). Our framework then exploits these conditions to construct an ordered (request) *dictionary* defined over so-called *graph motifs*. We show how to apply the framework to different graph families, discuss the implications on the request complexity, and also report on a small simulation study on realistic topologies. These empirical results show that many scenarios can indeed be captured with a small dictionary, and small motifs are sufficient to infer if not the entire, then at least a significant fraction of the topology.

2 Background

This section presents our model and discusses how it compares to related work.

Model. The VNet embedding based topology extraction problem has been introduced in [12]. The formal setting consists of two entities: a *customer* (the "adversary") that issues virtual network (VNet) requests and a *provider* that performs the access control and the embedding of VNets. We model the virtual network requests as simple, undirected graphs $G = (V, E)$ (the *guest graph*) where V denotes the virtual nodes and E denotes the virtual edges connecting nodes in V. Similarly, the infrastructure network is given as an undirected graph $H = (V, E)$ (the so-called *host graph* or *substrate*) as well, where V denotes the set of substrate nodes, E is the set of substrate links, and w is a capacity function describing the available resources on a given node or edge. Without loss of generality, we assume that H is connected and that there are no parallel edges or self-loops neither in VNet requests nor in the substrate.

In this paper we assume that besides the resource demands, the VNet requests do not impose any mapping restrictions, i.e., a virtual node can be mapped to *any* substrate node, and we assume that a virtual link connecting two substrate nodes can be mapped to an entire (but single) *path* on the substrate as long as the demanded capacity is available. These assumptions are typical for virtual networks [5].

A virtual link which is mapped to more than one substrate link however can entail certain costs at the *relay nodes*, the substrate nodes which do not constitute endpoints of the virtual link and merely serve for forwarding. We model these costs with a parameter $\epsilon > 0$ (per link). Moreover, we also allow multiple virtual nodes to be mapped to the same substrate node if the node capacity allows it; we assume that if two virtual nodes are mapped to the same substrate node, the cost of a virtual link between them is zero.

Definition 1 (Embedding π, Relation $\mapsto$). *An* embedding *of a graph $A = (V_A, E_A, w_A)$ to a graph $B = (V_B, E_B, w_B)$ is a mapping $\pi : A \to B$ where every node of A is mapped to exactly one node of B, and every edge of A is mapped to a path of B. That is, π consists of a node $\pi_V : V_A \to V_B$ and an edge mapping $\pi_E : E_A \to P_B$, where P_B denotes the set of paths. We will refer to the set of virtual nodes embedded on a node $v_B \in V_B$ by $\pi_V^{-1}(v_B)$; similarly, $\pi_E^{-1}(e_B)$ describes the set of virtual links passing through $e_B \in E_B$ and $\pi_E^{-1}(v_B)$ describes the virtual links passing through $v_B \in V_B$ with v_B serving as a relay node.*

To be valid, the embedding π has to fulfill the following properties: (i) Each node $v_A \in V_A$ is mapped to exactly one node $v_B \in V_B$ (but given sufficient capacities,

v_B can host multiple nodes from V_A). (ii) Links are mapped consistently, i.e., for two nodes $v_A, v'_A \in V_A$, if $e_A = \{v_A, v'_A\} \in E_A$ then e_A is mapped to a single (possibly empty and undirected) path in B connecting nodes $\pi(v_A)$ and $\pi(v'_A)$. A link e_A cannot be split into multiple paths. (iii) The capacities of substrate nodes are not exceeded: $\forall v_B \in V_B$: $\sum_{u \in \pi_V^{-1}(v_B)} w(u) + \epsilon \cdot |\pi_E^{-1}(v_B)| \leq w(v_B)$. (iv) The capacities in E_B are respected as well, i.e., $\forall e_B \in E_B$: $\sum_{e \in \pi_E^{-1}(e_B)} w(e) \leq w(e_B)$.

If there exists such a valid embedding mapping π, we say that graph A can be embedded in B, denoted by $A \mapsto B$. Hence, $\mapsto$ denotes the VNet embedding relation.

The provider has a flexible choice where to embed a VNet as long as a valid mapping is chosen. In order to design topology discovery algorithms, we exploit the following property of the embedding relation.

Lemma 1. *The embedding relation $\mapsto$ applied to any family $\mathcal{G}$ of undirected graphs (short: $(\mathcal{G}, \mapsto)$), forms a partially ordered set (a* poset*). [Proof in full version [10]]*

We are interested in algorithms that "guess" the target topology H (the host graph) among the set $\mathcal{H}$ of possible substrate topologies. Concretely, we assume that given a VNet request G (a guest graph), the substrate provider always responds with *an honest (binary) reply* R informing the customer whether the requested VNet G is embeddedable on the substrate H. Based on this reply, the attacker may then decide to ask the provider to embed the corresponding VNet G on H, or it may not embed it and continue asking for other VNets. Let ALG be an algorithm asking a series of requests $G_1, \ldots, G_t$ to reveal H. The *request complexity* to infer the topology is measured in the number of requests t (in the worst case) until ALG issues a request G_t which is isomorphic to H and *terminates* (i.e., ALG knows that $H = G_t$ and does not issue further requests).

Related Work. Embedding VNets is an intensively studied problem and there exists a large body of literature (e.g., [7,9,13,15]), also on distributed computing approaches [8] and online algorithms [3,6]. Our work is orthogonal to this line of literature in the sense that we assume that an (arbitrary and not necessarily resource-optimal) embedding algorithm is *given*. Instead, we focus on the question of how the feedback obtained through these algorithms can be exploited, and we study the implications on the information which can be obtained about a provider's infrastructure.

Our work studies a new kind of topology inference problem. Traditionally, much graph discovery research has been conducted in the context of today's complex networks such as the Internet which have fascinated scientists for many years, and there exists a wealth of results on the topic. The classic instrument to discover Internet topologies is *traceroute* [4], but the tool has several problems which makes the problem challenging. One complication of traceroute stems from the fact that routers may appear as stars (i.e., anonymous nodes), which renders the accurate characterization of Internet topologies difficult [1,11,14]. *Network tomography* is another important field of topology discovery. In network tomography, topologies are explored using pairwise end-to-end measurements, without the cooperation of nodes along these paths. This approach is quite flexible and applicable in various contexts, e.g., in social networks. For a good discussion of this approach as well as results for a routing model along shortest and second shortest paths see [2]. For example, [2] shows that for sparse random graphs, a

relatively small number of cooperating participants is sufficient to discover a network fairly well. Both the traceroute and the network tomography problems differ from our virtual network topology discovery problem in that the exploration there is inherently *path-based* while we can ask for entire virtual graphs.

The paper closest to ours is [12]. It introduces the topology extraction model studied in this paper, and presents an asymptotically optimal algorithm for the cactus graph family (request complexity $\Theta(n)$), as well as a general algorithm (based on spanning trees) with request complexity $\Theta(n^2)$.

3 Motif-Based Dictionary Framework

The algorithms for tree and cactus graphs presented in [12] can be extended to a framework for the discovery of more general graph classes. It is based on the idea of growing sequences of subgraphs from nodes discovered so far. Intuitively, in order to describe the "knitting" of a given part of a graph, it is often sufficient to use a small set of graph *motifs*, without specifying all the details of how many substrate nodes are required to realize the motif. We start this section with the introduction of motifs and their composition and expansion. Then we present the dictionary concept, which structures motif sequences in a way that enables the efficient host graph discovery with algorithm DICT. Subsequently, we give some examples and finally provide the formal analysis of the request complexity.

3.1 Motifs: Composition and Expansion

In order to define the motif set of a graph family $\mathcal{H}$, we need the concept of *chain (graph)* C: C is just a graph $G = (\{v_1, v_2\}, \{v_1, v_2\})$ consisting of two nodes and a single link. As its edge represents a virtual link that may be embedded along entire path in the substrate network, it is called a *chain*.

Definition 2 (Motif). *Given a graph family $\mathcal{H}$, the set of motifs of $\mathcal{H}$ is defined constructively: If any member of $H \in \mathcal{H}$ has an edge cut of size one, the* chain *C is a motif for $\mathcal{H}$. All remaining motifs are at least 2-connected (i.e., any pair of nodes in a motif is connected by at least two vertex-disjoint paths). These motifs can be derived by the at least 2-connected components of any $H \in \mathcal{H}$ by repeatedly removing all nodes with degree smaller or equal than two from H (such nodes do not contribute to the knitting) and merging the incident edges, as long as all remaining cycles do not contain parallel edges. Only one instance of isomorphic motifs is kept.*

Note that the set of motifs of $\mathcal{H}$ can also be computed by iteratively by removing all low-degree nodes and subsequently determine the graphs connecting nodes constituting a vertex-cut of size one for each member $H \in \mathcal{H}$. In other words, the motif set $\mathcal{M}$ of a graph family $\mathcal{H}$ is a set of non-isomorphic minimal (in terms of number of nodes) graphs that are required to construct each member $H \in \mathcal{H}$ by taking a motif and either replacing edges with two edges connected by a node or gluing together components several times. More formally, a graph family containing all elements of $\mathcal{H}$ can be constructed by applying the following rules repeatedly.

Definition 3 (Rules). *(1) Create a new graph consisting of a motif $M \in \mathcal{M}$ (*New Motif Rule*). (2) Given a graph created by these rules, replace an edge e of H by a new node and two new edges connecting the incident nodes of e to the new node (*Insert Node Rule*). (3) Given two graphs created by these rules, attach them to each other such that they share exactly one node (*Merge Rule*).*

Being the inverse operations of the ones to determine the motif set, these rules are sufficient to compose all graphs in $\mathcal{H}$: If $\mathcal{M}$ includes all motifs of $\mathcal{H}$, it also includes all 2-connected components of H, according to Definition 2. These motifs can be glued together using the *Merge Rule*, and eventually the low-degree nodes can be added using the *Insert Node Rule*. Therefore, we have the following lemma.

Lemma 2. *Given the motifs $\mathcal{M}$ of a graph family $\mathcal{H}$, the repeated application of the rules in Definition 3 allows us to construct each member $H \in \mathcal{H}$.*

However, note that it may also be possible to use these rules to construct graphs that are *not* part of the family. The following lemma shows that when degree-two nodes are added to a motif M to form a graph G, all network elements (substrate nodes and links) are *used* when embedding M in G (i.e., $M \mapsto G$).

Lemma 3. *Let $M \in (\mathcal{M} \setminus \{C\})$ be an arbitrary two-connected motif, and let G be a graph obtained by applying the* Insert Node Rule *(Rule 2 of Definition 3) to motif M. Then, an embedding $M \mapsto G$ involves all nodes and edges in G: at least ϵ resources are used on all nodes and edges.*

Proof. Let $v \in G$. Clearly, if there exists $u \in M$ such that $v = \pi(u)$, then v's capacity is used fully. Otherwise, v was added by Rule 2. Let a, b be the two nodes of G between which Rule 2 was applied, and hence $\{\pi^{-1}(a), \pi^{-1}(b)\} \in E_M$ must be a motif edge. Observe that for these nodes' degrees it holds that $\deg(a) = \deg(\pi^{-1}(a))$ and $\deg(b) = \deg(\pi^{-1}(b))$ since Rule 2 never modifies the degree of the old nodes in the host graph G. Since links are of unit capacity, each substrate link can only be used once: at a at most $\deg(a)$ edge-disjoint paths can originate, which yields a contradiction to the degree bound, and the relaying node v has a load of ϵ. □

Lemma 3 implies that no additional nodes can be inserted to an existing embedding. In other words, a motif constitutes a "minimal reservation pattern". As we will see, our algorithm will exploit this invariant that motifs cover the entire graph knitting, and adds simple nodes (of degree 2) only in a later phase.

Corollary 1. *Let $M \in (\mathcal{M} \setminus \{C\})$ and let G be a graph obtained by applying Rule 2 of Definition 3 to motif M. Then, no additional node can be embedded on G after embedding $M \mapsto G$.*

Next, we want to *combine* motifs explore larger "knittings" of graphs. Each motif pair is glued together at a single node *or edge* ("attachment point"): We need to be able to conceptually join to motifs at edges as well because the corresponding edge of the motif can be expanded by the *Insert Node Rule* to create a node where the motifs can be joined.

Definition 4 (Motif Sequences, Subsequences, Attachment Points, $\prec$). *A* motif sequence S *is a list* $S = (M_1 a_1 a'_1 M_2 \dots M_k)$ *where* $\forall i : \ M_i \in \mathcal{M}$ *and where* M_i *is glued together at exactly one node with* M_{i-1} *(i.e.,* M_i *is "attached" to a node of motif* M_{i-1}*): the notation* $M_{i-1} a_{i-1} a'_{i-1} M_i$ *specifies the selected attachment points* a_{i-1} *and* a'_{i-1}*. If the attachment points are irrelevant, we use the notation* $S = (M_1 M_2 \dots M_k)$ *and* M_i^k *denotes an arbitrary sequence consisting of* k *instances of* M_i*. If* S *can be decomposed into* $S = S_1 S_2 S_3$*, where* S_1, S_2 *and* S_3 *are (possibly empty) motif sequences as well, then* S_1, S_2 *and* S_3 *are called* subsequences *of* S*, denoted by* $\prec$*.*

In the following, we will sometimes use the *Kleene star* notation $X^\star$ to denote a sequence of (zero or more) elements of X attached to each other.

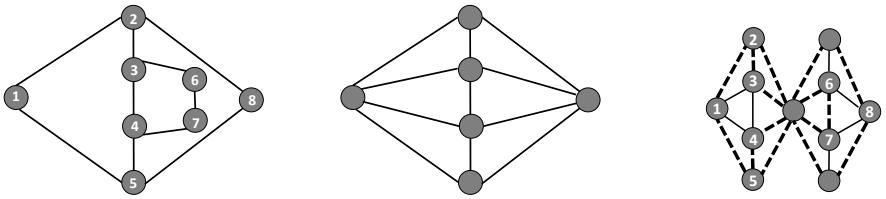


Fig. 1. *Left:* Motif A. *Center:* Motif B. Observe that $A \not\mapsto B$. *Right:* Motif A is embedded into two consecutive Motifs B. Observe that the central node has a relaying load of 4ϵ.

One has to be careful when arguing about the embedding of motif sequences, as illustrated in Figure 1 which shows a counter example for $M_i \not\mapsto M_j \Rightarrow \forall k > 0, M_i \not\mapsto M_j^k$. This means that we typically cannot just incrementally add motif occurrences to discover a certain substructure. This is the motivation for introducing the concept of a *dictionary* which imposes an order on motif sequences and their attachment points.

3.2 Dictionary Structure and Existence

In a nutshell, a dictionary is a *Directed Acyclic Graph (DAG)* defined over all possible motifs $\mathcal{M}$. and imposes an order (poset relationship $\mapsto$) on problematic motif sequences which need to be embedded one before the other (e.g., the composition depicted in Figure 1). To distinguish them from sequences, dictionary entries are called *words*.

Definition 5 (Dictionary, Words). *A* dictionary $D(V_D, E_D)$ *is a directed acyclic graph (DAG) over a set of motif sequences* V_D *together with their attachment points. In the context of the dictionary, we will call a motif sequence* word*. The links* E_D *represent the poset embedding relationship* $\mapsto$*.*

Concretely, the DAG has a single root r*, namely the chain graph* C *(with two attachment points). In general, the* attachment points *of each vertex* $v \in V_D$ *describing a word* w *define how* w *can be connected to other words. The directed edges* $E_D = (v_1, v_2)$ *represent the transitively reduced embedding poset relation with the*

chain C context: Cv_1C is embeddable in Cv_2C and there is no other word Cv_3C such that $Cv_1C \mapsto Cv_3C$, $Cv_3C \mapsto Cv_2C$ and $Cv_3C \not\mapsto Cv_1C$ holds. (The chains before and after the words are added to ensure that attachment points are "used": there is no edge between two isomorphic words with different attachment point pairs.)

We require that the dictionary be robust to composition*: For any node v, let $R_v = \{v' \in V_D, v \mapsto v'\}$ denote the "reachable" set of words in the graph and $\overline{R}_v = V_D \setminus R_i$ all other words. We require that $v \not\mapsto W, \forall W \in Q_i := \overline{R}_i^\star \setminus R_i^\star$, where the transitive closure operator $X^\star$ denotes an arbitrary sequence (including the empty sequence) of elements in X (according to their attachment points).*

See Figure 2 for an example. Informally, the robustness requirement means that the word represented by v cannot be embedded in any sequence of "smaller" words, unless a subsequence of this sequence is in the dictionary as well. As an example, in a dictionary containing motifs A and B from Figure 1 would contain vertices A, B and also BB, and a path from A to BB. In the following, we use the notation $\max_{v \in V_D}(v \mapsto S)$

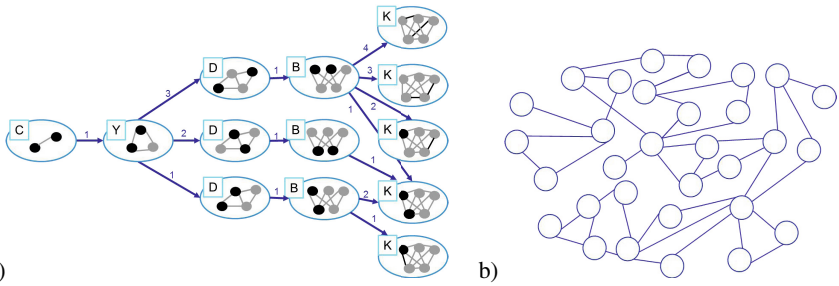


Fig. 2. a) Example dictionary with motifs Chain C, Cycle Y, Diamond D, complete bipartite graph $B = K_{2,3}$ and complete graph $K = K_5$. The attachment point pair of each word is black, the other nodes and edges of the words are grey. The edges of the dictionary are locally labeled, which is used in DICT later. b) A graph that can be constructed from the dictionary words.

to denote the set of "maximal" vertices with respect to their embeddability into S: $i \in \max_{v \in V_D}(v \mapsto S) \Leftrightarrow (i \mapsto S) \wedge (\forall j \in \Gamma^+(i), j \not\mapsto S)$, where $\Gamma^+(v)$ denotes the set of outgoing neighbors of v. Furthermore, we say that a dictionary D *covers* a motif sequence S iff S can be formed by concatenating dictionary words (henceforth denoted by $S \in D^\star$) at the specified attachment points. More generally, a dictionary covers a graph, if it can be formed by merging sequences of $D^\star$.

Let us now derive some properties of the dictionary which are crucial for a proper substrate topology discovery. First we consider maximal dictionary words which can serve as embedding "anchors" in our algorithm.

Lemma 4. *Let D be a dictionary covering a sequence S of motifs, and let $i \in \max_{v \in V_D}(v \mapsto S)$. Then i constitutes a subsequence of S, i.e., S can be decomposed to $S_1 i S_2$, and S contains no words of order at most i, i.e., $S_1, S_2 \in (\overline{R}_i \cup \{i\})^\star$.*

Proof. By contradiction assume $i \in \max_{v \in V_D}(v \mapsto S)$ and i is not a subsequence of S (written $i \not\prec S$). Since D *covers* S we have $S \in V_D^\star$ by definition.

Since D is a dictionary and $i \mapsto S$ we know that $S \notin Q_i$. Thus, $S \in D^\star \backslash Q_i$: S has a subsequence of at least one word in R_i. Thus there exists $k \in R_i$ such that $k \prec S$. If $k = i$ this implies $i \prec S$ which contradicts our assumption. Otherwise it means that $\exists j \in \Gamma^+(i)$ such that $j \mapsto k \prec S$, which contradicts the definition of $i \in \max_{v \in V_D}(v \mapsto S)$ and thus it must hold that $i \prec S$. □

The following corollary is a direct consequence of the definition of $i \in \max_{v \in V_D}(v \mapsto S)$ and Lemma 4: since for a motif sequence S with $S \in (\overline{R}_i \cup \{i\})^\star$, all the subsequences of S that contain no i are in $\overline{R}_i^\star$. As we will see, the corollary is useful to identify the motif words composing a graph sequence, from the most complex words to the least complex ones.

Corollary 2. *Let D be a dictionary covering a motif sequence S, and let $i \in \max_{v \in V_D}(v \mapsto S)$. Then S can be decomposed as a sequence $S = T_1 i T_2 i, \ldots, i T_k$ with $T_j \in Q_i, \forall j = 1, \ldots, k$.*

This corollary can be applied recursively to describe a motif sequence as a sequence of dictionary entries. Note that a dictionary always exists.

Lemma 5. *There exists a dictionary $D = (V_D, E_D)$ that covers all member graphs H of a motif graph family $\mathcal{H}$ with n vertices. [Proof in full version [10]]*

3.3 The Dictionary Algorithm

With these concepts in mind, we are ready to describe our generalized graph discovery algorithm called DICT (cf Algorithm 1). Basically, DICT always grows a request graph $G = H'$ until it is isomorphic to H (the graph to be discovered). This graph growing is performed according to the dictionary, i.e., we try to embed new motifs in the order imposed by the dictionary DAG.

DICT is based on the observation that it is very costly to discover additional edges between nodes in a 2-connected component: essentially, finding a single such edge requires testing all possibilities, which is quadratic in the component size. Thus, it is crucial to first explore the basic "knitting" of the topology, i.e., the minors which are at least 2-connected (the *motifs*). In other words, we maintain the invariant that there are never two nodes u, v which are not k-connected in the currently requested graph H' while they are k-connected in H; no path relevant for the connectivity is overlooked and needs to be found later.

Nodes and edges which are not contributing to the connectivity need not be explored at this stage yet, as they can be efficiently added later. Concretely, these additional nodes can then be discovered by (1) using an *edge expansion* (where additional degree two nodes are added along a motif edge), and by (2) adding "chains" C to the nodes (a virtual link C constitutes an edge cut of size one and can again be expanded to entire chain of nodes using *edge expansion*).

Let us specify the *topological order* in which algorithm DICT discovers the dictionary words. First, for each node v in V_D, we define an order on its outgoing edges $\{(v, w) | w \in \Gamma^+(v)\}$. This order is sometimes referred to as a "port labeling", and each path from the dictionary root (the chain C) to a node in V_D can be represented

as the sequence of port labels at each traversed node $(l_1, l_2, \ldots, l_l)$, where l_1 corresponds to a port number in C. We can simply use the lexicographic order on integers, $<^d$: $(a_1, a_2, \ldots, a_{n_1}) <^d (b_1, b_2, \ldots, b_{n_2}) \iff ((\exists\ m > 0)\ (\forall\ i < m)(a_i = b_i) \wedge (a_m < b_m)) \vee (\forall i \in \{1, \ldots n_1\}, (a_i = b_i) \wedge (n_1 < n_2))$, to associate each vertex with its minimal sequence, and sort vertices of V_D according to their embedding order. Let r be the *rank* function associating each vertex with its position in this sorting: $r : V_D \to \{1, \ldots |V_D|\}$ (i.e., r is the topological ordering of D).

The fact that subsequences can be defined recursively using a dictionary (Lemma 4 and Corollary 2) is exploited by algorithm DICT. Concretely, we apply Corollary 2 to gradually identify the words composing a graph sequence, from the most complex words to the least complex ones. This is achieved by traversing the dictionary depth-first, starting from the root C up to a maximal node: algorithm DICT tests the nodes of $\Gamma^+(v)$ in increasing port order as defined above. As a shorthand, the word $v \in V_D$ with $r(v) = i$ is written as $D[i]$; similarly $D[i] < D[j]$ holds if $r(D[i]) < r(D[j])$, a notation that will get useful to translate the fact that $D[j]$ will be detected before $D[i]$ by algorithm DICT. As a consequence, the word of a sequence S that gets matched first is uniquely identified: it is $i = \arg\max_x(D[x] \mapsto S) = \max\{r(v) | v \in \max_{v' \in V_D}(v' \mapsto S)\}$: i denotes the maximal word in S.

Algorithm DICT distinguishes whether the subsequences next to a word $v \in V_D$ are empty ($\emptyset$) or chains (C), and we will refer to the subsequence before v by BF and to the subsequence after v by AF. Concretely, while recursively exploring a sequence between two already discovered parts $T_<$ and $T_>$ we check whether the maximal word v is directly next to $T_<$ (i.e., $T_< v, \ldots, T_>$) or $T_>$ or both ($\emptyset$), or whether v is somewhere in the middle. In the latter case, we add a chain (C) to be able to find the greatest possible word in a next step.

DICT uses tuples of the form $(i, j, \text{BF}, \text{AF})$ where $i, j \in \mathbb{N}^2$ and $(\text{BF}, \text{AF}) \in \{\emptyset, C\}^2$, i.e., $D[i]$ denotes the maximal word in D, j is the number of consecutive occurrences of the corresponding word, and BF and AF represent the words before and after $D[i]$. These tuples are lexicographically ordered by the total order relation $>$ on the set of possible $(i, j, \text{BF}, \text{AF})$ tuples defined as follows: let $t = (i, j, \text{BF}, \text{AF})$ and $t' = (i', j', \text{BF}', \text{AF}')$ two such tuples. Then $t > t'$ iff $w > w'$ or $w = w' \wedge j > j'$ or $w = w' \wedge j = j' \wedge \text{BF} = C \wedge \text{BF}' = \emptyset$ or $w = w' \wedge j = j' \wedge \text{BF} = \text{BF}' \wedge \text{AF} = C \wedge \text{AF}' = \emptyset$.

With these definition we can prove that algorithm DICT is correct.

Theorem 1. *Given a dictionary for $\mathcal{H}$, algorithm* DICT *correctly discovers any $H \in \mathcal{H}$.*

Proof. We first prove that the claim is true if H forms a motif sequence (without edge expansion). Subsequently, we study the case where the motif sequence is expanded by Rule 2, and finally tackle the general composition case.

Discovery of Motif Sequences: Due to Lemma 4 it holds that for w chosen when Line 1 of *find_motif_sequence*() is executed for the first time, S is partitioned into three subsequences S_1, w and S_2. Subsequently *find_motif_sequence*() is executed on each of the subsequences $S' \in \{S_1, S_2\}$ recursively if $C \mapsto S'$, i.e., if the subsequences are not empty. Thus *find_motif_sequence*() computes a decomposition as described in Corollary 2 recursively. As each of the words used in the decomposition is a

subsequence of S and $find\_motif\_sequence()$ does not stop until no more words can be added to any subsequence, it holds that all nodes of S will be discovered eventually. In other words, $\pi^{-1}(u)$ is defined for all $u \in S$.

As a next step we assume $S' \neq S$ to be the sequence of words obtained by DICT to derive a contradiction. Since $S' := H'$ is the output of algorithm DICT and is hence embeddable in H: $S' \mapsto S$, there exists a valid embedding mapping π. Given $u, v \in V(S)$, we denote by $E^{\pi^{-1}}(S')$ the set of pairs $\{u, v\}$ for which $\{\pi^{-1}(u), \pi^{-1}(v)\} \in E(S')$. Now assume that S and S' do not lead to the same resource reservations "$\pi(S) \neq \pi(S')$". Hence there are some inconsistencies between the substrate and the output of algorithm DICT: $\Phi = \{\{u, v\} \in E(S) \backslash E^{\pi^{-1}}(S') \cup E^{\pi^{-1}}(S') \backslash E(S)\}$. With each of these "conflict" edges, one can associate the corresponding word $W_{u,v}$ (resp. $W'_{u,v}$) in S (resp. S'). If a given conflict edge spans multiple words, we only consider the words with the highest index as defined by DICT. We also define $i_{u,v} = r(W_{u,v})$ (resp. $i'_{u,v} = r(W'_{u,v})$). Since S' and S are by definition not isomorphic, $i'_{u,v} \neq i_{u,v}$.

Let $j = \max_{(u,v)\in\Phi}(i_{u,v})$ be the index of the greatest word embeddable on the substrate containing an inconsistency, and j' be the index of the corresponding word detected by DICT.

(i) Assume $j > j'$: a lower order motif was erroneously detected. Let J^+ (and J^-) be the set of dictionary entries that are detected before (after) $D[j]$ (if any) in S by DICT. Observe that the words in J^+ were perfectly detected by DICT, otherwise we are in Case (ii). We can decompose S as an alternating sequence of words of J^+ and other words using Corollary 2 : $S = T_1 J_1(a_1) T_2 \dots T_k$ with $J_i(a_i) \in (J^+)^\star$ and attachment points a_i and $T_i \in (J^-)^\star$. As the words in J^+ are the same in S', we can write $S' = T'_1 J_1 T'_2 \dots T'_k$ (using Corollary 2 as well).

Let T be the sequence among $T_1, \dots, T_k$ that contains our misdetected word $D[j]$, and T' the corresponding sequence in S'. Observe that $T' \mapsto T$ since the words J_i cut the sequences of S and S' into subsequences $T_i,\ T'_i$ that are embeddable. Observe that $D[j] \mapsto T$ since T contains it. Note that in the execution of $find\_motif\_sequence()$ when $D[j']$ was detected the higher indexed words had been detected correctly by DICT in previous executions of this subroutine. Hence, $T_<$ and $T_>$ cannot contain any words leading to edges in Φ. Thus $(j', ., ., .) < (j, ., ., .)$ which contradicts Line 1 of $find\_motif\_sequence()$.

(ii) Now assume $j' > j$: a higher order motif was erroneously detected. Using the same decomposition as step (i), we define J'^+ as the set of words perfectly detected, and therefore decompose S and S' as sequences $S = T_1 J'_1 T_2 \dots J'_{k-1} T_k$ and $S' = T'_1 J'_1 T'_2 \dots J'_{k-1} T'_k$ with $J'_i \in (J'^+)^\star$ and the property that each $T'_i \mapsto T_i$.

Let T' be the sequence among $T'_1, \dots, T'_k$ that contains our misdetected word $D[j']$, and T the corresponding sequence in S. Since $D[j'] \prec T'$, $D[j'] \mapsto T'$. Thus, since $T' \mapsto T$, we deduce $D[j'] \mapsto T$ which is a contradiction with j' and Corollary 2.

The same arguments can be applied recursively to show that conflicts in ϕ of smaller indices cannot exist either.

Expanded Motif Sequences. As a next step, we consider graphs that have been extended by applying node insertions (Rule 2) to motif sequences, so called *expanded* motif sequences: we prove that if H is an expanded motif sequence S, then algorithm DICT correctly discovers S. Given an expanded motif sequence S, replacing all

two degree nodes with an edge connecting their neighbors unless a cycle of length three would be destroyed, leads to a unique pure motif sequence T, $T \mapsto S$. For the corresponding embedding mapping π it holds that $V(S) \setminus \pi(T)$ is exactly the set $\mathcal{R}$ of removed nodes. Applying $find\_motif\_sequence()$ to an expanded motif sequence discovers this pure motif sequence T by using the nodes in $\mathcal{R}$ as relay nodes. All nodes in $\mathcal{R}$ are then discovered in $edge\_expansion()$ where the reverse operation node insertion is carried out as often as possible. It follows that each node in S is either discovered in $find\_motif\_sequence()$ if it occurs in a motif or in $edge\_expansion()$ otherwise.

Combining Expanded Sequences. Finally, it remains to combine the expanded sequences. Clearly, since motifs describe all parts of the graph which are at least 2-connected, the graph remaining after collapsing motifs cannot contain any cycles: it is a tree. However, on this graph DICT behaves like TREE, but instead of attaching chains, entire sequences are attached to different nodes. Along the unique sequence paths between two nodes, DICT fixes the largest words first, and the claim follows by the same arguments as used in the proofs for tree and cactus graphs. □

Algorithm 1. Motif Graph Discovery DICT

1: $H' := \{\{v\}, \emptyset\}$ /*current request graph*/, $\mathcal{P} := \{v\}$ /*set of unexplored nodes*/
2: **while** $\mathcal{P} \neq \emptyset$ **do**
3: choose $v \in \mathcal{P}$, $T := find\_motif\_sequence(v, \emptyset, \emptyset)$
4: **if** $(T \neq \emptyset)$ **then** $H' := H'vT$, add all nodes of T to $\mathcal{P}$, **for all** $e \in T$ **do** *edgeExpansion*(e)
5: **else** remove v from $\mathcal{P}$

find_motif_sequence$(v, T_<, T_>)$

1: find maximal i, j, BF, AF s.t. $H'v\ (T_<)\ \text{BF}\ (D[i])^j\ \text{AF}\ (T_>) \mapsto H$ where BF, AF $\in \{\emptyset, C\}^2$ /* issue requests */
2: **if** $((i, j, \text{BF}, \text{AF}) = (0, 0, C, \emptyset))$ **then return** $T_< C T_>$
3: **if** $(\text{BF} = C)$ **then** $\text{BF} = find\_motif\_sequence(v, T_<, (D[i])^j\ \text{AF}\ T_>)$
4: **if** $(\text{AF} = C)$ **then** $\text{AF} = find\_motif\_sequence(v, T_<\ \text{BF}\ (D[i])^j, T_>)$
5: **return** $\text{BF}\ (D[i])^j\ \text{AF}$

edge_expansion(e)

1: let u, v be the endpoints of edge e, remove e from H'
2: find maximal j s.t. $H'vC^ju \mapsto H$ /* issue requests */
3: $H' := H'vC^ju$, add newly discovered nodes to $\mathcal{P}$

3.4 Request Complexity

The focus of DICT is on generality rather than performance, and indeed, the resulting request complexities can often be high. However, as we will see, there are interesting graph classes which can be solved efficiently.

Let us start with a general complexity analysis. The requests issued by DICT are constructed in Line 1 of $finding\_motif\_sequence()$ and in Line 2 of $edge\_expansion()$. We will show that the request complexity of the latter is linear in the number of edges of the host graph while the request complexity of $finding\_motif\_sequence()$ depends on the structure of the dictionary. Essentially, an efficient implementation of Line 1 of $finding\_motif\_sequence$ in DICT can be seen as the depth-first exploration of the

dictionary D starting from the chain C. More precisely, at a dictionary word v requests are issued to see if one of the outgoing neighbors of v could be embedded at the position of v. As soon as one of the replies is positive, we follow the corresponding edge and continue recursively from there, until no outgoing neighbors can be embedded. Thus, the number of requests issued before we reach a vertex v can be determined easily.

Recall that DICT tests vertices of a dictionary D according to a fixed port labeling scheme. For any $v \in V_D$, let $p(C, v)$ be the set of paths from C to v (each path including C and v). In the worst case, discovering v costs $cost(v) = \max_{p \in p(C,v)}(\sum_{u \in p} |\Gamma^+(u)|)$.

Lemma 6. *The request complexity of Line 1 of* $find\_motif\_sequence(v', T_<, T_>)$ *to find the maximal* i, j, BF, AF *such that* $H'v'$ $(T_<)$ BF $(D[i])^j$ AF $(T_>) \mapsto H$ *where* BF, AF $\in \{\emptyset, C\}^2$ *and* H' *is the current request graph is* $O(\max_{v \in V_D} cost(v) + j)$.

Proof. To reach a word $v = D[i]$ in V_D with depth-first traversal there is exactly one path between the chain C and v. DICT issues a request for at most all the outgoing neighbors of the nodes this path. After v has been found, the highest j where $H'v$ $(T_<)$ BF (v^j) AF $(T_>) \mapsto H$ has to be determined. To this end, another $j + 1$ requests are necessary. Thus the maximum of $cost(v) + j$ over all word $v \in V_D$ determines the request complexity. □

When additional nodes are discovered by a positive reply to an embedding request, then the request complexity between this and the last previous positive reply can be amortized among the newly discovered nodes. Let $num\_nodes(v)$ denote the number of nodes in the motif sequence of the node v in the dictionary.

Theorem 2. *The request complexity of algorithm* DICT *is at most* $O(n \cdot \Delta + m)$*, where* m *denotes the number of edges of the inferred graph* $H \in \mathcal{H}$*, and* Δ *is the maximal ratio between the cost of discovering a word* v *in* D *and* $num\_nodes(v)$*, i.e.,* $\Delta = \max_{v \in V_D}\{cost(v)/num\_nodes(v)\}$.

Proof. Each time Line 1 of $find\_motif\_sequence()$ is called, either at least one new node is found or no other node can be embedded between the current sequences (one request is necessary for the latter result). If one or more new nodes are discovered, the request complexity can be amortized by the number of nodes found: If v is the maximal word found in Line 1 of $find\_motif\_sequence()$ then it is responsible for at most $cost(v)$ requests due to Lemma 6. If it occurs more than once at this position, only one additional request is necessary to discover even more nodes (plus one superfluous request if no more occurrences of v can be embedded there). Amortizing the request number over the number of discovered nodes results in Δ requests. All other requests are due to $edge\_expansion(e)$ where additional nodes are placed along edges. Clearly, these costs can be amortized by the number of edges in H: for each edge $e \in E(H)$, at most two embedding requests are performed (including a "superfluous" request which is needed for termination when no additional nodes can be added). □

3.5 Examples

Let us consider concrete examples to provide some intuition for Theorem 1 and Theorem 2. The execution of DICT for the graph in Figure 2.b), is illustrated in Figure 3.

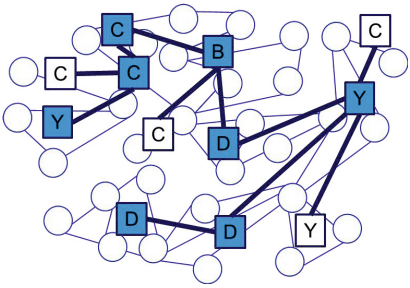


Fig. 3. Motif sequence tree of the graph in Figure 2 *b*). The squares and the edges between them depict the motif composition, the shaded squares belong to the motif sequence YC^2BDYD^2 discovered in the first execution of $find\_motif\_sequence()$ (chains, cycles, diamonds, and the complete bipartite graph over two times three nodes are denoted by C, Y, D and B respectively). Subsequently, the found edges are expanded before calling $find\_motif\_sequence()$ another four times to find Y and three times C.

A fundamental graph class are *trees*. Since, the tree does not contain any 2-connected structures, it can be described by a single motif: the chain C. Indeed, if DICT is executed with a dictionary consisting in the singleton motif set $\{C\}$, it is equivalent to a recursive version of TREE from [12] and seeks to compute maximal paths. For the cactus graph, we have two motifs, the request complexity is the same as for the algorithm described in [12].

Corollary 3. *Trees can be described by one motif (the chain C), and cactus graphs by two motifs (the chain C and the cycle Y). The request complexity of* DICT *on trees and cactus graphs is $O(n)$.*

Proof. We present the arguments for cactus graphs only, as trees constitute a subset of the cactus family. The absence of diamond graph minors implies that a cactus graph does not contain two closed faces which share a link. Thus, there can exist at most two different (not even disjoint) paths between any node pair, and the corresponding motif subgraph forms a *cycle* Y (or a triangle). Since the cycle has only one attachment point pair, Δ of D is constant. Consequently, a linear request complexity follows directly from Theorem 2 due to the planarity of cactus graphs (i.e., $m \in O(n)$). □

An example where the dictionary is efficient although the connectivity of the topology can be high are *block graphs*. A block graph is an undirected graph in which every bi-connected component (a *block*) is a clique. A *generalized block graph* is a block graph where the edges of the cliques can contain additional nodes. In other words, in the terminology of our framework, the motifs of generalized block graphs are *cliques*. For instance, cactus graphs are generalized block graphs where the maximal clique size is three.

Corollary 4. *Generalized block graphs can be described by the motif set of cliques. The request complexity of* DICT *on generalized block graphs is* $O(m)$*, where* m *denotes the number of edges in the host graph.*

Proof. The framework dictionary for generalized block graphs consists of the set of cliques, as a clique with k nodes cannot be embedded on sequences of cliques with less than k nodes. As there are three attachment point pairs for each complete graph with four or more nodes, DICT can be applied using a dictionary that contains three entries for each motif with more than three nodes ($num\_nodes() > 3$). Thus, the i^{th} dictionary entry has $\lfloor i/3 \rfloor + 3$ nodes for $i > 1$ and $cost(D[i]) < 3(i+2)$ and Δ of D is hence in $O(1)$. Consequently the complexity for generalized block graphs is $O(m)$ due to Theorem 2. □

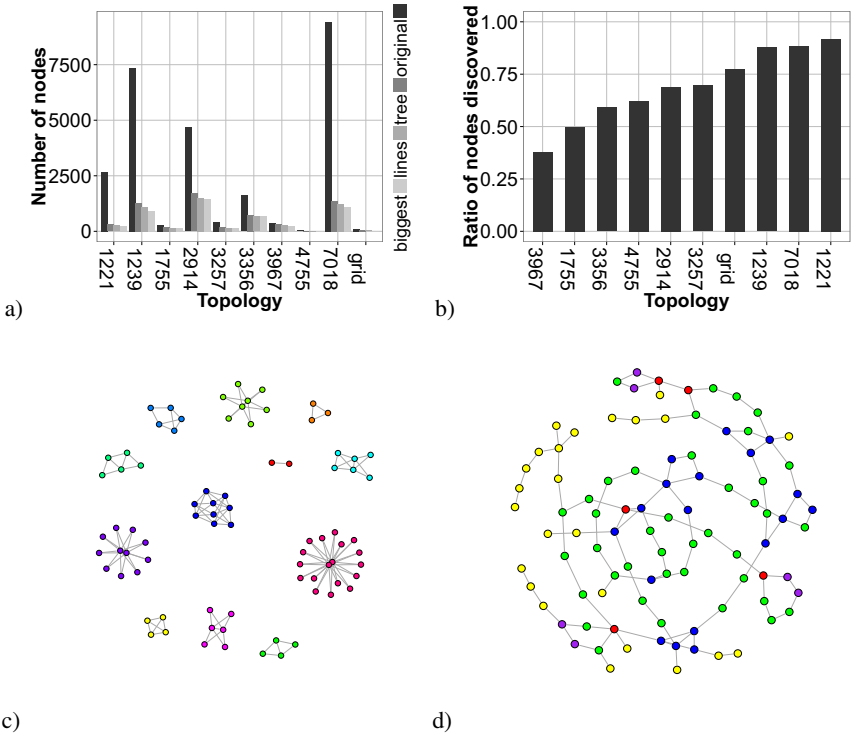


Fig. 4. Results of DICT when run on different Internet and power grid topologies. a) Number of nodes in different autonomous systems (AS). We computed the set of motifs of these graphs as described in Definition 2 and counted the number of nodes that: (i) belong to a tree structure at the fringe of the network, (ii) have degree 2 and belong to two-connected motifs, and finally (iii) are part of the largest motif. b) The fraction of nodes that can be discovered with 12-motif dictionary represented in Figure c). d) An example network where tree nodes are colored yellow, line-nodes are green, attachment point nodes are red and the remaining nodes blue.

On the other hand, Theorem 2 also states that highly connected graphs may require $\Omega(n^2)$ requests, even if the dictionary is small. In the next section, we will study whether this happens in "real world graphs".

4 Experiments

To complement our theoretical results and to validate our framework on realistic graphs, we dissected the *ISP topologies* provided by the Rocketfuel mapping engine[1]. In addition, we also dissected the topology of a European electricity distribution grid (`grid` on the legends). Figure 4 a) provides some statistics about the aforementioned topologies. Since DICT discovers both tree and degree 2 nodes in linear time, this figure shows that most of each topology can be discovered quickly. The inspected topologies are composed of a large bi-connected component (the largest motif), and some other small and simple motifs. Figure 4 b) represents the fraction of each topology that can be discovered by DICT using only a 12-motifs dictionary (see Figure 4 c)). Interestingly, this small dictionary is efficient on 10 different topologies, and contains motifs that are mostly symmetrical. This might stem from the man-engineered origin of the targeted topologies. Finally, Figure 4 d) provides an example of such a topology.

References

1. Acharya, H.B., Gouda, M.G.: On the hardness of topology inference. In: Aguilera, M.K., Yu, H., Vaidya, N.H., Srinivasan, V., Choudhury, R.R. (eds.) ICDCN 2011. LNCS, vol. 6522, pp. 251–262. Springer, Heidelberg (2011)
2. Anandkumar, A., Hassidim, A., Kelner, J.: Topology discovery of sparse random graphs with few participants. In: Proc. SIGMETRICS (2011)
3. Bansal, N., Lee, K.-W., Nagarajan, V., Zafer, M.: Minimum congestion mapping in a cloud. In: Proc. 30th PODC, pp. 267–276 (2011)
4. Cheswick, B., Burch, H., Branigan, S.: Mapping and visualizing the internet. In: Proc. USENIX Annual Technical Conference, ATEC (2000)
5. Chowdhury, M.K., Boutaba, R.: A survey of network virtualization. Elsevier Computer Networks 54(5) (2010)
6. Even, G., Medina, M., Schaffrath, G., Schmid, S.: Competitive and deterministic embeddings of virtual networks. In: Bononi, L., Datta, A.K., Devismes, S., Misra, A. (eds.) ICDCN 2012. LNCS, vol. 7129, pp. 106–121. Springer, Heidelberg (2012)
7. Fan, J., Ammar, M.H.: Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In: Proc. IEEE INFOCOM (2006)
8. Houidi, I., Louati, W., Zeghlache, D.: A distributed virtual network mapping algorithm. In: Proc. IEEE ICC (2008)
9. Lischka, J., Karl, H.: A virtual network mapping algorithm based on subgraph isomorphism detection. In: Proc. ACM SIGCOMM VISA (2009)
10. Pignolet, Y.-A., Schmid, S., Tredan, G.: Request Complexity of VNet Topology Extraction: Dictionary-Based Attacks. arXiv preprint of full version (2013)
11. Pignolet, Y.A., Schmid, S., Tredan, G.: Misleading Stars: What Cannot Be Measured in the Internet? In: Peleg, D. (ed.) DISC 2011. LNCS, vol. 6950, pp. 311–325. Springer, Heidelberg (2011)

[1] See `http://www.cs.washington.edu/research/networking/rocketfuel/`

12. Pignolet, Y.-A., Tredan, G., Schmid, S.: Adversarial VNet Embeddings: A Threat for ISPs? In: IEEE INFOCOM (2013)
13. Schaffrath, G., Schmid, S., Feldmann, A.: Optimizing long-lived cloudnets with migrations. In: Proc. IEEE/ACM UCC (2012)
14. Yao, B., Viswanathan, R., Chang, F., Waddington, D.: Topology inference in the presence of anonymous routers. In: Proc. IEEE INFOCOM, pp. 353–363 (2003)
15. Zhang, S., Qian, Z., Wu, J., Lu, S.: An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In: Proc. IEEE INFOCOM (2012)

Stability of Adversarial Routing with Feedback

Bogdan S. Chlebus[1,⋆], Vicent Cholvi[2,⋆⋆], and Dariusz R. Kowalski[3]

[1] University of Colorado Denver, Denver, Colorado, USA
[2] Universitat Jaume I, Castellón de la Plana, Spain
[3] University of Liverpool, Liverpool, United Kingdom

Abstract. We consider the impact of scheduling disciplines on the performance of routing in the framework of adversarial queuing. We propose an adversarial model which reflects stalling of packets due to transient failures and explicitly incorporates the feedback produced by the network when packets are stalled. This adversarial model provides a methodology to study stability of routing protocols when flow-control and congestion-control mechanisms affect the volume of traffic. We show that any scheduling policy that is universally stable, in the regular model of routing that additionally allows packets to have two priorities, remains stable in the proposed adversarial model.

1 Introduction

We consider routing in communication networks when transient transmission failures and congestion control mechanisms affect the number of packets handled by the nodes. Routing protocols do not act on their own but operate in an environment affected by flow control and congestion control. The overall goal is managing the rates of transmissions between pairs of nodes to provide fluency of traffic and avoiding congestive collapse, as means to optimize the use of the network. Routing protocols may drop packets, for instance when the processing time assigned to a packet has been exhausted, which has a stabilizing effect on the performance of routing. Because of the complexities of such systems, the traditional stochastic approach to investigate performance metrics encounters technical issues of mathematical nature which hinder obtaining better insights into what hinders network's performance and how to improve, see [26]. The stochastic approach usually makes strong assumptions about how traffic is generated, which may be considered unrealistic.

In this paper we consider network traffic in the framework of models that concentrate on only a few aspects of traffic generation and routing functionality. This allows to gain better understanding of traffic efficiency phenomena while avoiding making stochastic assumptions and abstracting from the low level mechanisms implemented in the network and transport layers.

⋆ The work of this author is supported by the NSF Grant 1016847.
⋆⋆ The work of this author was supported by the Spanish MEC Grant TIN2011- 28347-C02-01 and the Bancaixa Grant P11B2010-28.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 206–220, 2013.

The underlying components of the network functionality we assume are as follows. First, we want to have unbounded buffers in each node to accommodate any number of packets in transit. Second, we prune routing from any mechanisms of dropping packets in intermediate nodes, while the packets are still on their way to the assigned destinations.

These assumptions have been traditionally adopted in the models of adversarial traffic. Adversarial queuing was proposed as a methodology to analyze worst-case bounds on traffic within a traffic environment determined by parameters like injection rates and burstiness, see [5,12]. The basic aspect of a satisfactory behavior of a system in adversarial queuing is the stability of traffic, which is defined as the property that the number of packets handled simultaneously is bounded at all times.

Our goal is to extend the model of adversarial queuing by incorporating features representing congestion control. This includes the feedback provided by the network to the nodes to notify them that some packets have been stalled.

The routing protocols that we consider do not drop packets intentionally, but still some packets may be delayed due to malfunctioning of the network infrastructure. This includes transient wire-link failures or interferences on wireless links, so that packets may occasionally fail to be successfully transmitted between nodes.

Another related situation occurs when scheduled packets are delayed due to nodes being switched off for energy savings. The recently adopted IEEE 802.3az Energy Efficient Ethernet (EEE) standard, as described in [20], is expected to be conducive to energy savings in local area networks by implementing mechanisms to have nodes temporarily unavailable to cooperate in routing, see [17] for more on this issue.

Our Contribution. We propose an adversarial model to study routing in faulty systems. The adversary may learn of the failures after some time interval. Delayed feedback is more realistic than assuming that failures are known in advance when an execution of a routing protocol starts. Concerning stability, we demonstrate that any scheduling policy universally stable in the 2-priority model, as introduced by Àlvarez et al. [2], remains stable in the adversarial model that we propose.

Related Work. The adversarial methodology to study store-and-froward routing in wired networks was proposed by Andrews et al. [5] and Borodin et al [12]. Adversarial communication in wireless networks was considered by Andrews and Zhang [6]. Stability of broadcast protocols in adversarial multiple-access channels was studied by Anantharamu et al. [3,4], Bender et al. [8], and Chlebus et al. [14,15].

Adversarial models capturing failures have been proposed in the literature in various network settings. Borodin et al. [13] considered slowdowns associated with links. The papers [11,16,21,22], considered dynamic changes in the link capacities, with the intension to interpret such transient decreasing of capacity of a link as a transient failure of the link.

Álvarez et al. [2] proposed a model that allows transient disruptions of the connectivity of the system. That model was extended by Álvarez et al. [1] to incorporate node failures. The models mentioned above assume that the adversary can make a link fail at any round. The papers [1,2] and [23] assume that, at each round, the adversary knows when links fail. It is then natural to have the adversary be equipped with the power to adjust the injection of packets to such events, possibly even before link failures occur. We propose an approach in which the constraints on the adversary, in terms of the injection rate and burstiness, are modified after some time delay triggered by malfunctioning of links.

Adversarial approach has been applied to modeling malfunctioning of wireless networks, including single-hop multi channels. Bhandari and Vaidya [9,10] considered broadcast protocols in multi-hop wireless networks with nodes prone to failures. Gilbert et al. [18] considered a multi-channel where the adversary controls how information flows on subsets of channels. Meier et al. [25] considered adversarial multi-channel single-hop networks when some t channels out of m could be disrupted in a round, with m known and t not known.

Adversarial queuing was applied when studying interference and jamming in wireless networks, including single-hop multi channels and multiple access channels. Lim et al. [23] proposed an adversarial model to capture interferences among the links in wireless networks. In this case, at each round the adversary assigns specific edge rate vectors that are assumed to keep the network stable. These vectors can be interpreted as reflecting the degree in which edges fail by not providing their full capacity. Anantharamu et al. [3,4] considered multiple access channels with adversarial jamming, when the attached stations perceive jamming as colliding attempts by different stations to access the channel. Awerbuch et al. [7] studied saturation throughput of randomized protocols in adversarial multiple access channels subject to jamming. Gilbert et al. [19] studied single-hop multi-channel networks with communication subject to adversarial jamming.

For a general discussion of topics related to the mechanisms of flow and congestion control, see [24,26].

2 The Adversarial Model

We propose an adversarial approach to study stability of routing which captures packet delays due to failures of network elements. This methodology is an extension of the regular leaky-bucket adversarial model determined by injection rate and burstiness. The new component is the feedback from the network after a transmission that does not go through. This feedback restricts the adversary's capability to inject packets.

We will model networks as directed graphs $G = (V, E)$, where the vertices in V represent the nodes of the network and the edges in E are the links connecting nodes. The orientation of an edge represents the direction in which the link can transmit data. The networks we consider are *synchronous*, in that an execution of a communication protocol is partitioned into rounds.

Each packet is injected by the adversary into some node and assigned a path through the network to traverse. Such paths cannot contain the same link more

than once. If more than one packet wishes to cross an edge e in a round, then a routing protocol chooses one of these packets to send across e, while the remaining packets are kept in a queue at the tail of the edge e. When a packet reaches the destination node then it is absorbed, which means that it disappears.

A packet travels through the network with additional information associated with it, like the destination address or the round when it was injected into the network. A packet encapsulated in this information makes an atomic unit of data to be transmitted through links, which we call simply a *message*. Messages and rounds are scaled to each other, in that it takes one round to transmit a message through a link.

In this work, we consider routing environments in which a packet scheduled to be transmitted over a link in a round may fail to traverse the link, this possibly happening for a consecutive number of rounds at a time. When we use the terms faulty round an faulty link, then these refer to situations where failures in messages to traverse links occur. We assume that messages are never lost in transmissions, in that they are successfully handed over from a node to the next neighbor on the traversed path, until the packets reach their destination.

2.1 A Leaky-Bucket Regulation

We define the adversarial model by how traffic is regulated. We use a traffic descriptor using the notion of a leaky token bucket, as proposed by Turner [27]. Traffic demand is determined by packets injected into the network, each packet assigned a path to traverse. The notion of packets becoming stalled in their journeys is to represent a general malfunctioning of the system that results in a packet getting delayed, when an attempted transmission on a link that does not go through, regardless of what is the reason. A packet is *stalled* in round t if it is attempted to be transmitted but the transmission does not go through the link.

An adversary is defined by three parameters: *injection rate* r, such that $0 < r \leq 1$, *burstiness* b, which is a positive integer, and *feedback delay* δ, which is also a positive integer. These three parameters together determine the *adversarial type* (r, b, δ).

We will consider two kinds of virtual objects called tokens and antitokens. A *token* is in a bucket and represents the ability to inject a packet. An *antitoken* represents a stalled packet, and so the need to decrease the traffic by one packet to avoid congestion.

A single *bucket* is a variable K storing a number; K is initialized to 0. When $K \geq 0$, then $\lfloor K \rfloor$ is interpreted as the number of tokens in the bucket. The bucket's capacity is b. In general, K may assume negative values; in such a case the bucket does not contain any tokens. The operations performed on K are as follows.

1) In each round, r is added to K by $K \leftarrow K + r$. If at this point $K > b$ then K is modified by $K \leftarrow b$, which is interpreted as the bucket's overflow.

2) The adversary injects some i packets and simultaneously removes i tokens from the bucket by performing $K \leftarrow K - i$. For this to be possible to perform, the inequalities $0 < i \leq K$ need to hold.

3) Each stalled packet creates an antitoken carrying a value. The value of a newly created antitoken is initiated to δ. The value of an antitoken gets decremented by 1 in each round. An antitoken disappears in two possible ways, as decided by the adversary. One results in removing the antitoken and simultaneously modifying $K \leftarrow K - 1$ while the token's value is still positive. Another is when the value becomes 0, then the antitoken disappears, and simultaneously $K \leftarrow K - 1$; this represents the maximally delayed feedback.

Intuitively, when the adversary decides to annihilate a token, then this represents the moment when the adversary obtains a feedback from the network of a stalled packet. One token and one antitoken disappear simultaneously, as by annihilation resulting from their getting mixed together, which explains the terminology.

This completes the specification how a single bucket operates, when it is considered in isolation independently from other buckets. The complete picture is such that we associate a bucket K_e with each directed edge e of the network.

The operations on these buckets are coordinated as follows:

1) Every bucket gets incremented by δ in each round, subject to possible overflow which makes a bucket store precisely b tokens.

2) A packet has a path assigned to traverse; when a packet gets stalled then an antitoken is created for each bucket K_e associated with an edge e that the packet is still to traverse; all these anti tokens are said to be *related*.

3) When the adversary injects a packet to traverse a path, then it removes a token from each bucket K_e associated with any edge e of the path. For this to be possible to be performed, each bucket on the path needs to include at least one token.

4) When the adversary destroys an antitoken g created by a stalled packet p on some link, then such a destruction, and the matching operation $K \leftarrow K - 1$, is performed on each related token g' created on a link which p was still to traverse when g was created, and the bucket associated with a link that p was still to traverse when g was created.

This completes the specification of how the adversary can inject packets into the network.

To specify the adversarial model fully, we need to state precisely what the adversary controls. For instance, we interpret annihilation of a token from a bucket, and the matching removal of an antitoken by the adversary, as representing the event when the adversary obtains feedback from the network about a stalled packet. There are two possible cases here: one is to consider all the possible worst-case times when the adversary obtains the feedback, and another to assign the power to determine these times to the adversary. We may observe at this point that both these approaches are equivalent, as the adversarial model

is to capture a worst-case behavior of the system. Therefore we choose the logically simplest approach to interpret the adversary as controlling both packet injections and any malfunctioning of the network. In particular, the adversary controls which packets get stalled and when, and also when the feedback about a stalled packet is obtained, independently for each packet.

2.2 Comparison with the Regular Adversary

The regular leaky-bucket adversary is defined by two parameters: *injection rate* r, such that $0 < r \leq 1$, and a positive-integer *burstiness* b. These two parameters together determine the *adversarial type* (r, b).

Proposition 1. *A delayed-feedback leaky-bucket adversary of type* (r, b, δ) *is at least as powerful as the regular leaky-bucket adversary of the type* (r, b).

Proof. We compare the two adversarial models as regulated by a leaky bucket of tokens. When the delayed-feedback adversary of the type (r, b, δ) does not induce any stalling among the packets, then the regulatory properties of a bucket of tokens determine the regular adversary of the type (r, b). □

Corollary 1. *If a scheduling policy is unstable in a network* G *under a scheduling policy* $\mathcal{S}$ *against the regular adversary of type* (r, b) *then this same scheduling policy* $\mathcal{S}$ *is unstable in the network* G *against a delayed-feedback adversary of the type* (r, b, δ)*, for any positive integer* δ.

Proof. Consider an unstable execution of routing against the adversary of the type (r, b) when the scheduling policy $\mathcal{S}$ is applied. A similar unstable execution can be produced by the adversary of the type (r, b, δ), by Proposition 1. □

We consider the following specific scheduling policies: first-in-first-out (FIFO), nearest-to-go (NTG), farthest-from-source (FFS), and slowest-previous-link with ties broken using the nearest-from-source (SPL-NFS).

Theorem 1. *Each of the scheduling policies FIFO, NTG, FFS and SPL-NFS is unstable in some network against a delayed-feedback adversary of injection rate less than* 1.

Proof. We rely on Corollary 1 and the instability results obtained for the regular adversary. The instability of the scheduling policies FIFO, NTG, and FFS follows from the respective instabilities obtained by Andrews et al. [5], and the instability of SPL-NFS follows from the related result given by Blesa et al. [11]. □

3 Properties of the Adversarial Model with Feedback

In this section, we investigate properties of the adversarial model with feedback presented in the previous section. A key point of the model is the concept of antitoken, which represents a stalled packet and thus the need to decrease the

bound on the future traffic by one packet to avoid congestion. We re-define the adversary by expressing its power in a more analytical way to facilitate the future technical analysis. More precisely, we describe our adversary in terms of an admissibility condition, which involves a delay function accounting for the impact of each antitoken's annihilation on decreasing the amount of traffic that could be injected.

3.1 Reformulation of the Adversarial Model

In each round, the adversary may inject packets into some of the nodes in the network. In order for stability to be achievable in principle, the adversary needs to be subject to restrictions. These restrictions imposed on the regular leaky-bucket adversary are represented by its adversarial type (b, r), where $b \geq 1$ is a natural number and r satisfies $0 \leq r < 1$. The injection rate r models the rate at which a packet can be injected into the network that need to traverse the same link, for each link in the network. The burstiness b represents the maximum number of packets, that need to traverse the same edge, that the adversary can inject into the network in one round. The precise interpretation of such a type (b, r) is that in any time interval τ of length $|\tau|$ the adversary may inject at most $r|\tau| + b$ packets that need to traverse the same edge. The adversary is free to choose both the source and the destination node for any injected packet. It also determines the individual path from the source to the destination that any specific packet needs to traverse.

An extension to the adversarial model with delays is as follows. Let $I_q^\delta(t)$ represent the total number of packets which the adversary injects at round t that have queue q on their path. We say that the packet injections are *admissible for rate r and burstiness b* if the following holds for all q:

$$\sum_{t \in T} I_q^\delta(t) \leq r \sum_{t \in T} (1 - s_q^\delta(t)) + b, \tag{1}$$

where T represents a contiguous time interval and s_q^δ is a delay function. The restriction (1) is referred to as the *admissibility condition*.

Figure 1 gives an illustrative example of the admissibility condition in a queue q. This model is referred to as *feedback delayed by up to δ*, or $\mathcal{DF}_\delta$ in concise notation.

The function s_q^δ provides the rounds when the adversary becomes constrained by stalling that have occurred at queue q. We interpret such rounds as the adversary obtaining "notifications" from the network about the delay. Such notifications represent annihilations of antitokens. A formal specification of s_q^δ is provided in Definition 4 in Subsection 3.2.

At any round t, the adversary takes into account the values of $s_q^\delta(t')$, for $t' \leq t$, based only on the notifications received up to that round, as reflected in Definition 4. This means that the admissibility condition in Equation 1 is refers only to the received notifications, that is, annihilations of anti tokens, and the packets injected into the system. The way it is defined in Definition 4 is

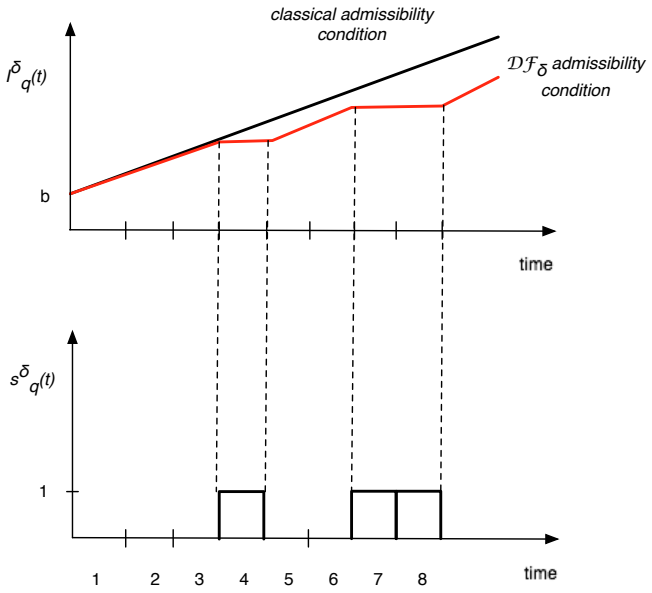


Fig. 1. An illustration for the admissibility condition (1). The number $s_q^\delta(t)$ represents the rounds when the adversary reacts to stalled packets in q. The injected data $I_q^\delta(t)$ represents the admissible amount of data that the adversary can inject due to the extra rounds incurred by stalling.

equivalent to the previously defined method of creating and annihilating antitokens, corresponding to each instance of a stalled packet.

The admissibility condition captures the following intuitions:

- When there are no annihilations of antitokens, then the adversary is allowed to inject as many packets as in the regular adversarial model.
- If some tokens get annihilated, then the adversary is allowed to inject a number of packets as determined by the traditional adversarial model, decreased by the number of annihilations as specified in the leaky-bucket regulation mechanism by way of tokens and antitokens.

The definition of stability in $\mathcal{DF}_\delta$ is similar to the definition stated under other adversarial models.

Definition 1. *Let G be a network, $\mathcal{P}$ a scheduling policy and $\mathcal{A}$ an adversary of type (r, b). Let $\mathcal{D}$ be and execution of protocol $\mathcal{P}$ against $\mathcal{A}$ in G. For a positive integer t, let $Q_{\mathcal{D}}(t)$ be the number of packets simultaneously queued in the system at time t. Protocol $\mathcal{P}$ is* stable *on G against $\mathcal{A}$ if in each such an execution $\mathcal{D}$, all the numbers $Q_{\mathcal{D}}(t)$ are bounded. Protocol $\mathcal{P}$ is* universally stable *if it is stable against any adversary of injection rate $r < 1$ and in any network.*

3.2 Delay Functions and Reactive Functions

We say that a queue q is *stalled* in round t if some packet in the queue is stalled in this round. Next we introduce the function w_q which represents the rounds where delays occur at queue q.

Definition 2. *Consider an execution of a system up to round t. Given a queue q, we define the function $w_q(t)$ such that $w_q(t) = 1$ if the queue q is stalled at round t and $w_q(t) = 0$ otherwise.*

The rounds that are added to a stalled packet's itinerary occur as a side effect of the adversary's actions. The adversary receives information about the extra rounds of stalled-packets occurring at the different queues. We consider the case where the adversary becomes constrained by the stalled packets after some time delay; the parameter δ is used to bound such a maximum delay.

Next we introduce the notations T_q, D_q^δ, and w_q^δ. We want w_q^δ to model the rounds where the adversary becomes constrained by the queue q getting stalled and it also provides the number of the notifications.

Definition 3. *We will use the following terminology and notations:*

1. *Let T_q be the set of these rounds t for which $w_q(t) = 1$ holds.*
2. *For a given function w_q, let the function $D_q^\delta : T_q \to \mathbb{N}$ be such that $D_q^\delta(t) = t'$, for $t \in T_q$ and $t \leq t' \leq t + \delta$, where t' is the time when the feedback about the queue q being staled at time t arrives.*
3. *Let $T_q^\delta(t)$ be the subset of T_q such that $t' \in T_q^\delta(t)$ if $D_q^\delta(t') = t$.*
4. *For a given w_q and a given D_q^δ, let the* delay function *w_q^δ by determined the equality $w_q^\delta(t) = |T_q^\delta(t)|$.*

Figure 2 provides a graphical representation of the notions introduced in Definition 3 in a specific example.

Reactive functions. When the adversary receives a notification of a queue q getting delayed, then this indicates that some packet in this queue will need an extra round. In order to maintain stability, the adversary must react to take into account such eventuality. Such a reaction is performed as if temporarily the injection rate were reduced, which is implemented by the mechanism of tokens and antitokens. Several notifications could be received at the same time, so the adversary needs to take that fact into account. For instance, if at some time t the adversary receives three notifications of stalling for some given link, it will reduce the long term injection rate of a packet that will cross such a link for three rounds after time t, provided such rounds have not been already reduced because of previous notifications, in which case the next "available" rounds will be chosen.

To formalize this, we define the function s_q^δ, which is intended to model the rounds when the adversary will react to a delayed notification of stalling which has occurred at a queue q.

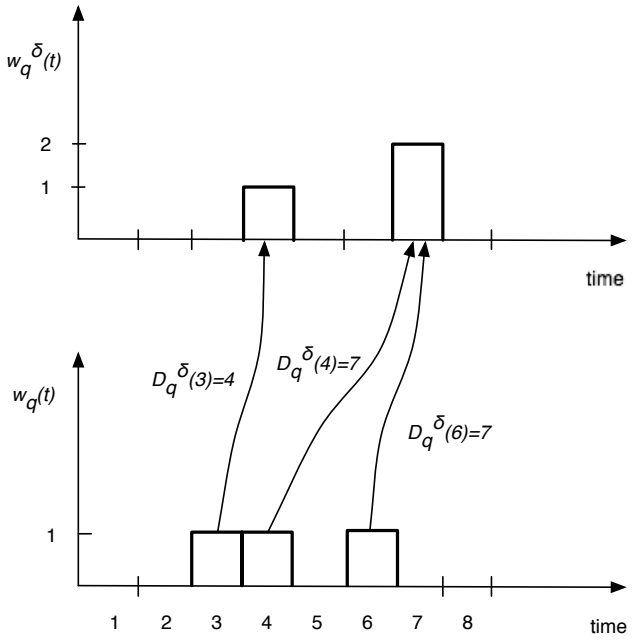


Fig. 2. An examples of how the function D_q^δ is determined

Definition 4. *For a given delayed function* w_q^δ*, we construct* the reactive function s_q^δ *as follows:*

init:
$s_q^\delta(t) \leftarrow 0$ *for all* t*;*
$t \leftarrow 1$*;* % time in w_q^δ
$t' \leftarrow 1$*;* % time in s_q^δ

Repeat forever
if $w_q^\delta(t) \neq 0$ **then** % when one or more notifications are received at time t
$t' \leftarrow \max(t, t')$*;* % set up the next "available" round in s_q^δ after t
for $t_{aux} = t'$ **to** $(t' + w_q^\delta(t) - 1)$ **do**
$s_q^\delta(t_{aux}) \leftarrow 1$*;* % mark t_{aux} as "reactive"
$t' \leftarrow t' + w_q^\delta(t) - 1$*;* % set up the next "available" round in s_q^δ
$t \leftarrow t + 1$*;* % increase time

This function s_q^δ is determined by w_q^δ, which gives the rounds when the adversary receives notifications of delays occurring at queue q, as represented by annihilations of antitokens.

4 The Stability of Scheduling Policies

In this section, we show that some scheduling policies are stable in the adversarial-queuing model $\mathcal{DF}_\delta$. We will use an auxiliary model, known as the *priority model*, which was introduced in [2]. We refer to the model as *c-priority model* when there are c priorities. It is obtained by modifying the regular adversarial model [5,12] so that packets have priorities in the following sense. If, at a certain time, more than one packet located at the same queue is ready to be transmitted, then the scheduling policy chooses the packet of the highest priority.

The following fact provides a relationship between the number of extra rounds due to delays and the number of reactive rounds at a given time interval.

Lemma 1. $\sum_{t\in T} w_q(t) \leq \sum_{t\in T} s_q^\delta(t) + \delta$.

Proof. By the definition of function w_q^δ, for each round t where $w_q(t) = 1$, there is a round t', which is not necessarily different for each t, such that $w_q^\delta(t') \neq 0$ and $t \leq t' \leq t + \delta$.

From the mechanism used to construct s_q^δ, see Definition 4, the function s_q^δ takes the value 1 as many times as the values taken by the function w_q^δ. It follows that the number of times where w_q takes the value 1 is the same as the number of times where s_q^δ takes the value 1, although not necessarily at the same rounds.

This means that there exists a bijective increasing function rD_q^δ such that, for all t, when $w_q(t) = 1$ then $rD_q^\delta(t) = t'$, where $s_q^\delta(t') = 1$.

We proceed with considering the following two cases.

The case of $t > t'$:

This cannot happen by the construction of s_q^δ, because t_{aux} in Definition 4 is always at least t.

The case of $t' > t + \delta$:

We prove this case by contradiction. Let t_1 be the first round such that $rD_q^\delta(t_1) = t_1^*$ and $t_1^* > t_1 + \delta$. From the construction of rD_q^δ described above, see Definition 4, we have that, for all $t'' \in [D_q^\delta(t_1), rD_q^\delta(t_1) - 1]$ the equality $s_q^\delta(t'') = 1$ holds. Since rD_q^δ is a bijective increasing function, there must exist some round t_m such that $t_m < t_1$ and $rD^\delta(t_m) = t_1^* - 1$. Let $t_m^* = t_1^* - 1$.

The following two facts hold. On one hand, we have that $t_m < t_1$ and $t_m^* = t_1^* - 1$, which means that $t_m^* > t_m + \delta$. On the other hand, we also have that $rD^\delta(t_m) = t_m^*$. All this contradicts our assumption that t_1 is the first round such that $rD_q^\delta(t_1) = t_1^*$ and $t_1^* > t_1 + \delta$. The fact that the inequality $t \leq t' \leq t + \delta$ holds means that for each t such that $w_q(t) = 1$, the corresponding image in s_q^δ will be for a round t' that is delayed by at most δ rounds. We conclude that the inequality $\sum_{t\in T} w_q(t) \leq \sum_{t\in T} s_q^\delta(t) + \delta$ holds true. □

The following Lemma 2 shows that if a given scheduling policy is unstable in $\mathcal{DF}_\delta$ then it is also unstable in the 2-priority model.

Lemma 2. *If a given scheduling policy is unstable against an adversary with injection rate r and burstiness b in $\mathcal{DF}_\delta$, then such a scheduling policy is unstable against some adversary of injection rate r' and burstiness b' in the 2-priority model, where $0 < r' < 1$.*

Proof. Let us take an adversary $\mathcal{A}$ in $\mathcal{DF}_\delta$ with parameters (r, b). Then, according to the admissibility condition in Equation 1 and by Lemma 1, the following estimates hold.

$$\begin{aligned}\sum_{t\in T} I_q^\delta(t) &\le r\sum_{t\in T}(1 - s_q^\delta(t)) + b\\ &\le r\sum_{t\in T} 1 - (\sum_{t\in T} w_q(t) - \delta)) + b\\ &\le r\sum_{t\in T}(1 - w_q(t)) + r\delta + b\ .\end{aligned}$$

The right-hand side of this bound equals $r\sum_{t\in T}(1 - w_q(t)) + b'$ for $b' = r\delta + b$. We obtain by algebraic manipulations that

$$\begin{aligned}\sum_{t\in T} I_q^\delta(t) + \sum_{t\in T} w_q(t) &\le r\sum_{t\in T}(1 - w_q(t)) + b' + \sum_{t\in T} w_q(t)\\ &= r\sum_{t\in T} 1 + (1 - r)\sum_{t\in T} w_q(t) + b'\\ &\le r\sum_{t\in T} 1 + ((1 - r)\sum_{t\in T}\frac{\tau}{\tau + 1}) + b'\\ &= (\frac{r + \tau}{\tau + 1})\sum_{t\in T} 1 + b'\\ &= (\frac{r + \tau}{\tau + 1}) \mid T \mid + b'\\ &= r' \mid T \mid + b'\ ,\end{aligned}$$

where we used $w_q(t) \le \frac{\tau}{\tau+1}$ and $r' = \frac{r+\tau}{\tau+1}$. Observe that $0 < r' < 1$.

Consider an adversarial pattern for injection rate r and burstiness b that results in an unstable execution of the given scheduling policy. Based on the obtained estimate on

$$\sum_{t\in T} I_q^\delta(t) + \sum_{t\in T} w_q(t)\ ,$$

we define the corresponding adversarial pattern in the 2-priority model, as specified in the claim of the lemma. The constructed specific adversarial behavior follows the same injection pattern as defined by the adversary $\mathcal{A}$ in $\mathcal{DF}_\delta$, with a low priority given to all these packets, and additionally it injects a high priority packet at the starting queue q in each round t such that $w_q(t) = 1$.

Consider the execution of the original scheduling policy under the defined adversarial pattern in the 2-priority adversarial model. By the inequality

$$\sum_{t\in T} I_q^{\delta}(t) + \sum_{t\in T} w_q(t) \leq r'|T| + \ b' \ ,$$

which holds in the execution in $\mathcal{DF}_{\delta}$, we obtain that queue-congestion of the injected packets, whether of high or of low priority, is constrained by the injection rate r', with $r' < 1$, and burstiness b'_v. This is because $\sum_{t\in T} I_q^{\delta}(t)$ from the original execution in $\mathcal{DF}_{\delta}$ corresponds to the node congestion of the low-priority packets, and $\sum_{t\in T} w_q(t)$ corresponds to the queue-congestion of the high-priority packets, both in the 2-priority execution.

It remains to argue that the newly defined execution is also unstable in the 2-priority model. The following invariant holds.

> There is at most one high-priority packet at a queue in any round in the latter execution, and the transmissions of low-priority packets are the same in both considered executions.

This follows by induction on the round numbers, because any such a packet is injected in the beginning of each round when a extra round due to stalling occurs in the execution in $\mathcal{DF}_{\delta}$ through some queue. Since the execution in $\mathcal{DF}_{\delta}$ results in unbounded queues, the other one also does. □

Theorem 2. *Any scheduling policy that is universally stable in the* 2*-priority model is universally stable in* $\mathcal{DF}_{\delta}$*.*

Proof. We suppose it is otherwise in order to arrive at a contradiction. This means that there is a scheduling policy S that is universal in the 2-priority model but for any burstiness b there is some rate r such that the inequalities $0 < r < 1$ hold and such that S is unstable against the adversary with rate r and burstiness b in $\mathcal{DF}_{\delta}$. Let us consider such an unstable execution. By Lemma 2, there is an unstable execution of the scheduling policy S in the 2-priority model, for some injection rate r' such that $0 < r' < 1$ and for burstiness b'.

This contradicts the universal stability of S in the 2-priority model. □

Àlvarez et al. [2] showed that FTG (farthest-to-go), NFS (nearest-from-source) and SIS (shortest-in-system) are universally stable for the 2-priority model. By this and Theorem 2 we obtain the following corollary.

Corollary 2. *Scheduling policies FTG, NFS and SIS are all universally stable in the adversarial model* $\mathcal{DF}_{\delta}$*.*

5 Conclusion

We study routing in the suitable adversarial frameworks. We investigate how unexpected packet delays may affect routing's performance. Packet delays represent either malfunctioning of the network's infrastructure, implemented below the network layer, or transient unavailability of nodes due to energy saving policies.

We assume that routing protocols are embedded into flow control and connection control mechanisms. These mechanisms react to packet delays by spreading the suitable information through the network with the goal to decrease packet injection rates.

We propose how to study stability of various classes of scheduling policies in such network settings. To this end, we propose a new adversarial model that has delays built into its machinery. The model we consider is an extension of the regular leaky-bucket model, which is determined only by the injection rate and burstiness.

Each transmission that fails to go through results in a feedback, which abstracts the flow control and connection control mechanisms. We treat this feedback as if given to the adversary, because it decreases the adversary's capability to inject packets.

We demonstrated that all scheduling policies stable in the 2-priority wireline adversarial model are also stable in the new proposed model. That includes such popular scheduling policies as FTG, NFS and SIS.

References

1. Àlvarez, C., Blesa, M., Serna, M.: The robustness of stability under link and node failures. Theoretical Computer Science 412(50), 6855–6878 (2011)
2. Àlvarez, C., Blesa, M.J., Díaz, J., Serna, M.J., Fernández, A.: Adversarial models for priority-based networks. Networks 45(1), 23–35 (2005)
3. Anantharamu, L., Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Deterministic broadcast on multiple access channels. In: Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM), pp. 1–5 (2010)
4. Anantharamu, L., Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Medium access control for adversarial channels with jamming. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 89–100. Springer, Heidelberg (2011)
5. Andrews, M., Awerbuch, B., Fernández, A., Leighton, T., Liu, Z., Kleinberg, J.: Universal-stability results and performance bounds for greedy contention-resolution protocols. Journal of the ACM 48(1), 39–69 (2001)
6. Andrews, M., Zhang, L.: Routing and scheduling in multihop wireless networks with time-varying channels. ACM Transactions on Algorithms 3(3), 33 (2007)
7. Awerbuch, B., Richa, A.W., Scheideler, C.: A jamming-resistant MAC protocol for single-hop wireless networks. In: Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC), pp. 45–54 (2008)
8. Bender, M.A., Farach-Colton, M., He, S., Kuszmaul, B.C., Leiserson, C.E.: Adversarial contention resolution for simple channels. In: Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms (SPAA), pp. 325–332 (2005)
9. Bhandari, V., Vaidya, N.H.: Reliable broadcast in wireless networks with probabilistic failures. In: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM), pp. 715–723 (2007)
10. Bhandari, V., Vaidya, N.H.: Reliable broadcast in radio networks with locally bounded failures. IEEE Transactions on Parallel and Distributed Systems 21(6), 801–811 (2010)
11. Blesa, M.J., Calzada, D., Fernández, A., López, L., Martínez, A.L., Santos, A., Serna, M.J., Thraves, C.: Adversarial queueing model for continuous network dynamics. Theory of Computing Systems 44(3), 304–331 (2009)

12. Borodin, A., Kleinberg, J., Raghavan, P., Sudan, M., Williamson, D.: Adversarial queueing theory. Journal of the ACM 48(1), 13–38 (2001)
13. Borodin, A., Ostrovsky, R., Rabani, Y.: Stability preserving transformations: Packet routing networks with edge capacities and speeds. Journal of Interconnection Networks 5(1), 1–12 (2004)
14. Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Maximum throughput of multiple access channels in adversarial environments. Distributed Computing 22(2), 93–116 (2009)
15. Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Adversarial queuing on the multiple access channel. ACM Transactions on Algorithms 5, 5:1–5:31 (2012)
16. Cholvi, V.: Stability bounds in networks with dynamic link capacities. Information Processing Letters 109(2), 151–154 (2008)
17. Christensen, K., Reviriego, P., Nordman, B., Bennett, M., Mostowfi, M., Maestro, J.A.: IEEE 802.3az: The road to energy efficient Ethernet. IEEE Communications Magazine 48(11), 50–56 (2010)
18. Gilbert, S., Guerraoui, R., Kowalski, D.R., Newport, C.: Interference-resilient information exchange. In: Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM), pp. 2249–2257 (2009)
19. Gilbert, S., Guerraoui, R., Newport, C.C.: Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. Theoretical Computer Science 410(6-7), 546–569 (2009)
20. IEEE P802.3az Energy Efficient Ethernet. Task force public area (2008), `http://grouper.ieee.org/groups/802/3/az/public/index.html`
21. Koukopoulos, D., Mavronicolas, M., Spirakis, P.G.: The increase of the instability of networks due to quasi-static link capacities. Theoretical Computer Sciience 381(1-3), 44–56 (2007)
22. Koukopoulos, D., Mavronicolas, M., Spirakis, P.G.: Performance and stability bounds for dynamic networks. Journal of Parallel and Distributed Computing 67(4), 386–399 (2007)
23. Lim, S., Jung, K., Andrews, M.: Stability of the max-weight protocol in adversarial wireless networks. In: Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM), pp. 1251–1259 (2012)
24. Mamatas, L., Harks, T., Tsaoussidis, V.: Approaches to congestion control in packet networks. Journal of Internet Engineering 1(1), 22–33 (2007)
25. Meier, D., Pignolet, Y.A., Schmid, S., Wattenhofer, R.: Speed dating despite jammers. In: Krishnamachari, B., Suri, S., Heinzelman, W., Mitra, U. (eds.) DCOSS 2009. LNCS, vol. 5516, pp. 1–14. Springer, Heidelberg (2009)
26. Srikant, R.: The Mathematics of Internet Congestion Control. Birkhäuser (2004)
27. Turner, J.S.: New directions in communications (or which way to the information age?). IEEE Communications Magazine 40(5), 50–57 (2002)

A Distributed Protocol for Privacy Preserving Aggregation

Yahya Benkaouz and Mohammed Erradi

Networking and Distributed Systems Research Group, TIES, SIME Lab, ENSIAS, Mohammed V-Souissi University, Rabat, Morocco
y.benkaouz@um5s.net.ma, erradi@ensias.ma

Abstract. Techniques that combine and analyze data collected from multiple partners are very useful for distributed collaborative applications. Such collaborative computations could occur between trusted partners, between partially trusted partners, or between competitors. Therefore preserving privacy is an important issue in this context. This paper presents a distributed protocol for privacy-preserving aggregation to enable computing a class of aggregation functions that can be expressed as Abelian group. The aim is to ensure participants privacy such that their inputs are not disclosed to any other entity be it trusted or not. The proposed protocol is based on an overlay structure that enables secret sharing without the need of any central authority or heavyweight cryptography.

Keywords: Privacy, Aggregation Protocol, Security, Distributed polling application.

1 Introduction

With the continuous increase of collaborative computation, the subject of data aggregation becomes increasingly important. Techniques that combine and analyze data collected from multiple partners are very useful for collaborative applications. Collaborative tasks and computations are often conducted based on data supplied by the collaborative users. Such computations could occur between trusted partners, between partially trusted partners, or between competitors. Hence, privacy has become one of the top priorities and requirements of many distributed collaborative applications.

Conducting a computation based on private inputs supplied by two or more parties, without disclosing the private inputs, is referred to Secure Multiparty Computation (SMC) in the literature. The problem is how to conduct such computation while preserving the privacy of the inputs. Secure Multiparty Computation techniques have been long studied within the field of cryptography. Generally speaking, a secure multiparty computation problem deals with computing any probabilistic function on any input in a distributed network where each participant holds one of the inputs [17]. In theory, the general secure multiparty computation problem is solvable. SMC techniques are generally secure,

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 221–232, 2013.

even if several parties (i.e. up to a fraction of the parties involved in the computation) collude to break the privacy of the other parties [5, 7, 8]. But as pointed out in [8], using the solutions derived by these general results for special cases of multiparty computation can be impractical; special solutions should be developed for special cases for efficiency reasons [6]. This is due to the fact that, the traditional setting of multiparty computation assume a number of parties which is relatively small and the network is assumed to be full mesh and the proposed techniques can be prohibitively expensive to execute [18].

In this paper, we focus on a particularly useful tool in collaborative applications which is aggregation functions. Aggregation functions aim to compute a global aggregate that depends on values held by participants in a distributed environment. Since each participant holds a small part of the global result, interaction between participants is required to compute such functions. Aggregation functions could be simple functions such as addition, multiplication, disjunction, min/max or more sophisticated functions such as functions that manipulate distributed databases, elections...and so on. In SMC, the involved parties want to compute a function $f(x_1, x_2, ..., x_n) = \{y_1, y_2, ..., y_n\}$, where x_i is the input of the participant i, and the result y_i is returned to the participant i only. Thus, each participant i knows nothing but his own result y_i [19]. Whereas in aggregation function, the computed function output a single value $f(x_1, x_2, ..., x_n) = y$. And it is known by each participant.

Current privacy-preserving data aggregation solutions come with cryptographic primitives to ensure privacy. Thus, these solutions involve a significant complexity cost. In this paper, we present a distributed protocol to compute aggregation function without the need of any central authority or heavyweight cryptography. In the proposed protocol, we consider a set of M participants $P = \{p_1, p_2, ..., p_m\}$, involved in an aggregation function. The output of the protocol consists in the aggregation of values supplied by the set of users. We consider a fully decentralized scheme where there is neither central trusted server nor an entity with a specific role. Hence, all participants are involved in the computation of the aggregation result. We aim to ensure the privacy of the inputs supplied by participants in such a way that no participant should learn anything about a given honest participant input.

Our contribution consists in DiPA, a Distributed protocol for Privacy preserving Aggregation that allows a set of partners to compute a class of aggregation functions that derive from *Abelian Group* without revealing the partners inputs. The suggested protocol is based on an overlay construction. The communication cost of our protocol is $O(r.k + N_i)$ where r is the number of group in the overlay and N_i is the number of participant per group. As we will show in the rest of this paper, our protocol has the following advantages: (1) It preserves data privacy such that participant data is only known to their owner with a given probability; (2) The aggregation result is computed by participants themselves without interacting with a specific aggregator; (3) The aggregation result is accurate when there is no data loss. Moreover, among the advantages of our approach: It does

neither rely on heavyweight cryptographic techniques nor on trustworthiness of a third party.

The remainder of this paper is organized as follows. Section 2 presents related works. We define the problem of distributed aggregation in Section 3. The system model is described in Section 4. Section 5 describes the suggested protocol and discusses its correctness. A distributed polling application and its privacy analysis are given in Section 6, and we conclude in Section 7.

2 Related Work

Different data aggregation protocols have been proposed for distributed systems and different applications. In order to ensure the privacy requirement, different techniques have been used. [11], [16] consider the data aggregation concern, and propose schemes based on applied cryptographic techniques and perturbation algorithms to add noise to the input data. Authors of [13] present an implementation of a secure system for double auction based on encryption and Shamir secret sharing scheme. In [15], the authors proposed a framework for enabling secure multiparty numerical computations in a Peer-to-Peer network. The proposed solution combines different cryptographic techniques such as Random Perturbation, Homomorphic Encryption and Shamir secret sharing. The cryptographic based protocols help to ensure data privacy. However they require a big amount of computational cost. [14] adds an overlay based on a cluster setting in order to decrease the computational cost of the cryptographic solution. Therefore most of the above solutions require an interaction with a central authority, trusted or not, which is responsible for the computation, or at least for managing public/private keys pairs.

One of the most important applications of distributed data aggregation is distributed voting. Several works have been done to ensure the privacy and the anonymity of voting participants. Authors of [7] describe techniques to ensure unconditional privacy. However, their approaches rely on a number of trusted parties. They assume that these third parties do not collude. Authors of [8] present a multi-authority secret ballot scheme that guarantees privacy, and robustness. [9] presents an electronic voting scheme based on a distributed version of Paillier cryptosystem. Hence, the majority of existing works related to decentralized voting protocols are based on cryptographic primitives, trusted parties, or assign specific role to some entities [10], [12].

3 Problem Definition

We consider a set of M participants $P = \{p_1, p_2, ..., p_m\}$, involved in an aggregation computation (e.g. a set of organizations wish to launch an analysis process on their own raw private inputs). Each participant stores locally its own initial value. We consider a fully decentralized scheme where there is neither central trusted server nor an entity with a specific role. Hence, all participants are aggregators who wish to compute aggregate functions over the set of inputs. The participants combine all the inputs to produce the final result.

An aggregation function is defined as any function whose input is a set or multi-set of values and outputs a single value (e.g. the sum or maximum/minimum of the set of inputs). In the case where the inputs are gathered from different parties, the aggregation function is called a distributed aggregation. The problem of distributed aggregation is defined as follows: Given a set of inputs which is arbitrarily distributed among the participants of a distributed system, the aim is to apply a specific function over the set of inputs and outputs a single value.

Definition: An aggregation function is a triplet (f, S, G), where S and G are two arbitrary sets, f is a composition law $f : S^* \rightarrow G$.

In this paper, we are only concerned with a subset of aggregate functions where S is a subset of G (i.e. $S \subseteq G$) and (G, f) is an Abelian group. For example, in distributed storage systems, in peer to peer settings; it may be useful to query for availability of a file and to compute the number of copies of a given file. This aggregation function may be modeled as $(sum, \mathbb{N}, \mathbb{Z})$. A more concrete example consists in binary poll which can be represented as the distributed aggregation function (f, S, G) where f is the function sum, S is the set $\{-1, +1\}$ and $G = \mathbb{Z}$.

Our approach may also be useful for aggregating data collected by different sensor networks for different use cases. Sensor networks are being widely deployed to monitor the safety of buildings, measure traffic flows, or track environmental pollutants [11-13]. Data aggregation represents an essential paradigm for wireless sensor networks. In the case where sensors are deployed across multiple organizations, our construction may provide a promising approach to address privacy issues arising in sensor network aggregation.

4 System Model

The suggested system model consists in M participants. Each participant is represented as a uniquely identified node. Each participant p_i has its private numerical input q_i. The global outcome is the output of the aggregation function. Each participant is assigned to a private input. The system model we provide for our approach assumes that an input could be sent as a single message or split out into a set of different messages. Participants could either be honest or dishonest. An honest participant strictly follows the protocol. A dishonest node may misbehave to reveal the input of honest nodes. An important issue is when a group of dishonest participants, named coalition, conspire in order to reveal the input of a given participant. We aim to enable participants to have the precise aggregate of their inputs. At the same time, no user should learn anything about the values of other honest users, even if he colludes with the other malicious users.

The M nodes are clustered into r ordered groups, from g_0 to g_{r-1}. Each group contains N_i nodes ($\sum_{i=0}^{r-1} N_i = M$). A node p_i in group g_j maintains two sets of nodes: A set P_o of officemates containing all nodes belonging to the same group ($P_o = \{p\} \in g_j \backslash p_i$) (i.e. the set p of participants in g_j except p_i)

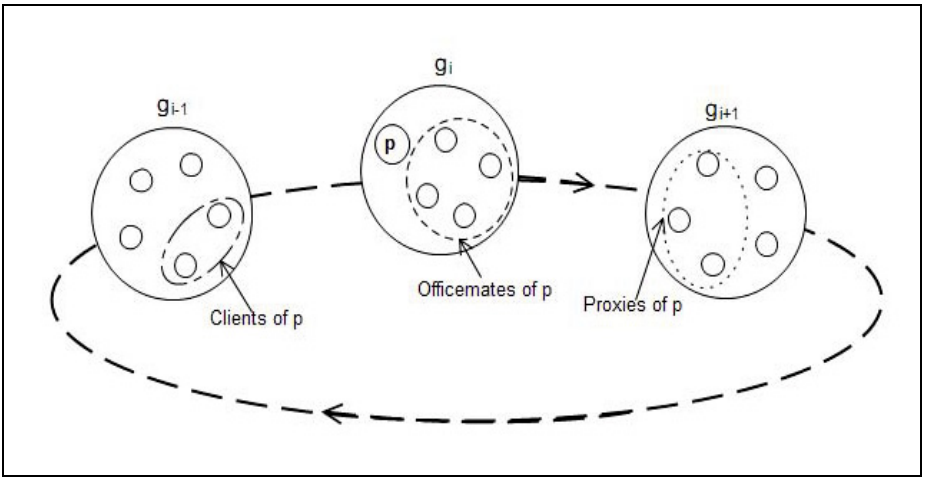


Fig. 1. The System Model

and a fixed-size set P_p of proxies, containing a subset of nodes in the next group ($P_p \subseteq g_{j+1 \bmod r}$). Therefore, all groups virtually form a ring, g_0 being the successor of g_{r-1}. Each group g_j is a clique (Fig. 1). We define a client of p as a node for which p acts as a proxy. Which means that if p is a proxy of q, then q is one of its clients. Every node maintains a list of its clients in the previous group ($P_c \subseteq g_{j-1 \bmod r}$). A node discards every message originating from a node that is not in $P_c \cup P_o$. We assume a random uniform distribution of nodes across the r groups. Nodes in the successor groups are distributed uniformly at random as proxies in the predecessor groups. Communication is assumed to be reliable, which means that no message is lost after it has been sent, and that every message sent from p to q arrives unaltered at node q. Furthermore, the nodes are always operational.

5 The Distributed Protocol

In the following we give a description of the suggested *"Distributed protocol for Privacy preserving Aggregation" DiPA*. In the proposed protocol, participants may use a simple sharing scheme to encode their private inputs. Then they send the shares of their inputs to proxies, belonging to another group. Each group computes a partial aggregate that is further broadcasted to all other groups. Each participant eventually outputs the same global aggregate.

5.1 Description of DiPA

The suggested protocol consists in three steps: Sharing, counting and broadcasting. During the first step, each participant generates a single message or a set of shares reflecting the private input and sends each generated share to one of its

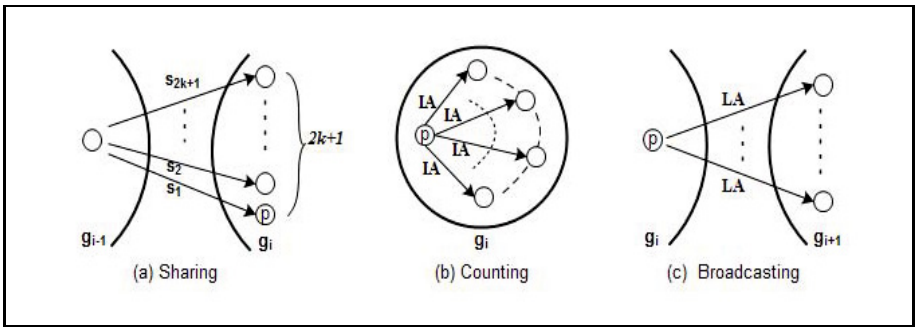


Fig. 2. The protocol steps

proxies. In the counting step, each node computes the *sum* of the clients' shares. Such *sum* is called individual aggregate. Then each proxy broadcasts the result to its officemates. Each participant computes the local aggregate which is the *sum* of the individual aggregate received from its officemates. Finally, the local aggregates are forwarded along the ring so that all nodes eventually compute the final outcome.

Step 1: Sharing. (Fig. 2(a)) Each participant node could cast its input as: a single share, 3 shares, ... , $2k+1$ shares, ... or as a set of $2k_{max}+1$ shares. If a given participant wants to send a value x, in the case where this participant should split its data out into $2k+1$ shares such that $k \in \{1, 2, ..., k_{max}\}$, the set of $2k+1$ shares is generated as follows: k values are randomly chosen from the set S (i.e. set of possible entries). The other k values represent the inverses of the first k chosen value, and a single share represents the real participant's input. Thus, aggregating the generated shares outputs the actual value x. Let us recall that some participants could send the value of x in a single share. Hence, when a proxy receives a message from a given client node, the proxy could not distinguish if such share was generated as a single one or it is one among the previously generated $2k+1$ shares with $k \in \{1, 2, ..., k_{max}\}$. To compute functions of type min/max, the set of shares may be chosen randomly such that the min/max of those shares is the real input (e.g. if the true input is 3, the set of shares may be $\{6, 3, 5\}$).

Once a node has generated its $2k+1$ shares, it sends each of them to a distinct proxy. The number of proxies is to be chosen accordingly, $\mid P_p \mid = 2k+1/k \in \{0, 1, ..., k_{max}\}$. Once every node in the system has received one share from each of its clients, the sharing round is over.

Algorithm 1. Sharing phase

```
Procedure share(v, k)
1.  for i ← 1 to k do
2.      s_i ← rand(S)          # choose random value from S
3.      s_{i+k} ← inv(s_i)     # the inverse of the chosen value
4.  end for
5.  s_{2k+1} ← v               # the real input
6.  for i ← 1 to 2k + 1 do
7.      send([share, s_i], proxy)
```

Step 2: Counting. (Fig. 2(b)) Recall that in the sharing step each participant node, within a group g_{i-1}, sends $2k+1$ shares with $k \in \{0, 1, ..., k_{max}\}$, to its proxies in the group g_i. Now, in the counting step, each proxy within the receiving group g_i aggregates the received shares from its clients in the group g_{i-1}. The resulted value is designated as the Individual Aggregate (IA). Note that each proxy will have its own individual aggregate. Once a participant node has received the expected number of shares from its clients, it broadcasts the computed individual aggregate to its officemates. Each officemate will compute the aggregation of the received individual aggregates resulting in a Local Aggregate (LA) of its group. Then, each officemate will forward the computed local aggregate to its proxies in the next group.

Algorithm 2. Counting phase

```
Upon event <receive|[share, s_i]> do
1.  ia ← f(ia, s_i)
Procedure count(ia)
2.  foreach officemate ∈ P_o do
3.      send([IndividualAggregate, ia], officemate)
Upon event <receive|[IndividualAggregate, ia]> do
4.  la ← f(la, ia)
```

Step 3: Broadcasting. (Fig. 2(c)) During the previous step, each officemate send the local aggregate of its group to its proxies. If all the officemates are honest, then all local tallies computed in a group are identical. However, if a computation error (i.e. a message lost) or dishonest acts occur, the same group may have different values of the local aggregate. In fact when a proxy in the group g_i receives local aggregates from its clients in the group g_{i-1}, the local aggregate of the group g_{i-1} could be the value corresponding to the most received ones. Then this value will be sent to the proxies in the next group g_{i+1}. Local aggregates are then forwarded along the ring.

Since, a message that represents a local tally contains the index of the group; participants can notice which group has the received local tally. Once a participant node in the group g_i receives the local aggregate of its group from the

client in the previous group g_{i-1}, the local aggregate is no longer forwarded. This means that such value has crossed the ring and received back to its initial sender. When a participant node receives local aggregates of all groups, the global aggregate (GA) is computed from these local aggregates. This global aggregate is the global outcome of the protocol. Note that all participants should compute the global aggregate after reception of local aggregates.

Algorithm 3. Broadcasting phase

```
Procedure broadcast(la, i_group, P_p)
1.  foreach proxy ∈ P_p do
2.      send([LocalAggregate, i_group, la], proxy)
Upon event <receive|[LocalAggregate, i_group, la]> do
3.  if (i_group ≠ p_group) then
4.      broadcast(la, i_group, P_p)
5.  end if
```

5.2 Correctness

The aim of this subsection is to prove the correctness of the suggested protocol. Here we assume that all participants are honest respecting the protocol specification. An honest participant votes using $2k+1$ messages with $k \in \{0, 1, ..., k_{max}\}$, and remains conform to the rules of the other steps of the protocol.

Theorem: Consider a distributed application for computing an aggregation function over an Abelian group. The DiPA protocol terminates and each participant eventually outputs the global aggregate of the participants' inputs.

Proof: In the proposed protocol each participant maintains a list of clients, a list of officemates and a list of proxies. Thus, each participant knows the number of messages it is supposed to receive in each step. Since every participant respects the protocol rules and every message eventually arrives, each step completes. As the algorithm is a finite sequence of steps, it is guaranteed to eventually terminate.

Each participant in a group g_i sends $2k+1$ shares where $k \in \{0, 1, ..., k_{max}\}$, such that the input of a given participant is $v = f(s_1, s_2, ..., s_{2k+1})$. Each share s_i is sent to a distinct proxy within the group g_{i+1}. Thus, the aggregation of inputs sent by all participants in g_i is defined by the following formula: $S = f[f(s_1, ..., s_{2k+1}), f(s_1^{"}, ..., s_{2k^{"}+1}^{"}), ..., f(s_1^{(n)}, ..., s_{2k^{(n)}+1}^{(n)})]$, where n is the number of participants in the group g_i. Since all the generated shares are received by proxies in the group g_{i+1} and f is associative and commutative function, the aggregation of the computed individual aggregates reflects the aggregation of inputs of participants in the group g_i. Therefore the local aggregate computed in each group reflects the aggregation of the inputs of all participants in the previous group over the ring.

During the broadcasting step, each participant forward the received local aggregates along the ring. The global aggregate is computed by each participant,

once all local aggregates are received. So, since participants do honestly forward the local aggregates along the ring and the messages are eventually received, each node ends up with the correct values for the local aggregates of every group, thus the correct global aggregate.

Complexity: During the first step of the protocol, each node sends $2k + 1/k \in \{0, 1, ..., k_{max}\}$ shares reflecting its actual input. Each node sends $N_i - 1$ individual aggregates to its officemates during the counting phase, and $r.(2k + 1)$ local aggregates to compute the global aggregate. Thus, the message complexity of the suggested protocol is $O(r.k + N_i)$. The spatial complexity of DiPA is similar to the message complexity, since each node maintains a list of $2k + 1$ proxies, $N_i - 1$ officemates and at most N_{i-1} clients and they store $r.(2k + 1)$ possible values of local tallies.

6 Privacy in a Distributed Polling Application

6.1 Distributed Polling

In order to evaluate the suggested protocol, we have considered a real application of the algorithm, which consists in making a binary poll within a social network. During a polling session, the inputs represent the votes of different participants (e.g. a value (yes or no), (agree or not agree), (support or against) etc...). After the execution of the polling protocol, the output represents the tendencies of the majority of participants.

We use DiPA to study the privacy ensured in a concrete case study (i.e. polling). Polling may be represented as the distributed aggregation function $(sum, \{-1, +1\}, \mathbb{Z})$. Each participant p votes for a binary value $v_p \in \{-1, +1\}$. The global outcome of a voting session is the aggregate-sum of the votes made by the existing participants $(\sum_p v_p)$.

In a polling session, the data to hide consists in the participants votes. As specified in DiPA, during the first step each participant generates a set of $2k+1$ shares such that $k \in \{0, 1, ..., k_{max}\}$, those shares are also called ballots in this case study. Regarding the protocol specification, the $2k+1$ shares are generated in such a way that k shares are randomly chosen from the set $S = \{-1, +1\}$, k shares are the inverses of the first shares, and a single share represents the actual participant vote. Therefore, in the polling protocol, $k+1$ shares will contain the value of the participant vote and k shares represent the inverse of the vote. The remaining steps of the protocol stay unchanged.

6.2 Privacy Analysis

The aim of this subsection is to discuss the privacy ensured by DiPA for a polling application. During the polling session, nodes send $k + 1$ messages with a given tendency and k messages with the inverses where $k \in \{0, 1, ..., k_{max}\}$. We assume that B dishonest nodes in a group of size N collude to disclose votes of their clients. The coalition of B dishonest nodes wishes to take a decision on an

honest client vote based on the received shares. Thus, they proceed as follows: As they receive the set of shares from a given honest node, they compute the sum of the received messages. If the computed sum is strictly-negative, they will assume that the participant voted -1. And if it is strictly-positive, the vote is considered to be $+1$. Otherwise, they cannot decide on the value of the participant vote. Based on this rule, the set of dishonest node may success to disclose the client vote, and they may fail.

In the following, we aim to compute the probability of success of their decision. Let us note k_1 and k_2 the number of messages received by the set B of dishonest nodes, representing respectively the participant vote and the inverse of the vote. Thus, $0 \leq k_1 \leq (k+1)$ and $0 \leq k_2 \leq k$. So, it can be seen that the honest nodes will receive $k+1-k_1$ messages reflecting the vote and $k-k_2$ messages representing the inverse of the vote. Therefore, there is P_d possible distributions of shares over the list of nodes in the successor group, such that $P_d = \sum_{k_1,k_2} \binom{B}{k_1}\binom{B-k_1}{k_2}\binom{N-B}{k+1-k_1}\binom{N-B-k-1+k_1}{k-k_2}$. As dishonest nodes respect the specified decision rule, they success to disclose the participant vote when $k_1 > k_2$ (i.e. Most of the received shares reflect the vote). Therefore, there is $P_s = \sum_{k_1>k_2} \binom{B}{k_1}\binom{B-k_1}{k_2}\binom{N-B}{k+1-k_1}\binom{N-B-k-1+k_1}{k-k_2}$ possibilities to satisfy this condition (i.e. $k_1 > k_2$). Thus, the probability to disclose the vote of an honest participant, using $2k+1$ shares, by a coalition of B dishonest nodes is $p(k) = P_s/P_d$.

Since participant nodes vote with $2k+1$ shares such that $k \in \{0, 1, ..., k_{max}\}$, the probability to reveal votes of honest participants by a coalition of B dishonest nodes is $p = \sum_{k=0}^{k_{max}} \pi_k.p(k)$, where π_k is the proportion of nodes voting with $2k+1$ shares (i.e. $\sum_{k=0}^{k_{max}} \pi_k = 1$).

In Fig. 3, we plot the average of the probability of vote detection as a function of the number of dishonest nodes. In this study, we consider a group of 100 participants with $k_{max} = 3$ (i.e. participants use $1, 3, 5$ or 7 shares to express

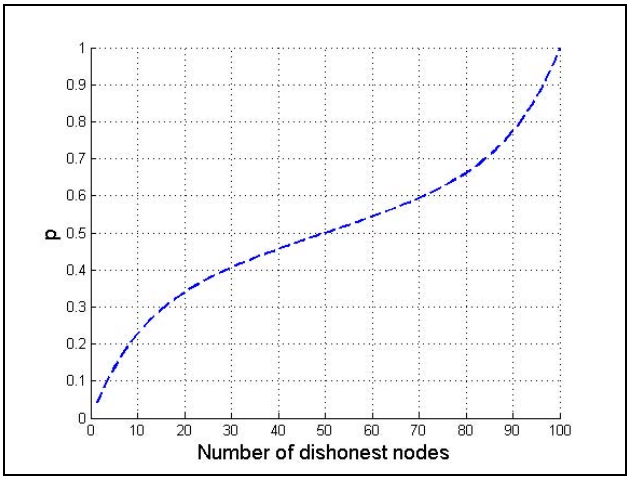


Fig. 3. The probability of vote detection

their votes). The proportions π_k of nodes using $2k + 1$ shares are chosen uniformly at random. The probability has been computed several times varying the proportions π_k for each computation.

As expected, the probability of detecting an honest participant's vote is an increasing function. This is due to the fact that: more dishonest participants collude, more opportunities arise to disclose the vote. However, the result shown in Fig. 3 might be considered as the upper bound of the probability to detect an input in a DiPA based system. In this privacy analysis, we have considered the binary-poll application. So, there are just two possible initial inputs $\{-1, +1\}$. This make disclosing vote by dishonest nodes easier. Hence, enlarging the set of possible inputs S, for other use case, imply an improvement of input's privacy. Furthermore, we assumed a decision rule where the set of dishonest nodes always take a decision about the participants vote whatever they receive. Thus, dishonest nodes do not have any proof that the revealed value represents the actual input. Also, this rule is not applicable for other DiPA system. Therefore, using the protocol for applications, with large set of possible inputs, and high value of k_{max} leads to an enhancement of the privacy ensured by DiPA for other scenarios.

7 Conclusion

In this paper, we have presented DiPA, a distributed protocol that allows a set of users to compute aggregate functions, which can be expressed as Abelian group, over their private inputs. The ability to privately compute this kind of function has applications in several widely varying contexts. The suggested protocol is based on an overlay construction. The communication cost of our protocol is $O(r.k + N_i)$ where r is the number of group in the overlay and N_i represents the number of participant per group. In order to evaluate the privacy ensured by our protocol, we have considered the distributed polling application, where participants care about the privacy of their votes. Currently, we plan to study the behavior of this protocol in a dynamic environment where participants may disconnect during the protocol execution. Moreover, we are working on more realistic scenarios in different environment, such as e-health application and Wireless sensor networks, where the suggested technique can be implemented.

References

1. Blum, A., Dwork, C., McSherry, F., Nissim, K.: Practical privacy: the SuLQ framework. In: Proceedings of the Twenty-Fourth Symposium on Principles of Database Systems (PODS 2005), pp. 128–138. ACM, New York (2005)
2. Castelluccia, C., Mykletun, E., Tsudik, G.: Efficient aggregation of encrypted data in wireless sensor networks. In: Proceedings of The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005), July 17-21, pp. 109–117 (2005)
3. Guerraoui, R., Huguenin, K., Kermarrec, A.M., Monod, M.: Decentralized Polling With Respectable Participants. In: Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS 2009), Nmes, France, December 15-18, pp. 144–158 (2009)

4. Du, W., Atallah, M.J.: Secure multi-party computation problems and their applications: a review and open problems. In: Proceedings of the 2001 Workshop on New Security Paradigms (NSPW 2001), Cloudcroft, New Mexico, September 11-13, pp. 13–22 (2001)
5. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), Chicago, Illinois, USA, November 03-05, pp. 160–164 (1982)
6. Su, B., Wang, T.: Design and analysis for private determination protocol of segment-circle position relation. In: Proceedings of the Industrial Control and Electronics Engineering (ICICEE 2012), August 23-25, pp. 1430–1433 (2012)
7. Pfitsmann, B., Waidner, M.: Unconditionally Untraceable and Fault-tolerant Broadcast and Secret Ballot Election. Communications of the ACM 21(21) (1992)
8. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
9. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Pailliers public-key system with applications to electronic voting. The International Journal of Information Security - Special Issue on Special Purpose Protocols 9, 371–385 (2010)
10. Malkhi, D., Margo, O., Pavlov, E.: E-voting without Cryptography. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 1–15. Springer, Heidelberg (2003)
11. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: Proceedings of the 2010 International Conference on Management of data (SIGMOD 2010), pp. 735–746. ACM, New York (2010)
12. Bocek, T., Peric, D., Hecht, F.V., Hausheer, D., Stiller, B.: Towards a Decentralized Voting Mechanism for P2P Collaboration Systems. Technical Report, No. ifi-2009.02
13. Bogetoft, P., et al.: Secure Multiparty Computation Goes Live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (2009)
14. Gambs, S., Guerraoui, R., Harkous, H., Huc, F., Kermarrec, A.M.: Scalable and Secure Aggregation in Distributed Networks. CoRR (2011)
15. Bickson, D., Dolev, D., Bezman, G., Pinkas, B.: "Peer-to-Peer Secure Multi-party Numerical Computation. In: Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing (P2P 2008), pp. 257–266. IEEE Computer Society, Washington, DC (2008)
16. Shi, E., Hubert Chan, T.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-Preserving Aggregation of Time-Series Data. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 2011), San Diego, California, USA, February 6-February 9 (2011)
17. Goldwasser, S.: Multi party computations: past and present. In: Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing (PODC 1997), pp. 1–6. ACM, New York (1997)
18. Kreitz, G., Dam, M., Wikström, D.: Practical private information aggregation in large networks. In: Aura, T., Järvinen, K., Nyberg, K. (eds.) NordSec 2010. LNCS, vol. 7127, pp. 89–103. Springer, Heidelberg (2012)
19. Jung, T., Li, X.Y., Tang, S.J.: Privacy-Preserving Data Aggregation without Secure Channel: Multivariate Polynomial Evaluation. arXiv:1206.2660 [cs.CR] (August 01, 2012)

Identifying the Optimal Level of Parallelism in Transactional Memory Applications*

Diego Didona[1], Pascal Felber[2], Derin Harmanci[2], Paolo Romano[1], and Jörg Schenker[2]

[1] Instituto Superior Técnico/INESC-ID, Portugal
[2] University of Neuchâtel, Switzerland

Abstract. In this paper we investigate the issue of automatically identifying the "natural" degree of parallelism of an application using software transactional memory (STM), i.e., the workload-specific multiprogramming level that maximizes application's performance. We discuss the importance of adapting the concurrency level to the workload in two different scenarios, a *shared-memory* and a *distributed* STM infrastructure. We propose and evaluate two alternative self-tuning methodologies, explicitly tailored for the considered scenarios. In shared-memory STM, we show that lightweight, black-box approaches relying solely on on-line exploration can be extremely effective. For distributed STMs, we introduce a novel hybrid approach that combines model-driven performance forecasting techniques and on-line exploration in order to take the best of the two techniques, namely enhancing robustness despite model's inaccuracies, and maximizing convergence speed towards optimum solutions.

1 Introduction

The pervasive adoption of multi-core architectures (from HPC clusters to embedded systems) has raised the urge to identify paradigms capable of simplifying the development of parallel applications. Transactional memory (TM) [1] has garnered a lot of interest of late precisely because, thanks to its simplicity and scalability, it appears to be a promising alternative to classic lock-based synchronization. Over the last years, a wide body of literature has been published on TM, and several variants have been developed, including hardware-based (HTM), software-based (STM), and distributed (DTM) [2]. One of the key results highlighted by existing research is that, independently of the nature of the synchronization scheme adopted by a TM platform, its actual performance is strongly workload dependent and affected by a number of complex, often intertwined factors (e.g., duration of transactions, level of data contention, ratio of update vs read-only transactions).

* This work has been partially supported by the projects "Cloud-TM" and "ParaDIME" (co-financed by the European Commission through the contract no. 257784 and 318693), project specSTM (PTDC/EIA-EIA/122785/2010), the COST Action Euro-TM (IC1001) and by FCT (INESC-ID multiannual funding) through the PEst-OE/EEI/LA0021/2011 Program Funds.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 233–247, 2013.

Among these numerous factors, an often neglected one is the concurrency level used by the application. However, as we will also quantitatively show in this paper, the identification of the "right" level of concurrency represents a critical factor that can have a strong impact on performance of TM applications. Unfortunately, this decision is far from being trivial, as the off-line tuning of this parameter is a costly and error-prone process. Further, any static configuration can lead to suboptimal performance in presence of dynamic workloads. Hence, we argue that work done by the TM programmers to develop parallel applications risks to be wasted, unless effective mechanisms are available to tune the concurrency level of TM applications, and let them take full advantage of the underlying parallel architecture.

In this paper, we address the problem of self-tuning the concurrency level according to the application workload (which we call "elastic scaling") in various application settings. Related problems have been addressed previously but limited attention has been devoted to dynamically identifying the optimal degree of parallelism for a (D)TM platform, namely the degree of local (i.e., number of active threads) and possibly global (i.e., number of nodes in a DTM) concurrency that maximizes the throughput of complex (D)TM applications.

We present experimental results obtained considering two extreme scenarios: on shared-memory systems with a low-level STM library written in C, and in distributed systems with a high-level DSTM infrastructure written in Java. We first show that realistic benchmarks exhibit widely different performance depending on the degree of parallelism, and that adapting the number of threads at runtime can improve performance of some applications over any execution with a fixed number of threads. By applying small modifications to the benchmarks and the underlying STM runtime in a shared-memory system, one can straightforwardly optimize the concurrency level using exploration-based on-line optimization techniques, e.g., using hill climbing or gradient descent algorithms.

In distributed settings, however, the cost of testing configurations with a different number of threads (i.e., nodes) is prohibitive, as it requires transferring state, generates additional traffic, and takes orders of magnitude more time than in centralized settings. Therefore, in such settings, one should instead rely on modeling techniques to predict the expected gains from adding or removing nodes for adapting the concurrency level. However, performance modeling techniques are unavoidably subject to approximation errors, which can lead to the identification of suboptimal configurations. We show how this problem can be tackled by introducing a novel self-tuning methodology that combines exploration-based and model-driven approaches: models help predicting the evolution of performance at a large scale, while local, inexpensive exploration can be used to gather feedback on the model's accuracy and allow its progressive enhancement. To this end, we show how machine learning techniques can be exploited to learn corrective factors aimed at "patching" the output of performance models and correcting biases/approximation errors that may otherwise impair their accuracy.

The rest of this paper is organized as follows. We first give a brief overview of related work in Section 2. We present, in Section 3, our on-line exploration-based approach for shared-memory STM systems. In Section 4, we focus on distributed STM systems, and present a hybrid self-tuning approach combining on-line exploration and model-driven optimization techniques. We finally conclude in Section 5.

2 Related Work

Adapting the concurrency level to system workload (elastic scaling) in order to improve performance and/or resource usage is an issue that has already been addressed in previous research. Different approaches to the problems exist, especially depending on the area where the adaptation is performed.

Part of the research targets concurrent execution on a single machine. Reimer et al. [3] propose to adapt the concurrency level of parallelizable portions of a scientific application. This approach does not allow changing the concurrency level during the execution of the code portion, whereas with our exploration-based approach workload changes within such portions can be tracked and concurrency levels can be adapted. Heiss and Wagner [4] study the tuning of the concurrency level (number of concurrently running transactions) within a transactional processing system (e.g., database server) running on a single machine. They propose two algorithms, one of which is a hill climbing approach similar to ours. Schroeder et al. [5] use a feedback control loop that is initialized with a close-to-optimal value thanks to the use of queueing theoretic models. This initialization allows the approach to converge fast under abrupt workload changes. Abouzour et al. [6] propose a hybrid approach merging these two studies. Our work differs since we adapt the concurrency level of threads within an arbitrary application, while other approaches tune the concurrency level of transactions only in the restricted context of a transactional processing system.

Few studies exist for the adaptation of the concurrency level for TM-based applications (using non-replicated TM). Yoo and Lee [7] propose rescheduling threads in order to reduce contention due to data conflicts. Such a technique has the effect of adapting the concurrency level of the application, because rescheduled threads are removed from the execution for a while. Ansari et al. [8] aim to improve application efficiency by reducing resource usage without sacrificing performance. To achieve this objective, they maintain the transaction abort rate under a predefined threshold. Our approach differs from this work as (i) we do not use any fixed thresholds, and (ii) we focus on optimizing transaction throughput, allowing us to improve application performance (especially under changing workloads) compared to an execution with any fixed concurrency level.

Elastic scaling in distributed settings corresponds to automatically adapting, in face of varying workloads, the number of nodes the platform is deployed onto. In the area of replicated relational databases, several mechanisms have been proposed [9,10] to tackle this problem. Our work proposes a solution specialized for (D)TM platforms. A distinguishing aspect of our work is that it identifies the

level of concurrency that maximizes the throughput of complex (D)TM applications by considering scaling at two levels: (i) number of nodes of the platform, and (ii) number of active threads running on a node.

In the area of performance modelling of STM, existing literature can be subdivided in solutions based either on analytical techniques [11,12], or on statistical methods [13,14]. Solutions based on analytical models have a good extrapolation power, namely they typically exhibit good accuracy even in workload/scale scenarios not previously explored. However, their accuracy can degrade significantly with scenarios that challenge the set of assumptions they rely on to ensure mathematical treatability. Statistical methods, due to their black box nature, suffer of limited extrapolation, but can achieve typically very accurate predictions in regions of the state space close to those already observed during the learning phase. The solution proposed in Section 4 aims at combining the advantages of both approaches. On the one hand, it relies on analytical performance models to achieve high extrapolation. On the other hand, it exploits feedback collected from deployed DTM system to learn, using machine learning techniques, a corrective function aimed at fixing possible inaccuracies of the analytical model.

3 Shared-Memory STM

Ideally, it is desired to run applications at their natural degree of parallelism, i.e., a point where each thread does "sufficient" useful work without inducing "too much" contention. The exact definition of both quantities varies depending on the context and it is generally not obvious to find this natural degree of parallelism for a given application. For workloads where contention due to data synchronization does not change throughout the application execution, the best level of parallelism can be found offline by repeatedly restarting the application with different sets of parameters. However, the contention a workload generates may vary during the lifetime of the application, i.e., the natural degree of parallelism represented by the workload varies as the application executes. Hence, a general solution to this problem would need to track the workload generated by the application on-line. In this section, we first motivate why adaptivity is important. We then describe our exploration-based mechanisms for adjusting the degree of parallelism. Finally, we show how these mechanisms can help optimize throughput according to the dynamic properties of the workload.

3.1 The Need for Adaptivity

Before dwelling on the actual exploration algorithm, let us briefly consider the benefits of such adaptive techniques on the application `intruder`, part of the widely used STAMP benchmark suite [15]. This application emulates a signature-based network intrusion detection system and exhibits a workload that evolves over time.

Figure 1 indicates the performance of the benchmark when executed with varying number of threads (dashed line), or when dynamically changing the

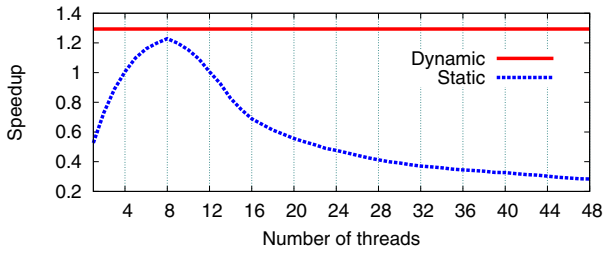


Fig. 1. Speedup of the `intruder` benchmark as compared to sequential (non-STM) version, using static and dynamically evolving numbers of threads

number of threads (plain straight line corresponding to a constant value). One can observe that performance is significantly better when dynamically adapting concurrency than with any fixed number of threads. Note that the experiment was run on a 48-core machine, i.e., the number of physical cores was not the limiting factor.

3.2 An Exploration-Based Approach

Our exploration-based approach performs on-line monitoring of key performance metrics. It allows us to find the natural degree of parallelism of an application by running it with an iterative algorithm controlling its concurrency level. The algorithm terminates when all the work to be performed by the application is accomplished. Each iteration of the algorithm has three phases, as illustrated in Figure 2:

- **Measurement phase:** In this phase, the application runs with a fixed number of threads. Key performance metrics (numbers of commits and aborts) are measured during a certain time period. The commit rate gives an indication of raw transaction throughput, while the abort rate is a good measure of contention.
- **Decision phase:** In this phase the algorithm decides between two actions: increasing or decreasing the number of threads. If the last measurement phase shows improvements in terms of commit rate, the action performed in the previous iteration is repeated (addition or removal of threads); otherwise, it is reversed. The decision taken in this phase corresponds to a *hill climbing* technique maximizing transaction throughput, i.e., commit rate. The technique explores configurations in the vicinity of the current one by dynamically adding or removing threads, until a (local) maximum is reached. Even when reaching such a point, the configuration is tested for adapting to possible variations in the workload that would shift the optimal configuration(s).[1]

[1] One should note at this point that none of the benchmarks we experimented with (STAMP applications and various micro-benchmarks) exhibits multiple maxima when observing throughput as a function of the number of threads, up to the hardware limit of our 48-core test machine.

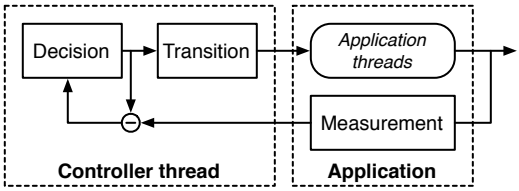


Fig. 2. The principle of the exploration-based algorithm is akin a feedback control loop. The three phases are shown in rectangles with solid lines.

– **Transition phase:** An external controller thread adds or removes threads to/from the application according to outcome of the decision phase.

For faster adaptation to the workload, we tune the duration of the measurement phase such that we have sufficiently many samples (i.e., commits) to take sound decisions but without wasting too much time. In this way, the algorithm reacts fast by quickly collecting measurements with applications composed by short transactions while it will take more time to adapt, but will still take correct decisions, for applications with long transactions.

Inserting the application inside the iterative algorithm required us to introduce (i) code observing performance, for the measurement phase, and (ii) a *controller thread* that performs decision and transition phases to modify the parameters of the application based on the measured performance. This extra thread controls the main execution loop of the application and can add or remove as many transactional threads as required during run time.

3.3 Performance in Centralized Systems

We give a brief overview of the performance of our exploration-based approach on a 48 core machine with four AMD Opteron 6172 processors with one STAMP application. A large collection of other experimental results can be found in a companion research report [16].

Figure 3 (left) shows the behavior of the exploration-based algorithm with the `intruder` application. As one can observe, the number of threads (values averaged over 2-second periods for clarity) increases in the first half of the execution to reach 13, then drops sharply to account for changes in the workload. The last part of the execution uses only few threads, which reduces the commit throughput but limits contention and avoids most aborts.

To better understand what triggers such changes in the workload, we show in Figure 3 (right) the variations in transaction lengths, as reported by the size of the read and write sets, during the execution of `intruder`. Values are averaged over groups of 10,000 transactions and sizes are shown on a logarithmic scale. The application repeatedly executes a sequence of 3 transactions. Two of them, denoted as T_2 and T_3 in the graph, do not vary much over time. The third one, T_1, exhibits an interesting trend that explains why our approach is so

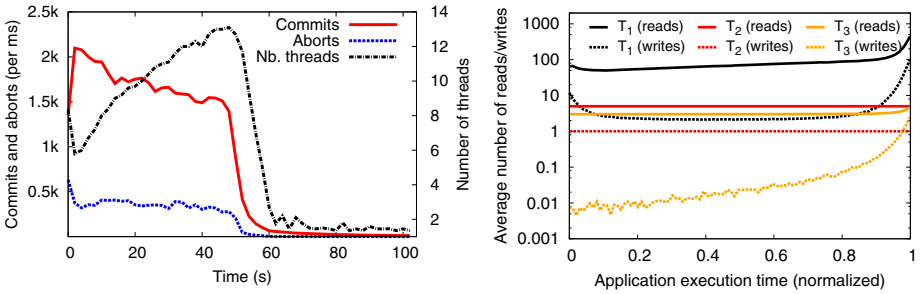


Fig. 3. Evolution of the number of threads (left) and transaction read- and write-set size (right) with the `intruder` benchmark using exploration-based scaling

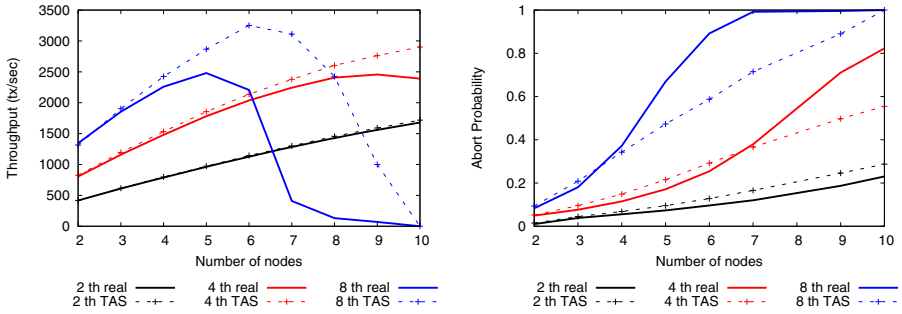
effective: transactions read more and more data, with a sharp spike in the end, while their number of writes first decreases before stabilizing and increasing steeply in the end. Therefore, the last transactions to execute are very long and, hence, are expected to encounter much contention. Limiting concurrency increases the likelihood of commit and, in turn, improves overall performance. This example clearly shows the benefits of using a adaptive strategy for best tuning the concurrency degree to the varying workload properties encountered in real applications.

Note that we modified and experimented with other applications of the STAMP benchmark suite. We found out that, while `intruder` benefits most from dynamic adaptation of the concurrency level because of the wide variations in its workload, our exploration-based algorithm is also effective with other applications and can quickly find the optimal number of threads [16].

4 Distributed STM

When considering DTM systems [17], the degree of concurrency of the platform, which we call also *global multiprogramming* level, is determined not only by the number of threads deployed on each node, but also by the number of nodes composing the platform. We note that in distributed settings, purely exploration-based techniques, like the one described in Section 3, are much less effective for two main reasons. First, due to the quadratic growth of the solution's space, the number of exploration steps required to identify the optimal solution is expected to grow significantly. Second, unlike in shared memory TM systems, in DTM scenarios exploratory steps requiring altering the number of nodes in the platform require triggering state-transfer phases that can induce significant additional load [18,19] and lead to severe performance degradation [9].

We argue, therefore, that in DTM settings approaches relying on performance models to forecast the optimal degree of concurrency of the platform are preferable to strategies based on pure exploration. Model-driven elastic scaling techniques are also attractive as they represent a fundamental building block for



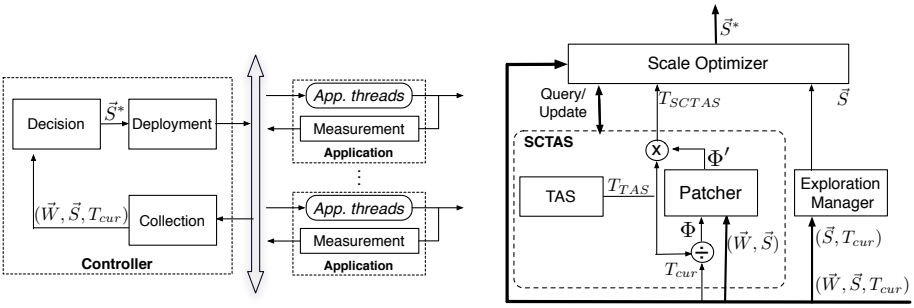
(a) Throughput (b) Abort probability

Fig. 4. Accuracy of TAS' predictions

QoS-oriented provisioning schemes, in which the amount of computational resources (and their type, e.g., medium vs large instances in a IaaS platform) needs to be elastically adjusted to ensure given SLAs in face of variable loads [20,21,22]. On the other hand, model-based performance forecasting techniques, due to their approximate nature, rely on simplifying assumptions which can degrade significantly their accuracy in presence of challenging workloads. Further, most of these modelling techniques rely on analytical methods (e.g., queuing theory), which are *rigid*, in the sense that they cannot be "bent" to learn from the feedback gathered by the actual system and accordingly correct to enhance their accuracy. In the following we report the result of an experimental study that highlights the above mentioned issues of model-driven approaches by assessing the accuracy of a state-of-the-art performance forecasting model for DTM, namely Transactional Auto Scaler (TAS) [21].

The plot in Figure 4a shows the accuracy of TAS in predicting the throughput of a DTM application when deployed over different scales. The DTM platform used in this study is Infinispan [23], a popular in-memory distributed transactional key-value store [24]. The application running over it is a porting of the TPC-C benchmark [25], and we used, as experimental testbed, a cluster of 10 servers, each equipped with 8 cores and interconnected via a private Gigabit Ethernet. The plot highlights the ability of TAS to correctly forecast the throughput of the application when deployed over different scales, as long as the global multiprogramming level (and correspondingly the abort rate) does not grow too high. This loss of accuracy of TAS is imputable to the very high contention probability that the application exhibits in those scenarios, as reported in Figure 4b. In fact, TAS, like other analytical models of transactional systems [26,20], relies on a set of simplifying assumptions on the modelling of transactions' conflict patterns, which can lead to significant errors in very high contention scenarios.

In the remainder of this Section we introduce a novel, hybrid approach that combines model-driven and exploration-based techniques in order to achieve the benefits of both approaches, namely high robustness and high speed of



(a) General architecture (b) Internals of the decision module

Fig. 5. Overview of the Controller for the DTM platform

convergence to the optimal solution. Finally, we evaluate its effectiveness via a trace-driven simulation study.

4.1 Combining Model-Driven and Exploration-Based Techniques

The key idea at the basis of the proposed solution consists in progressively enhancing the accuracy of an analytical-based performance model (such as TAS), by exploiting the feedback on the actual performance achieved by the platform when using alternative multiprogramming levels. In order to minimize exploration costs, the additional multiprogramming configurations explored by the self-tuning mechanism are obtained by fixing the current number of nodes in the platform (hence avoiding state transfers) and altering exclusively the number of active threads per node. Figure 5a illustrates the architecture of the system, which is composed of a set of DTM nodes and a controller process. The controller process is a logical component, which can be physically deployed either on one of the computing nodes or on a dedicated one. The controller is responsible for adjusting the scale of the DTM platform, i.e., the number of nodes and threads per node, which we note as $\overrightarrow{S}$. Its logic is implemented via a closed control loop that is analogous to the one show in Figure 2, with two noteworthy exceptions: (i) in this case the controller is fed with data from multiple nodes, which need to be aggregated and averaged by the collection module, and (ii) in addition to the current throughput of the DTM, noted T_{cur}, the controller gathers also a set of statistics characterizing the workload, referred to as $\overrightarrow{W}$, which are used as input parameters of its performance prediction methodology. The workload is characterized by means of the following set of key features (which represent also the main input parameters of the analytical performance model integrated in the decision module): duration, and relative frequency, of read-only and update transactions, abort rate, average number of write operations per transaction.

The key component of the controller is the decision module, whose internal structure is illustrated in Figure 5b. The decision module is composed of three main sub-components: the *Self Correcting Transactional Auto Scaler (SCTAS)*,

the *Exploration Manager* and the *Scale Optimizer*. Below, we describe these components in detail.

SCTAS. Like classic performance forecasting models, SCTAS allows predicting the throughput achievable by the DTM platform when faced with a workload $\overrightarrow{W}$ and configured to use a global multiprogramming level $\overrightarrow{S}$. To this end, we assume that SCTAS exposes the primitive `query`, which takes as input parameters $\overrightarrow{W}$ and $\overrightarrow{S}$, and returns the forecast throughput, denoted as T_{SCTAS}. Additionally, SCTAS allows incorporating feedback on the actual throughput (T_{cur}) achieved by the DTM platform in a given operational condition via the primitive `update`, which takes as input parameters $\overrightarrow{W}$, $\overrightarrow{S}$ and T_{cur}.

As shown in Figure 5b, SCTAS is actually composed of two main modules: the TAS model, and a, so called, Patcher. The role of the Patcher is to learn a *correcting function*, denoted as Φ and defined over $\overrightarrow{W} \times \overrightarrow{S}$, which, when "applied" to the output of TAS, denoted as T_{TAS}, allows minimizing its prediction error. In principle, several approaches may be used to derive the value of $\Phi(\overrightarrow{W}, \overrightarrow{S})$, given T_{cur} and the corresponding T_{TAS} for a given $(\overrightarrow{W}, \overrightarrow{S})$ pair. In this paper, however, we take a pragmatical approach and use a relatively simple solution which, as we will show in the following, was experimentally proved to work quite effectively. Namely, we define:

$$\Phi(\overrightarrow{W}, \overrightarrow{S}) = \frac{T_{TAS}(\overrightarrow{W}, \overrightarrow{S})}{T_{cur}(\overrightarrow{W}, \overrightarrow{S})} \tag{1}$$

In order to learn how the corrective factor Φ varies with $\overrightarrow{W}$ and $\overrightarrow{S}$, SCTAS uses a decision-tree based machine learning regressor, namely Cubist [27]. Analogously to classic decision tree based classifiers, such as C4.5 and ID3 [28], Cubist builds decision trees choosing the branching attribute such that the resulting split maximizes the normalized information gain. However, unlike C4.5 and ID3, which contain elements in a finite discrete domain (i.e., the predicted class) as leaves of the decision tree, Cubist places a multivariate linear model at each leaf.

As we will see shortly, the knowledge base of the machine learner embedded in the Patcher is progressively built by exploring alternative multiprogramming levels ($\overrightarrow{S}$) in presence of a workload $\overrightarrow{W}$. For each couple $(\overrightarrow{W}, \overrightarrow{S})$ corresponding to an explored scenario, the machine learner is fed with the triple $<\overrightarrow{S}, \overrightarrow{W}, \Phi(\overrightarrow{W}, \overrightarrow{S})>$. Based on its knowledge base, the machine learner builds a function Φ', which is used by the Patcher to estimate the corrective factor for a future (unexplored) configuration $(\overrightarrow{W'}, \overrightarrow{S'})$. The throughput forecast by SCTAS, T_{SCTAS}, can then be obtained (inverting Eq. 1) as:

$$T_{SCTAS}(\overrightarrow{W}, \overrightarrow{S}) = \Phi'(\overrightarrow{W}, \overrightarrow{S}) \cdot T_{TAS}(\overrightarrow{W}, \overrightarrow{S}) \tag{2}$$

Exploration Manager. This module is in charge of determining which configurations of the multiprogramming level should be tested in order to gather feedback on the accuracy of the SCTAS performance model, and, ultimately,

Algorithm 1. Pseudocode of the controller for the DTM platform.

```
seenWorkloads ← ∅
function Optimize()
    while TRUE do
        (W⃗, T_cur) ← Collect()
        if W⃗ ∉ seenWorkloads then
            CorrectViaLocalExploration(W⃗, S⃗, T_cur)
            seenWorkloads ← seenWorkloads ∪ W⃗
        S⃗* ← argmax_{s⃗} (SCTAS.Query(W⃗, s⃗))
        if S⃗* ≠ S⃗ then
            Deploy (S⃗*)

function CorrectViaLocalExploration(W⃗, S⃗, T_cur)
    numExploration ← 0
    while numExploration ≤ M do
        SCTAS.Update(W⃗, S⃗, T_cur)
        numExplorations ← numExplorations + 1
        if numExplorations ≥ μ ∧ SCTAS.GetCurErr() ≤ ε then
            break
        S⃗ ← ExporationManager.Explore(T_cur, S⃗)
        Deploy (S⃗)
        (W⃗, T_cur) ← Collect()
```

allow its correction. We abstract over the implementation of the exploration algorithm via the Explore () primitive that takes as input parameters the pair $(\overrightarrow{S},\overrightarrow{T})$. This abstraction allows to encapsulate arbitrary exploration logics, but, in this paper, we propose and evaluate a specific heuristic, described as follows.

At each invocation of the Explore () primitive, the heuristic alters the multiprogramming level (i.e., the number of active threads per node) leaving however unchanged the total number of nodes in the platform, in order to avoid triggering expensive state transfers. The heuristic operates in two phases. In the first phase, it executes according to a hill climbing technique analogous to the one described in the Sec. 3, and aims to identify the optimal multiprogramming level t^*, using the current number of nodes. In case the Explore () primitive is invoked after having identified such value, the heuristic enters a second phase during which it suggests to test configurations according to a *zig-zag* policy that explores (untested) values around t^*, by picking alternatively between values larger and smaller than t^* at increasing distance from t^*. The rationale underlying the design of this heuristic is to prioritize the exploration of configurations that maximize the throughput of the system, and, if necessary, to allow the testing of additional, suboptimal configurations which may be beneficial to the enhancement of the knowledge base of the Patcher module of SCTAS.

Scale Optimizer. This module is responsible for choosing the scale of the DTM, by exploiting in synergy the performance forecasting capabilities of SCTAS and the local exploration-based policy of the Exploration Manager. Its purpose is to drive SCTAS through its learning phase, by inducing local explorative steps,

aimed at assessing and improving SCTAS' accuracy. The pseudocode describing its logic is in reported Algorithm 1. The method Optimize() illustrates the interactions between the Scale Optimizer and the other modules composing the architecture of the Controller. It implements a simple control loop, which starts by gathering, via the Collect() method, information concerning the current throughput T_{cur} and workload $\overrightarrow{W}$. Once collected this information, the correction process of SCTAS is triggered by invoking the method CorrectViaLocalExploration(). This method, whose detailed description will be provided shortly, explores a number of alternative configurations of the multiprogramming level, in order to gather feedback from the DTM system and extending the knowledge base of the Patcher module of SCTAS. As this method terminates, SCTAS is queried to determine the optimal global multiprogramming level. Finally, if the optimal scale predicted by SCTAS differs from the current one, the DTM is accordingly reconfigured via the Deploy() primitive. The loop cycles back, in order to continue monitoring for possible changes of the workload. Note that the correction process for a workload $\overrightarrow{W}$ is triggered only if SCTAS has not been already corrected against $\overrightarrow{W}$, which ensures the system stability in presence of stable inputs (i.e., workloads).

Let us now analyze the logic of the CorrectViaLocalExploration() method, which consists of a loop that performs two main operations. First, SCTAS is provided with a feedback about the throughput of the application in the current configuration. Next, the Exploration Manager is queried to determine the multiprogramming level to be tested in the next iteration, and the DTM is accordingly reconfigured via the Deploy() primitive. The loop terminates when either one of the following two conditions is met: (i) a minimum number of explorations, μ, has already been performed, and the accuracy of the SCTAS predictions on the set of configurations tested so far has achieved a configurable threshold, denoted as ϵ in the pseudocode; (ii) a maximum number of explorations, M, has already been performed.

The latter condition ensures the eventual termination of the local exploration phase after a bounded number of attempts. The former allows to control the duration of the local exploration phase via two parameters: ϵ allows to bound the error of SCTAS on the configurations explored so far, for which it has therefore already collected measurements from the DTM platform; μ, on the other hand, allows to control directly the minimum duration of each local exploration phase, and, indirectly, to determine the amplitude of the knowledge phase of the Patcher module and its ultimate effectiveness.

4.2 Evaluation

In this section we assess the validity of the proposed hybrid approach. We compare it with a purely model-driven one and evaluate it in terms of final accuracy of the corrected model, capability of identifying the optimal scale for a DTM application and convergence speed towards it. To this end, we built a simulator which implements the logic of the Controller. Data consumed by the simulator

Table 1. Comparison between TAS and SCTAS with different values of μ

Model	Normalized Throughput wrt Max	#Local Expl	#Global Expl	Global Avg Rel Err
TAS	0.70	0	0	1.03
SCTAS($\mu = 3$)	1	9	1	0.7
SCTAS($\mu = 5$)	1	15	1	0.6
SCTAS($\mu = 7$)	1	28	2	0.05

are relevant to real traces, collected deploying the TPC-C application over a DTM of different scales, varying the number of nodes from 2 to 10, and the number of threads per node from 1 to 12. The testbed is the same as the one described at the beginning of Section 4.

We ran four simulations. Each run starts with the application deployed over a DTM composed by two nodes and one thread per node, and simulates the elastic scaling policy described in Section 4.1 by feeding it with measurements gathered (off-line) from the DTM system, till a stable state is reached. In the first simulation, the controller's decision module only relies on TAS' performance forecasting model to determine the optimal scale for the DTM; in the others, it implements the DECISION() method of the pseudocode in Algorithm 1. For these three runs we fix $M = 10$ and $\epsilon = 10\%$ and vary only the value of μ, which, we recall, determines the minimum duration of SCTAS learning phase through local exploration.

The results in Table 1 clearly demonstrate the advantages (in terms of increased accuracy) of the proposed hybrid approach when compared a purely model-driven one. Using only TAS' performance forecasting model, the controller selects a scale for which the DTM delivers a throughput that is 30% lower than the maximum achievable. On the other hand, the SCTAS-based controller is able to converge to the optimal scale, regardless of the value of μ. The sensibility of the hybrid approach to this parameter is clear when analyzing the last columns in Table 1. Results demonstrate that the value of μ represents a trade-off between convergence speed towards the optimal solution and the accuracy of the SCTAS' performance forecasting model. In Figure 6 we show how the accuracy of SCTAS increases with the numbers of explored configurations, by plotting the average prediction error of SCTAS' model. The prediction error is computed over the set of all possible scales for the DTM (in the considered range), including the unexplored ones, for which the actual throughput value has been collected offline. The discontinuity points of the curves are in correspondence of the deployment of the DTM on a different number of nodes, which yields to a major update of the corrective function learnt by the Patcher component. The plot shows that the highest accuracy is reached for μ=7; for this configuration we also show, in Figure 7, the predictions produced by SCTAS after the controller has reached its final state (contrasting them with the ones produced by the pure model-driven approach of TAS). The plot demonstrates the self-correcting capabilities of SCTAS, which, when fed with a sufficient number of feedbacks concerning the prediction's errors of TAS, can significantly improve its accuracy both in explored and unexplored scenarios, lowering the average relative

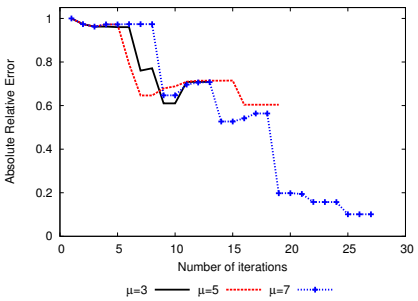


Fig. 6. Average of the relative absolute error across all scenarios

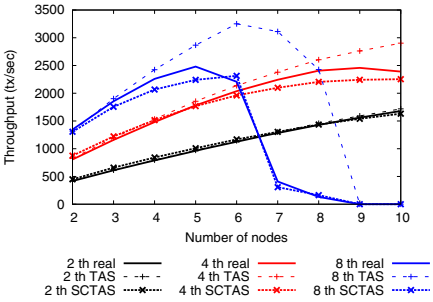


Fig. 7. Forecasts of SCTAS after a full optimization cycle ($\mu = 7$)

absolute error across all scenarios from 103% to 5%. These results highlight that the self-correcting capabilities of SCTAS can be beneficial not only to optimize the multiprogramming level of DTM applications, but also in applications (such as QoS/cost driven elastic scaling policies and what-if analysis of the scalability of DTM applications) requiring to speculate on the performance of the platform in different scale settings.

5 Conclusion

In this paper, we proposed and evaluated algorithms aimed to self-tune the multi-programming level in two radically different types of TM systems: a *shared memory STM* and a *distributed STM*. We showed that for shared memory a simple exploration-based hill-climbing algorithm can be extremely effective, even when faced with challenging workloads. However, in the distributed case, pure exploration-based approaches are no longer a viable option, as testing configurations with a different number of nodes requires triggering costly state transfer phases. We tackled this problem by introducing a novel, hybrid approach that combines performance models and local exploration, in order to achieve the best of the two methodologies: quick convergence towards the global optimum and robustness to possible inaccuracies of the performance models.

References

1. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: Proc. of ISCA (1993)
2. Harris, T., Larus, J.R., Rajwar, R.: Transactional Memory, 2nd edn. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publisher (2010)
3. Reimer, N., Haenssgen, S., Tichy, W.F.: Dynamically adapting the degree of parallelism with reflexive programs. In: Saad, Y., Yang, T., Ferreira, A., Rolim, J.D.P. (eds.) IRREGULAR 1996. LNCS, vol. 1117, Springer, Heidelberg (1996)
4. Heiss, H.U., Wagner, R.: Adaptive load control in transaction processing systems. In: Proc. of VLDB (1991)

5. Schroeder, B., Harchol-Balter, M., Iyengar, A., Nahum, E., Wierman, A.: How to determine a good multi-programming level for external scheduling. In: Proc. of ICDE (2006)
6. Abouzour, M., Salem, K., Bumbulis, P.: Automatic tuning of the multiprogramming level in Sybase SQL Anywhere. In: Proc. of ICDE Workshops (2010)
7. Yoo, R.M., Lee, H.H.S.: Adaptive transaction scheduling for transactional memory systems. In: Proc. of SPAA (2008)
8. Mohammad, A., Mikel, L., Christos, K., Kim, J., Chris, K., Ian, W.: Robust adaptation to available parallelism in transactional memory applications. HIPEAC Journal (2008)
9. Ghanbari, S., Soundararajan, G., Chen, J., Amza, C.: Adaptive learning of metric correlations for temperature-aware database provisioning. In: Proc. of ICAC (2007)
10. Zhang, Q., Cherkasova, L., Smirni, E.: A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In: Proc. of ICAC (2007)
11. di Sanzo, P., Ciciani, B., Quaglia, F., Romano, P.: A performance model of multi-version concurrency control. In: Proc. of MASCOTS (2008)
12. Heindl, A., Pokam, G., Adl-Tabatabai, A.R.: An analytic model of optimistic software transactional memory. In: Proc. of ISPASS (2009)
13. Dragojevic, A., Guerraoui, R.: Predicting the scalability of an stm: A pragmatic approach (2010)
14. Rughetti, D., Di Sanzo, P., Ciciani, B., Quaglia, F.: Machine learning-based self-adjusting concurrency in software transactional memory systems. In: Proc. of MASCOTS (2012)
15. Cao Minh, C., Chung, J., Kozyrakis, C., Olukotun, K.: STAMP: Stanford transactional applications for multi-processing. In: Proc. of IISWC (2008)
16. Schenker, J.: Optimistic Synchronization and the Natural Degree of Parallelism of Concurrent Applications — MSc Thesis (June 2012)
17. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D2stm: Dependable distributed software transactional memory. In: Proc. of PRDC (2009)
18. Narasimha Raghavan, R.V.: Balancing the communication load of state transfer in replicated systems. In: Proc. of SRDS (2011)
19. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G.: Non-intrusive, parallel recovery of replicated data. In: Proc. of SRDS (2002)
20. Elnikety, S., Dropsho, S., Cecchet, E., Zwaenepoel, W.: Predicting replicated database scalability from standalone database profiling. In: Proc. of EuroSys (2009)
21. Didona, D., Romano, P., Peluso, S., Quaglia, F.: Transactional auto scaler: elastic scaling of in-memory transactional data grids. In: Proc. of ICAC (2012)
22. Singh, R., Sharma, U., Cecchet, E., Shenoy, P.: Autonomic mix-aware provisioning for non-stationary data center workloads. In: Proc. of ICAC (2010)
23. Francesco, M., Manik, S.: Infinispan Data Grid Platform. Packt Publishing (2012)
24. Red Hat/JBoss: JBoss Infinispan (2011), `http://www.jboss.org/infinispan`
25. TPC Council: TPC-C Benchmark (2011), `http://www.tpc.org/tpcc`
26. Yu, P.S., Dias, D.M., Lavenberg, S.S.: On the analytical modeling of database concurrency control. ACM Journal (1993)
27. Quinlan, J.R.: Rulequest Cubist, `http://www.rulequest.com/cubist-info.html`
28. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc. (1993)

Biologically Sound Neural Networks for Embedded Systems Using OpenCL

István Fehérvári[1], Anita Sobe[2], and Wilfried Elmenreich[1,3]

[1] Mobile Systems Group, Lakeside Labs, Institute for Networked and Embedded Systems, Alpen-Adria-Universität Klagenfurt, Austria
{istvan.fehervari,wilfried.elmenreich}@aau.at
[2] Computer Science Department, University of Neuchâtel, Switzerland
anita.sobe@unine.ch
[3] Complex Systems Engineering, University of Passau, Germany

Abstract. In this paper, we present an OpenCL implementation of a biologically sound spiking neural network with two goals in mind: First, applied neural dynamics should be accurate enough for bio-inspired training methods, thus resulting network data is reproducible in "*in vitro*" experiments. The second is that the implementation produces code that runs adequately on up-to-date embedded graphical chips for fast on-board classification applications, e.g., video image processing. We describe the necessary steps required to implement an efficient algorithm using the OpenCL framework and present evaluation results of the execution time compared to traditional serial CPU code. We show that an optimized GPU kernel code can perform sufficiently fast to be used for future embedded neural processing.

1 Introduction

Spiking neural networks (SNNs) are a type of artificial neural networks (ANN) where communication between neurons occurs by means of time-stamped events (spikes). Researchers in the field of computational intelligence have shown that biologically sound spiking neural networks (SNNs) are comparable, but more powerful than traditional artificial neural networks [1]. Such neural networks are usually computationally complex and often require high performance computers (or even supercomputers) to run. They are, however, inherently parallel processes and therefore implementations on cheap and easy available GPUs are advantageous. In [2], the authors showed the feasibility of running different simple spiking neural network models on GPUs, but concentrated on the architecture comparison. The authors of [3] implemented a SNN using CUDA and showed its real-time capabilities. However, this implementation and others (e.g., [4], [5], [6]) rely on simplified models that are easy to implement, but neither accurate nor biologically-sound.

In this paper, we present an OpenCL implementation of a large, biologically plausible, SNN, based on the spike response model (SRM)[7], which is accurate and trainable. We have chosen OpenCL because of its support for different operating systems and graphics card vendors, including hardware without GPU support. In comparison with CUDA, which is more directly connected to the execution platform, OpenCL shows comparable performance given that the same optimizations are implemented as in the CUDA implementation [8]. CUDA is limited to Nvidia GPUs, but it is notable

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 248–252, 2013.

that a CUDA kernel can be converted to an OpenCL kernel with minimal modifications [8]. One challenge is that the biologically sound SNN models are memory-intense and GPUs provide a limited memory bandwidth. We are facing this challenge by introducing a local connection scheme with a constant number of synapses per neuron. This fact can also be exploited to optimize the memory management of the OpenCL application, which resulted in a further reduction of execution time.

2 Spiking Neural Network Model

In a spiking neural network (SNN), neurons emit pulses or *spikes* through their synapses whenever their membrane potential ϑ_i reaches its threshold value. This is caused by incoming spikes from other presynaptic neurons. Since all the spikes in the network have the exact same form, information is encoded in the chronological order of the spikes, or the so-called *spike train*. Related work indicates that there are numerous ways to model a SNN, but most of them are overly simple integrate-and-fire models, neglecting several aspects of neural dynamics (bursting, inhibitory rebound and shunting inhibition [9]). The model presented in this paper is based on the *Spike Response Model* (SRM) model described in [7]. Resulting networks are mathematically tractable, trainable and approximate the Hogkin-Huxley model with a high degree of accuracy [10]. The major difference between earlier models and the SRM is that neurons keep track of past spikes instead of being binary. Although the SRM is also considered a generalization of the leaky integrate-and-fire model [10] it expresses the neuron's membrane potential in a form of a sum of integrals instead of a set of differential equations, which makes the computation simpler while providing high accuracy.

Figure 1 shows an example of the evolution of a neuron's membrane potential. The top graph comprises a sample of input pulses and the bottom graph depicts the corresponding membrane potential along with the neuron's dynamic threshold. Individual pulses coming from excitatory synapses increase the membrane potential for a short time period (seen at 30 and 70 ms). Once the threshold indicated by a dashed line has been reached at around 80 ms, the neuron fires by emitting a near instantaneous pulse and enters a refractory period. This also increases the threshold to avoid multiple firings within a short time.

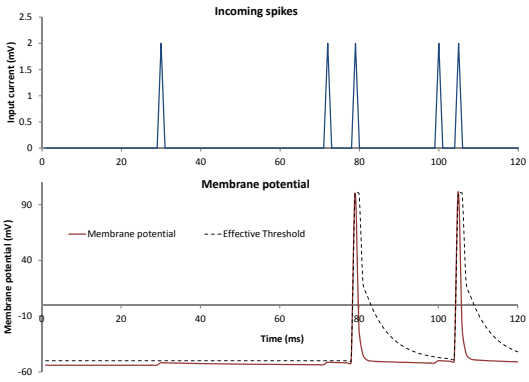


Fig. 1. Changes to the membrane potential and threshold due to incoming spikes

3 SNN Implementation with OpenCL

Related implementations of spiking neural networks meant for parallel executions are mostly event-based simulations [5], however, the integrating nature of the spike response model requires a simulation of discrete time steps, thus we choose a step length of 1 ms. To calculate the membrane potential of a neuron at any time step, we need to keep track of the time steps elapsed since its last firing, its threshold and the weights of all connecting synapses. Moreover, the membrane potential of every neuron also has to be stored for analytical purposes.

We created two different kernel programs: An *update kernel* and a *threshold kernel*. The first is used to update the neuron's membrane potential , while the second handles the firing if the membrane potential is higher than the neuron's threshold. Thus, one time step of the simulation consists of the execution of the first kernel on each neuron, followed by the second kernel executed on every neuron. The reason for this divided architecture is to provide a general execution model that is not influenced by the connection topology of the neural network. A fully-connected neural network requires to store one floating point value for every $n(n-1)/2$ connections, which is impracticable for larger network sizes in terms of memory usage, training and execution times. To make the neural network scalable for a high number of nodes, we will use a partially-connected neural network where each neuron has a fixed number of connections to its immediate neighbors. We model the neural network as an $N \times N \times N$ grid network structure (a cube) where each neuron is only connected to the local Moore neighborhood of range 2, thus having maximum $5^3 - 1 = 124$ synapses based on how far the neuron is situated from the edges. We further split the functionality of the cube, where neurons on the frontal face of the grid are used as inputs and neurons on the backface are used as outputs (see Figure 2).

Since we focus on embedded systems, we evaluate our approach by a contrived example from the image processing domain. We obtain 8-bit greyscale images from a camera as input to our neural network, where each pixel's intensity is used as constant input current of an input neuron, i.e., the darker the color, the higher the input current. At some point the neurons start to fire and the signal propagates through the network, changing the neuron's membrane potential. The output represents the frequency of the spikes, since we are only interested in the network development. Hence, we concentrate on the execution time of one time step of the SNN simulation for different neural network sizes being between $N = 10...100$. The evaluation comprises 100 runs. We ran our experiments on an AMD ATI Mobility Radeon HD 5750 state-of-the-art graphics card. We create a *naive implementation* and compare it to the baseline, a native C++

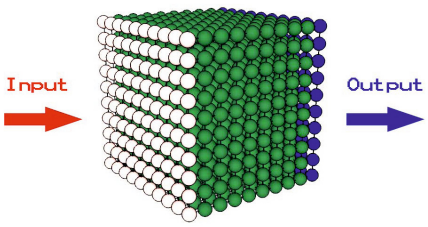


Fig. 2. Example network structure with $N \times N \times N$ neurons

version of the same code executed on a single thread. In practice, a state-of-the-art microprocessor might also provide parallel processing via multiple cores. However, for an embedded system, the parallelization would be still limited to a relatively low number of cores. In the best case, such a multi-core system would be faster by a factor equal to the number of cores.

As shown in Figure 3, the execution time for the CPU code increased rapidly with the size of the network. It shows further a significant speed performance improvement of a factor of at least 50 over a single-core CPU for all input sets larger than $40 \times 40 \times 40$ neurons. Realistic input sets are expected to have at least this size or be even larger. Thus, in comparison to CPU code, our OpenCL implementation performed exceptionally fast, with one million neurons one step took 126 ms on average.

The OpenCL memory model adopts the device's model, where the global memory refers to the main memory of the device but it can also be shared with the host. The global memory is the largest but slowest memory, especially if shared with the host system. Work-items belonging to the same work-group can access a shared local memory that provides much faster read/write operations. This memory is mainly used to synchronize work-items in a group and has a maximum size of 16kB. Finally, each of the work-items has a private memory space, which is a register-type memory. If we analyze the data flow of the naive implementation, we observe that in each time step both kernels read and write data directly from the global memory. A more effective approach would be to use the local memory block of a work-group that provides significantly faster read/write access. OpenCL instructs the target device to execute the parallel work items in groups, which in our case form a $K \times K \times K$ grid layout. By exploiting the fact that neurons are only connected in a local neighborhood, the program of each neuron could share its own data, which is already loaded from the global memory, via the shared local memory. By making this transition from exclusive read/write on global data to a locally shared architecture we could theoretically improve the performance of the algorithm. Furthermore, since the membrane potential of each neuron only depends on locally connected neurons there is no need to wait a whole execution cycle of the *update kernel* before running the *threshold kernel*. Instead the two kernels can be merged saving additional time on memory access.

By running the same simulation as described above, we obtained lower execution times (depicted in Figure 3 (b)). We achieved an increase of speed of approximately 25 %. Our results also match the results of similar implementations of spiking neural networks with different neural dynamics [5].

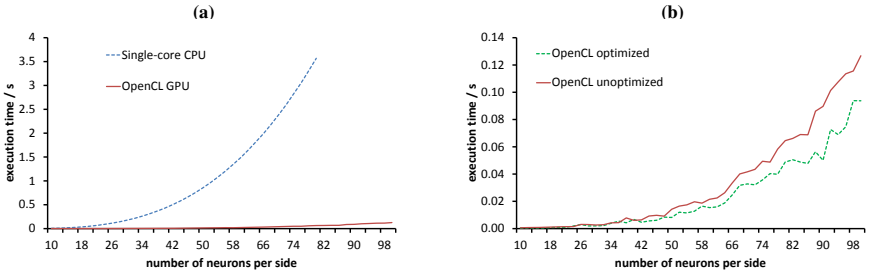


Fig. 3. Execution time for different network sizes for (a) serial CPU vs OpenCL implementation (b) for naive vs. optimized implementation

4 Conclusions

In this paper, we presented an OpenCL implementation of a biologically plausible spiking neural network. We based our neural dynamics on the spike response model that is evaluated in a discrete-time simulation. We compared our naive OpenCL implementation to a single-thread CPU code, as well as to an optimized version of the same code. Our results show that OpenCL provides much faster execution time, thus being a valid platform for future embedded on-board neural processing.

In our test scenario, a network with 1,000,000 neurons and approximately 124,000,000 synapses could be calculated within 93 ms with GPU support. This is sufficient for typical low-framerate image processing scenarios.

Acknowledgments. We thank Lizzie Dawes and John NA Brown for proofreading and constructive comments. This work was supported by the Austrian Science Fund grant FFG 2305537, Lakeside Labs via funding from the EU Regional Development Fund and the KWF under grant KWF 20214|21532|32604, and funding from the ParaDIME Project (FP7/2007-2013 grant 318693).

References

1. Maass, W.: Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. Advances in Neural Information Processing Systems 9 (1997)
2. Pallipuram, V., Bhuiyan, M., Smith, M.: Evaluation of GPU architectures using spiking neural networks. In: 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC), pp. 93–102 (July 2011)
3. Yudanov, D., Shaaban, M., Melton, R., Reznik, L.: GPU-based simulation of spiking neural networks with real-time performance amp; high accuracy. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (July 2010)
4. Nageswaran, J., Dutt, N., Wang, Y., Delbrueck, T.: Computing spike-based convolutions on GPUs. In: IEEE International Symposium on Circuits and Systems (ISCAS 2009), pp. 1917–1920 (2009)
5. Yudanov, D., Reznik, L.: Scalable multi-precision simulation of spiking neural networks. In: 2012 IEEE World Congress on Computational Intelligence (WCCI 2012) (June 2012)
6. Bernhard, F., Keriven, R.: Spiking neurons on GPUs. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 236–243. Springer, Heidelberg (2006)
7. Ferrari, S., Mehta, B., Di Muro, G., VanDongen, A., Henriquez, C.: Biologically realizable reward-modulated hebbian training for spiking neural networks. In: IEEE International Joint Conference on Neural Networks, IJCNN 2008 (IEEE World Congress on Computational Intelligence), pp. 1780–1786 (June 2008)
8. Karimi, K., Dickson, N.G., Hamze, F.: A performance comparison of CUDA and OpenCL. In: CoRR abs/1005.2581 (2010), `http://arxiv.org/abs/1005.2581`
9. Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks 10(9), 1659–1671 (1997)
10. Gerstner, W., Kistler, W.M.: Spiking Neuron Models: Single Neurons, Populations, Plasticity, 1st edn. Cambridge University Press (2002)

FStream: A Decentralized and Social Music Streamer

Antoine Boutet[1], Konstantinos Kloudas[1], and Anne-Marie Kermarrec[1,2]

[1] INRIA Rennes, France
{antoine.boutet,konstantinos.kloudas,anne-marie.kermarrec}@inria.fr
[2] EPFL, Switzerland
anne-marie.kermarrec@epfl.ch

Abstract. Internet streaming services and social networks have drastically changed how people discover and consume music. Existing streaming services allow users to listen to music available on a centrally controlled web infrastructure. However, recent trends, *e.g.* large, inexpensive home storage devices and always on, high-speed broadband connectivity, provide the opportunity for users to collaboratively share their music collections directly, without the intervention of a service provider. In this paper we present FStream, a distributed, social music streaming service. Users contribute to the system by hosting at home a FStream-Box which enables them to have access to their music collection everywhere and at any time. In addition, they can share their collection with friends, discover new tracks thanks to a recommendation system and search for specific tracks in the whole network without using a central entity. FStream provides a fine grained sharing policy allowing the user to have full control over the way she shares her music and manage resource allocation to address traffic spikes and content availability.

Keywords: Social Networks, Recommendation System, Streaming.

1 Introduction

In recent years the way we are consuming music has drastically changed. Aided by technological advances in network access and storage, personal music collections have grown considerably while, at the same time, fit in devices as small as a usb stick. The above, in combination with the integration of music players in mobile devices like smartphones that we carry everywhere, has led to music accompanying many of our everyday activities. The above phenomenon has led to the emergence of numerous Internet services like Spotify[1] or Jango[2] that provide large collections of tracks directly available for streaming (without downloading).

An aspect of music that does not go unnoticed by these services is the social one. Music sharing sites, in their effort to improve their offered services and to engage users, add more and more functionalities based on user interactions. This

[1] http://www.spotify.com
[2] http://www.jango.com

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 253–257, 2013.

includes music recommendation based on collaborative filtering, track rating and playlist sharing. On the other hand, and for the same reasons, social networking sites tend to integrate music sharing functionalities. The recent deal between Facebook and Spotify illustrates the above.

Existing services, allow users to listen to music available on a centrally controlled web infrastructure. Even services like Spotify that are peer-assisted [4], assume the existence of a central index, which serves as the gateway to locate music tracks. However, recent trends such as large, inexpensive home storage devices and always on, high-speed broadband connectivity at low (and decreasing) cost provide the opportunity for users to collaboratively share their music collections directly from home [6,5], without the intervention of a service provider. Hosting a streaming server at home, removes the need for users to install specific software on each one of their devices or download each track before listening to it, as in existing filesharing applications. In addition, an alternative architecture that permits the direct interaction between users, avoids drawbacks of the centralized alternatives: (i) no site-specific constraints on the upload data (e.g. format, size, etc) and no need to upload content to remote data centers; (ii) avoid complex terms of ownership rights (e.g. facebook, instagram) and let users regain control over their data (i.e., what to share, whom to share with); (iii) no advertisement and service restriction (e.g. restricted free access) or restriction on the available music collection (e.g. only albums from one label); (iv) scalability issue (e.g. track unavailable due to social sprike).

In this paper we present FStream, the first distributed, social music streaming network. FStream enables users to have access to their own music collection everywhere and at any time. In addition, they can share their collection with friends, discover new tracks thanks to a recommendation system that brings closer people with similar music tastes, and search specific tracks in the whole network without relying on a central entity. FStream provides a fine grained sharing policy that permits the user to have full control over the way she shares her music. Instead of installing a p2p client, users in FStream use a browser to connect their box called FStreamBox. FStream is hosted at home and takes care of managing all operations for them. In particular, FStream helps users to maintain their social network, compute recommendation based on their music tastes and manage resource allocation to address traffic spikes and content availability.

2 Overview of FStream

FStream resides on user provided resources leveraging their capabilities in storage and bandwidth. In FStream, each user contributes to the system by hosting at home a FStreamBox, which grants access to not only its own music collection but the ones of other members of the service. The only thing required to benefit from the streaming service is a web browser to connect to her machine. All the operations of referencing friends' collections, discovering new nodes and music collections, computing recommendations and content caching are managed

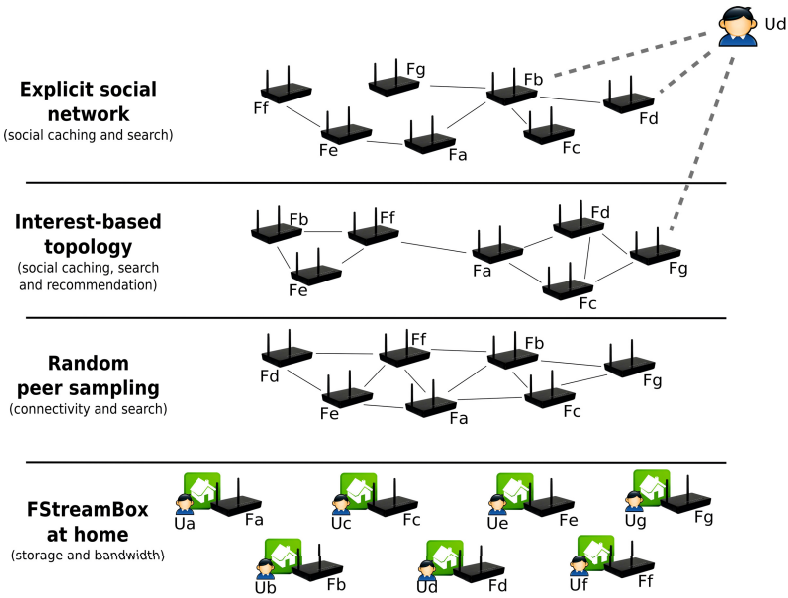


Fig. 1. FStream *architecture*

by the FStreamBox. We based our solution on Subsonic[3], an open-source, web-based media streamer and added several components implementing the additional functionality provided by FStream. This additional functionality includes: (i) social network construction and maintenance, (ii) ownership and sharing restrictions control, (iii) recommendation system, (iv) search mechanism, (v) storage space and social caching.

Figure 1 depicts the architecture of a FStream system. Our system can be seen as composed of four different layers. At the bottom layer are the FStreamBoxes that provide the storage service and are the link between a user and the rest of the system. The second layer relies on a random-peer-sampling (rps) protocol [7] which ensures connectivity between FStreamBoxes by building and maintaining a continuously changing random topology. This random overlay is also leveraged to search items through the system. The upper-layer clustering protocol [8] uses this overlay to provide users with the most similar candidates based on their profiles (*i.e.* short descriptors of their tastes) to form an interest-based topology. This gossip-based clustering protocol, is based on a periodic exchanges of user profiles, on which it applies a similarity metric to cluster users with similar music tastes. To guarantee that this overlay represents the user's most recent "mood", a user's profile contains information on her latest activity on the system (songs that she has been listening lately) and the clustering algorithm updates a user's "neighborhood" dynamically. Finally, the last layer in

[3] `http://www.subsonic.org`

the architecture maintains connectivity with friends through the explicit social network of users. Each user initially connects to her FStreamBox which manages all operations on her behalf. For instance, user d of Figure 1 is connected to her FStreamBox and benefits from social caching from the FStreamBox of both an explicit friend (*i.e.* User b) and an implicit friend on the interest-based topology (*i.e.* User g). Functionalities provided by FStream are explained below.

Social network. As depicted in the architecture mentioned above, users can explicitly declare other users as *friends* in FStream. Declaring a user as a *friend*, is equivalent to granting her access to your music collection. The granted access is subject to the sharing preferences on the owner of the collection has defined (described below) and the data remain stored on the owner's FStreamBox while the two users are presented with the merged indexes of their collections. Listening a track generates a stream from the box where it is stored to the user who consumes it (User d on Figure 1).

Ownership and sharing restriction. FStream enables users to define their sharing preferences at the granularity of tracks. The full collection or part of it can be *public*, *shared only with its social network* or *private*. *Public* means that anybody can view, listen or copy the content while *private* restricts access to only the owner of the collection. Finally, a user is able to restrict access to her collection *only to her social network*, in this case only users declared as friends can view and listen the content.

Recommendation system. As mentioned earlier, one of the objectives of FStream is to discover and connect users that share similar tastes, even in the case that they are not among each other's explicit social network. This is guaranteed by the gossip-based clustering protocol described above that manages to cluster users by exchanging their profiles. Recommendations presented to users are based on the activity of both explicit and implicit *friends*. This functionality permits FStream to work as a recommendation system that "proposes" new content to a given user based on what other users with similar tastes listen. Furthermore, this interest-based topology could be further leveraged in FStream to recommend items to users [1,2] and to perform the social caching.

Search. Users can also launch requests for specific tracks. To answer this need and to locate specific content in the network, FStream benefits from search mechanisms. This mechanism first leverages profile exchanged for the building of overlays. As the profile contains information on the collection, the requested track can be locally present in profile information. Otherwise FStream uses a gossip protocol [3] to localize the content on the system.

Storage Space and Social Caching. FStream relies on a self-sufficient storage infrastructure (no need for central infrastructure) where users contribute storage resources to the service. This contributed storage space is divided in two parts,

the first one is used to store the user's local collection and is controlled by the user, while the second is used by FStream to guarantee content availability and good Quality-Of-Service (QoS). In this respect, it can be seen as a *distributed social cache*. Its *social* aspect relies on the fact that FStream decides what to store and which item to replace in this part according to the music preferences of the owner of the machine, but also the ones of her *social network*. This differentiates FStream's cache replacement policy from classic ones that apply policies like *Least-Recently-Used* or *Least-Frequently-Used*. The purpose of this social cache is twofold. On one hand, it helps with content availability in the face of node disconnection while on the other, it helps the system to serve popular content and face traffic spikes.

3 Summary

Following the evolution on how users consume music, we present the design of FStream, a distributed social music sharing and streaming service based on FStreamBox hosted at each user. FStream enables users to access to their music collection everywhere and at any time and to share it with friends, search for specific tracks and discover new ones thanks to a recommendation system in a fully decentralized manner. At the same time, FStream provides a fine grained sharing policy that permits the user to have full control over the way she shares her music and manage resource allocation.

References

1. Boutet, A., Frey, D., Guerraoui, R., Jégou, A., Kermarrec, A.-M.: WhatsUp Decentralized Instant News Recommender. In: IPDPS 2013, Boston, États-Unis (May 2013)
2. Carretero, J., Isaila, F., Kermarrec, A.-M., Taïani, F., Tirado, J.: Geology: Modular Georecommendation in Gossip-Based Social Networks. In: ICDCS (2012)
3. Kempe, D., Kleinberg, J., Demers, A.: Spatial gossip and resource location protocols. J. ACM (2004)
4. Kreitz, G., Niemela, F.: Spotify – large scale, low latency, p2p music-on-demand streaming. In: P2P (2010)
5. Marcon, M., Viswanath, B., Cha, M., Gummadi, P.: Sharing social content from home: a measurement-driven feasibility study. In: NOSSDAV (2011)
6. Valancius, V., Laoutaris, N., Massoulie, L., Rodriguez, P., Diot, C.: Greening the Internet with Nano Data Centers. In: CoNEXT (2009)
7. Voulgaris, S., Gavidia, D., Steen, M.V.: Cyclon: Inexpensive membership management for unstructured p2p overlays. J. Network Syst. Manage. (2005)
8. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1143–1152. Springer, Heidelberg (2005)

BFT Selection

Ali Shoker and Jean-Paul Bahsoun

University of Toulouse III, IRIT, Toulouse, France
{ali.shoker,jean-paul.bahsoun}@irit.fr

Abstract. This paper presents the first BFT selection model and algorithm that can be used to choose the most convenient protocol according to the BFT user (i.e., an enterprise) preferences. The selection algorithm applies some mathematical formulas to make the selection process easy and automatic. The algorithm operates in three modes: Static, Dynamic, and Heuristic. The Static mode addresses the cases where a single protocol is needed; the Dynamic mode assumes that the system conditions are quite fluctuating and thus requires runtime decisions, and the Heuristic mode uses additional heuristics to improve user choices. To the best of our knowledge, this is the first work that addresses selection in BFT.

Keywords: Byzantine fault tolerance, BFT selection, dynamic switching.

1 Introduction

Byzantine fault tolerance [1] (BFT) is a replication-based approach used to maintain the resiliency of services, often state-machines, against *Byzantine* (i.e., arbitrary) faults in a partially synchronous [1] environments. Many BFT protocols have been introduced so far to maintain safety and liveness in such systems; however, no one-size-fits-all protocol was proposed. A vast discrepancy can be noticed among these protocols which governs their characteristics and performance. This can bring some confusion to BFT users [1] to choose the protocol that is most convenient to their services according to their own demands. Choosing a convenient protocol can be hard when the candidate protocols and their characteristics are numerous. Guerraoui et al. [2] proposed an abortable framework to launch alternating BFT protocols on the same service based on the changes in the underlying system conditions; however, this approach did not introduce any switching policy to run the candidate protocols efficiently and dynamically.

In this paper, we introduce the first BFT selection model and algorithm that automates the selection process of the 'preferred' BFT protocol among a set of candidate ones. The 'preferred protocol' is the one that matches user preferences the most. An evaluation process is in charge of matching the user preferences against the profiles of the nominated BFT protocols considering both: reliability

[1] A BFT user in our context is any enterprise that is choosing a BFT protocol to deploy on its services.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 258–262, 2013.

and performance. The selected protocol is the one that achieves the highest evaluation score. The mechanism is automated via mathematical matrices, and produces selections that are reasonable and close to reality. We explore in this paper the selection model and algorithm. The selection algorithm operates in three modes: Static, Dynamic, and Heuristic. We focus on discussing the Static mode, and we describe the Dynamic and Heuristic modes in [3,4].

Though our model is generic (it may cover any functional and non-functional protocol), we introduce it in the context of BFT for two main reasons. First, to make a first steps towards dynamic switching between existing BFT protocols at runtime, and second, to make it easier for enterprises to select their preferred BFT protocol when BFT is provided as a service, e.g., in clouds.

The rest of the paper is organized as follows. We introduce the selection model and the selection algorithm in Sections 2 and 3, respectively. We address the evaluation in Section 4, and we conclude our paper in Section 5.

2 BFT Selection Model

2.1 Notations and Terms

We define two types of indicators: Key Characteristic Indicators (KCI) and Key Performance Indicators (KPI). KCIs are those properties (with boolean values) of a protocol that indicate its properties, and requirements, e.g., 'the minimum number of replicas needed'. The KCI can strictly decide whether an evaluated protocol could be selected or not. The KPIs are the properties that evaluate the performance of the protocol like throughput, and latency. These values are usually real numbers. KPIs are used to recommend a protocol over the others but, in general, it could not rule out a protocol. In addition, we define the system state by $S = \{s_i = (f_1, f_2, ..., f_j, ..., f_m)\}$ where f_j represents the j^{th} impact factor of the system state and m is the number of considered impact factors. 'Number of clients' and 'message size' are examples of impact factors.

2.2 Selection Model

Consider a service provider (e.g., a cloud vendor) that offers n different BFT protocols along with its provided services (e.g., signed in SLA contract). We define the set of BFT protocols $\psi = \{p_i;\ where\ 1 \leq i \leq n\}$. On the other hand, consider a selection model represented by:$\Sigma = \{\text{Protocol,User,Mode}\}$. Protocol represents the profile of a BFT protocol, User represents the preferences of the user (i.e., the enterprise), and Mode represents the selection mode of the system. Selection occurs through matching the Protocol profile with the User preferences according to the mapping: $f : \Sigma \longmapsto \psi$; this yields the 'preferred' protocol among all competing protocols. Here we define the 'preferred' protocol:

Definition 1. *A protocol p_i with profile $Protocol_i$ is called the 'preferred' protocol among a set of candidate protocols ψ with respect to a specific user with preferences $User_j$ if and only if according to an evaluation function $e : \Sigma \longmapsto \Re$, $e(Protocol_i, User_j, \phi)$ is maximal.*

The interpretation of Protocol, User, and Mode is as follows:

Protocol. Each protocol has a profile: Protocol=$\{A_P, A_U, B_P, B_V\}$. $A_P = (\alpha_1, \alpha_2, ..., \alpha_a)$ is a vector of a KCIs. A_U represents the vector of the default weights of these KCIs: $A_U = (u_1, u_2, ..., u_a)$. $B_P = (\beta_1, \beta_2, ..., \beta_b)$ is a vector of b KPIs and, finally, B_V represents the vector of the default weights of these KPIs: $B_V = (v_1, v_2, ..., v_b)$.

User. Each user, e.g., an enterprise, defines his preferences in User=$\{U, V, M\}$, where U (resp., V) is a vector of user defined weights corresponding to the KCIs (resp., KPIs) of the Protocol's preference A_P (resp., B_P). M defines the mode required by the user, i.e, either Dynamic, Static, or Heuristic.

Mode. The selection can occur in three different modes: Static, Dynamic, or Heuristic. In the former, the selection occurs only once, i.e., at the time the BFT user requires a service; afterwards, the user does not change his selection (i.e., the used protocol) until the system is halted/rebooted and, thus a new selection is provoked. On the other hand, the Dynamic mode makes the system react dynamically to the changes of the system state. This mode allows the system to adapt to the upcoming conditions at runtime and hence the user will be using multiple alternating protocols. The Heuristic mode uses some heuristics to adjust the preferences of the user, especially V, to improve his choices in some cases.

3 Selection Mechanism

The selection mechanism of the preferred protocol according to the user preferences is achieved through computing the evaluation scores E of the competing protocols, and then electing the protocol that corresponds to the maximum score. For any state s, and protocol $p_i \in \psi$ that has an evaluation score $E_{i,s}$; a protocol p_{pref} is chosen according to Equation 1:

$$p_{pref} = p_i,\ s.t.\ E_{i,s} = \max_{1 \leq j \leq n} E_{j,s}. \tag{1}$$

If the mode of the system is Dynamic or Heuristic, the KPIs are computed at runtime, and the system chooses the protocol that has the highest evaluation score E among all protocols to launch it in the next phase. To make computations easier, we define a new operator, i.e., the OR product $\dot{\vee}$.

Definition 2. *Consider two boolean matrices $A \in \{0,1\}^{n \times l}, B \in \{0,1\}^{l \times m}$ with entries a_{ij}, and b_{ij}, respectively. The OR product $A \dot{\vee} B$ is a matrix $C = A \dot{\vee} B \in \mathbb{N}^{n \times m}$, where its elements are defined by: $c_{ij} = \sum_{k=1}^{m} a_{ik} \vee b_{kj}$. The operator $\vee$ is the logical OR operator.*

$$\begin{cases} E = C \circ P \\ where\ C = \left\lfloor \frac{1}{a}.(A\ \dot{\vee}\ (e_n - U)) \right\rfloor \\ and\ P = B^{\pm}.(V \circ W). \end{cases} \tag{2}$$

The evaluation score E is calculated according to the formulas introduced in Equation 2. The evaluation matrix E is the Schur product of the KCI matrix C and the KPI matrix P. C represents the part of the evaluation that deals with the KCIs of the profiles of the protocols; whereas, P represents the evaluation part that deals with the KPIs. E is calculated after computing the values of C and P. If the mode of the system is Dynamic or Heuristic, then E may change at runtime as P changes.

The KCI matrix $C = \left\lfloor \frac{1}{a}.(A \dot{\vee} (e_n - U)) \right\rfloor$ matches the user preferences against the profiles of different protocols. a represents the number of KCIs considered. The operator $\lfloor\ \rfloor$ is the integer value operator (it is sometimes indicated by [ ] too). The operator $\dot{\vee}$ was defined in Definition 2. Matrix A represents the profiles of the protocols. The dimension of A is $n \times a$; where n is the number of candidate protocols and a is the number of KCIs considered in the evaluation. Each row of the matrix represents a KCI vector profile of a protocol. Matrix U represents the preferences of the user. According to this matrix, the protocols that satisfy all user requirements will be considered for selection (i.e., will continue the competition). On the contrary, the protocol that lacks a single property among those demanded by the user will be out of selection. The column matrix e_n is a unit matrix is to invert the values of the matrix U to $-U$. After defining the matrices A and U, the computation of C becomes straightforward.

Matrix P is used to complete the selection process by considering the KPIs of the protocols, seeking better performance. The KPI matrix P is defined in the formula: $P = B^{\pm}.(V \circ W)$. $B^{\pm}$ is a normalized version of another matrix B that represents the KPI profiles of each protocol. Each profile is presented in one row. B and $B^{\pm}$ have the same dimension $n \times b$ where n is the number of protocols and b is the number of KPIs considered. The entries of the matrix $B^{\pm}$ are denoted by $\beta^{\pm}$ and are calculated from the entries of B that are denoted by β. We say that a KPI has the property Tendency='high', if a higher value means better evaluation score E, e.g., throughput; this KPI is denoted by β^{+}. On the contrary, a KPI of type β^{-} has the property Tendency='low', e.g., latency, and a higher KPI value means worst evaluation score E. Suppose the number of β-KPIs is b, then the matrix B can be divided into b column matrices (i.e., vectors): $B_1, B_2, B_i, ...,$ and B_b. Let the maximum (resp., minimum) value of the entries of each vector B_i be max_i (resp., min_i). Then, the entries of the matrix $B^{\pm}$ can be calculated according to Equation 3:

$$\begin{cases} \beta^{+}_{ji} = 1 - \dfrac{max_i - \beta_{ji}}{max_i - min_i}; \\ \beta^{-}_{ji} = 1 - \dfrac{\beta_{ji} - min_i}{max_i - min_i}; \\ where\ i \leq b\ and\ j \leq n. \end{cases} \tag{3}$$

Matrix V represents the KPI user preferences used to recommend a protocol. V is a column matrix of dimension $b \times 1$, where b is the number of KPIs considered in the evaluation. The entries of this matrix follow two constraints: (1) all entries $\in$ [0,10], and (2) their sum $\sum_{i=1}^{b} v_{i1} = 10$. Matrix W is a column matrix used in

the Heuristic mode only. W is important to adjust the user preferences given in V by considering the system state to improve his choice according to predefined heuristic rules. If the mode is Static, then the entries of W are equal to 1, i.e., W=e_b.

4 Evaluation

To evaluate our approach we have considered seven existing BFT protocols by listing their different KCIs like the number of replicas needed, speculative or not, tolerate malicious clients or not, etc. Also we have considered three KPIs: throughput, latency, and capacity. The KPI values are estimated based on the message exchange patterns of the different protocols. Our mechanism gave selection results as expected according to many user preferences we have chosen. The mechanism minimizes the complexity of selection significantly. Due to lack of space, we do not reveal our examples and results in this paper, but we encourage the reader to read our extended papers in [3,4].

5 Conclusion

We presented a BFT selection model and algorithm to automate the selection of the 'preferred' BFT protocol according the preferences defined by the BFT user, i.e., an enterprise. This is useful in large services that provide BFT as a service, and in fluctuating systems that require dynamic runtime switching of BFT protocols as the underlying system conditions change. We consider three modes: (1) Static mode: where the user chooses a protocol only once; he can only change it when the service is rebooted. (2) Dynamic mode: which allows the user to multiple protocols, where a running protocol can be stopped and another protocol is launched after performing selection process. The intuition is that the performance of protocols differ as the underlying system state changes, and thus adapting to the new state is required. (3) Heuristic mode: this mode is similar to the Dynamic mode; however, it allows to modify the weights (i.e., preferences) chosen by the user as the system state changes using some predefined heuristics. This paper focused on the Static mode, while future work addresses the other interesting modes: Dynamic and Heuristic.

References

1. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4), 398–461 (2002)
2. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 bft protocols. In: EuroSys 2010: Proceedings of the 5th European Conference on Computer Systems, pp. 363–376. ACM, New York (2010)
3. Shoker, A., Bahsoun, J.P.: Bft selection. Technical report, IRIT (2013)
4. Shoker, A.: Byzantine fault tolerance: From static selection to dynamic switching. PhD thesis, IRIT (2012)

Modeling of Human Head Interaction with Planar Antenna for Multi Standard Cellular Phones

Ahmed Zakaria Manouare[1], Abdelilah Ghammaz[1], Abdelaziz El idrissi[1], and Saida Ibnyaich[2]

[1] Faculty of Science and Technology, Cadi Ayyad University, LSET Laboratory, Marrakech, Morocco
ahmedzakaria.manouare@gmail.com, {a.ghammaz,a.elidrissi}@uca.ma
[2] Faculty of Sciences Semlalia, Cadi Ayyad University, LEI Laboratory, Marrakech, Morocco
s.ibnyaich@ucam.ac.ma

Abstract. Many wireless communication devices are used with their antennas in close proximity of the human head. This fact changes the antenna characteristics and affects the communications performance on one hand, and causes the deposition of microwave energy in the user's head, on the other hand. The amount of power absorbed by the human body is limited to a given maximum value, according to the two standards IEEEC95.1 and EN 50360. In this paper the design of a multiband compact antenna for integration into the multifunction mobile phones is presented. The specific absorption rate (SAR) of the planar antenna is calculated.

Keywords: Dosimetry, patch antenna, multi standard(GSM900/1800), specific anthropomorphic mannequin (SAM), specific absorption rate (SAR).

1 Introduction

Growing consumer demand for multifunctional mobile handsets has seen an increase in the development of small multi-band antennas. It seems to be highly desirable to develop lightweight and single feed mobile antenna, which can be used simultaneously in different frequency ranges and for different mobile services. The ability to cover a number of communications bands with one small antenna benefits both the end user and the manufacturer as it reduces the cost and complexity of the antenna system [1].

Starting with a simple microstrip antenna, several techniques must be simultaneously applied to achieve multiband performances: use of slot (in our study we use slot).

Interaction of handset antennas with human head is a great consideration in cellular communications. The user's head influence on the antenna radiation pattern. Furthermore, thermal effect, when tissues exposed to unlimited electromagnetic energy, can be a serious health hazard. So standard organizations have set exposure limits in terms of the specific absorption rate (SAR) [1-3].

In this paper, a new miniaturized multiband antenna which supports GSM 900 and GSM 1800 communication standard is proposed. In order to achieve the objective of

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 263–267, 2013.

designing a multiband patch antenna, we have begun with a simple model of patch on the GSM 1800 band frequency and for second band frequency GSM 900; we add a rectangular slot. The final geometry of the presented antenna is evaluated in the presence of the head model. In the addition, the SAR quantity is computed for this antenna.

2 Antenna Design and Results Discussion

The parametric study is carried to optimize the antenna structure and provide more information about the effects of the essential design parameters. The antenna performance is mainly affected by geometrical parameters such as adding slots.

2.1 Model of the Proposed Antenna

As starting model, Figure 1 shows the three dimensional view of the proposed antenna. The antenna is fed by a 50Ω microstrip line and use the Rogers RT/duroid 5880(tm) as substrate with a thickness of h=1.6 mm and a relative permittivity ε_r=2.2. The substrate size of the proposed antenna is 70x74 mm².

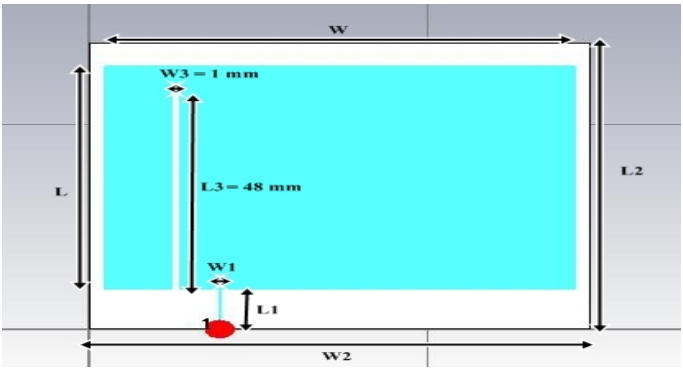


Fig. 1. Geometry of the proposed antenna

Table 1. Antenna size details

Parameter	*Value*
L	55 mm
W	70 mm
L1(feed line)	9.547 mm
W1(feed line)	0.50 mm
L2	70 mm
W2	74 mm
L3(slot)	48 mm
W3(slot)	1 mm

2.2 Results and Discussion

The slots force the surface currents to meander, thus artificially increase the antenna's electrical length without modifying its global dimensions, which results a decrease in resonant frequency.

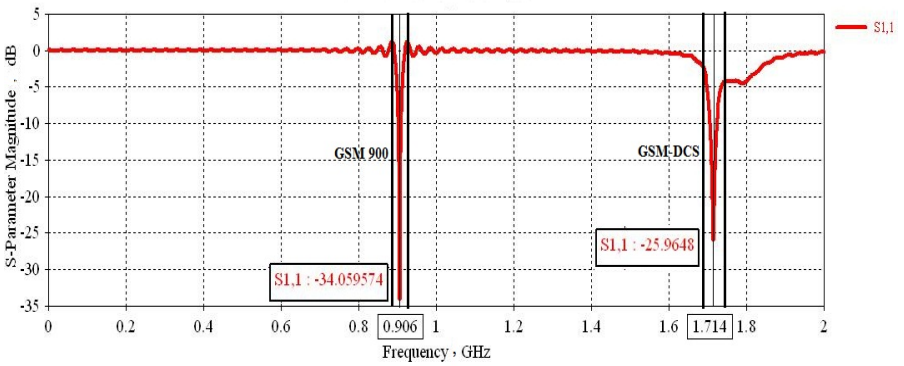


Fig. 2. Return loss coefficient as a function of frequency

Figure 2 shows the simulated return loss coefficient, as we can see; the antenna model presents a good resonance in GSM 900 and GSM 1800.

3 The Antenna Dosimetry

For the cellular phone compliance, the SAR value must not exceed the exposure guidelines [4-6]. Some numerical results have implied that the peak 1g averaged SAR value (SAR_{1g}) may exceed the exposure guidelines when a portable telephone is placed extremely close to the head [4], [7,8].

The SAR is defined as the absorbed RF energy by unity of volume, and its dimensions are W/Kg or mW/g. The formulation of SAR is defined as:

$$SAR = \frac{d}{dt}\left(\frac{dW}{dm}\right) = \frac{\sigma|E|^2}{2\rho} \qquad [4] \qquad (1)$$

Where E (V/m), σ (S/m) and ρ (Kg/m^3) are the electric field, conductivity, and mass density in the head, respectively.

3.1 SAR Calculation in the Head

In this section, a model is built of the proposed antenna next to the left ear on SAM model; the shape of the head model is similar with real human head shape. The head model consists of homogenous dielectric representing the human tissue with relative permittivity ε_r=41.5 and electric conductivity σ=0.97S/m [9, 10] shows the human head. The antenna was arranged parallel to the z axis.

The SAR limit for mobile terminal equipment is 2W/Kg and measured as an average over a 10 gram cube of tissue. In the USA and Canada, mobile terminal equipment must comply with the 1.6 W/Kg SAR limit measured as an average in a 1 gram cube of tissue.

The simulations approach consists on calculating the level of SAR absorbed by the phantom (SAM) in the case of a phone composed by the multiband patch antenna when a mobile phone emits at the maximum power.

3.2 SAR Determination for GSM 900 and GSM 1800 Frequencies Bands

In this section, we simulated the antenna in the presence of human head model for 906 MHz and 1.714 GHz. The SAR distributions over the head in 1 gram cube of tissue are illustrated respectively in figures 3 and 4.

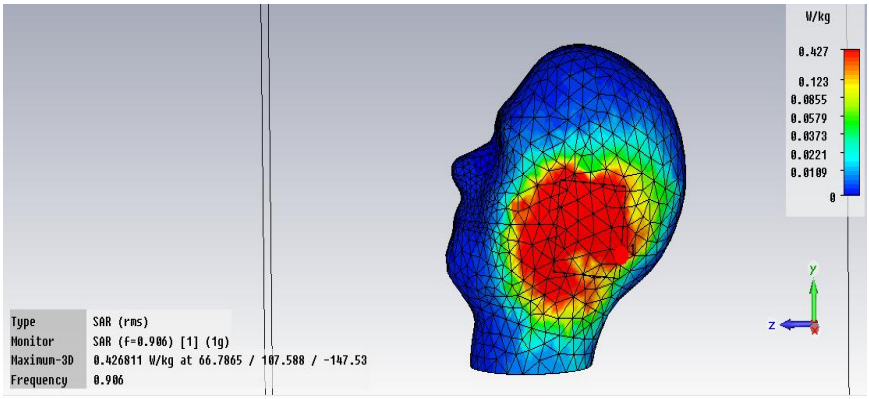


Fig. 3. SAR distribution over the head in a 1 gram cube of tissue for 906MHz

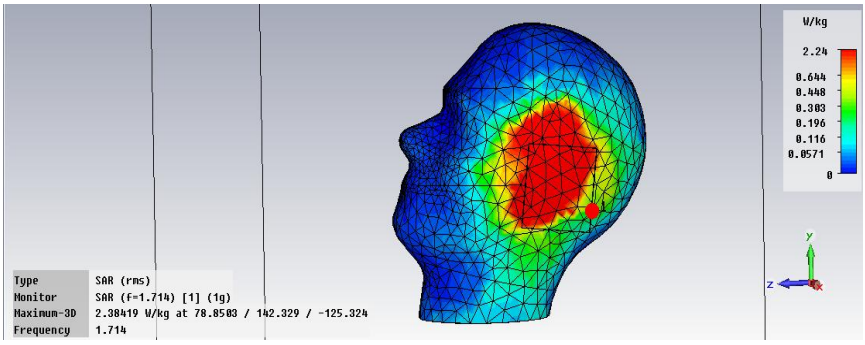


Fig. 4. SAR distribution over the head in a 1 gram cube of tissue for 1.714 GHz

Figures 3 and 4 presents the SAR distribution in the human head for GSM 900 and GSM 1800. The test of specific absorption rate for 906 MHz and 1.714GHz is given in Table 2. We conclude that the SAR value (SAR level) depends on excitation frequency.

Table 2. SAR calculation for 906 MHz and 1.714GHz

Frequency	*SAR value*	*Parameter*	*Value*
906 MHz	1gram cube of tissue	SAR_{1g} (W/Kg)	0.426
906 MHz	10 gram cube of tissue	SAR_{10g} (W/Kg)	0.226
1.714GHz	1gram cube of tissue	SAR_{1g} (W/Kg)	2.384
1.714 GHz	10 gram cube of tissue	SAR_{10g} (W/Kg)	1.552

4 Conclusion

In this paper we firstly present a new multiband patch antenna with slot and secondly we calculate the specific absorption rate (SAR) for two frequencies bands (GSM 900 and GSM 1800): know the influence of the electric field inside the biological tissues of the human head (SAM model). It is possible to conclude that a large part of the absorbed power is concentrated in the tissues in the vicinity of the antenna.

References

1. Ben Ahmed, M., Bouhorma, M., Elouaai, F., Mamouni, A.: Low SAR planar antenna for multi standard cellular phones. Eur. Phys. J. Appl. Phys. 53, 33604 (2011)
2. ICNIP, Health Phys. 74, 494 (1988)
3. IEEE standard for safety levels with respect to human exposure to radio frequency electromagnetic fields. 3kHz to 300GHz, IEEE Std C95.1^{TM} -2005. IEEE, New York (2005)
4. Hwang, J.N., Chen, F.C.: Reduction of the Peak SAR in the Human Head With Matemateriels. IEEE Transactions on Antennas and Propagation 54(12) (December 2006)
5. IEEE Standard for Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields, 3 kHz to 300 GHz, IEEE C95.1-1991. Institue of Electrical and Electronics Engineers, Inc. New York (1992)
6. Guidelines on limits of exposure to radiofrequency electromagnetic fields in the frequency range from 100 KHz to 300 GHz. Health Phys. 54(1), 115–123 (1988)
7. Wang, J., Fujiwara, O.: FDTD computation of temperature rise in the human head for portable telephones. IEEE Trans. Microwave Theory Tech. 47(8), 1528–1534 (1999)
8. Kuo, C.M., Kuo, C.W.: SAR distribution and temperature increase in the human head for mobile communication. In: IEEE–APS Int. Symp. Dig., Columbus, OH, pp. 1025–1028 (2003)
9. Beard, B.B., Kainz, W., Onishi, T., et al.: Comparisons of computed mobile phone induced SAR in the SAM phantom to that in anatomically correct models of the human head. IEEE Transaction on Electromagnetic Compatibility 48(2), 397–407 (2006)
10. Ae-Kyoung, L., Hyung-Do, C., Jae-Ick, C.: Study on SARs in Head Models With Different Shapes by Age Using SAM Model for Mobile Phone Exposure at 835 MHz. IEEE Transactions on Electromagnetic Compatibility 49(2), 302–312 (2007) ISSN: 0018-9375

Modeling the Cut-off Frequency of Acoustic Signal with a Fuzzy Logic System

Youssef Nahraoui[1], Elhoucein H. Aassif[1], Rachid Latif[2], and Gérard Maze[3]

[1] LMTI, Faculty of sciences, IbnZohr University, Agadir, Morocco
nahraoui21y@yahoo.com, elhoucein_aassif@hotmail.com
[2] ESSI, National School of Applied Sciences, IbnZohr University, Agadir, Morocco
latif_rachid@yahoo.fr
[3] LOMC, Université du Havre, Institut Universitaire de Technologie,
Place Robert Schuman, 76610 Le Havre, France
Gerard.maze@univ-lehavre.fr

Abstract. In this paper, an AdaptativeNeuro-Fuzzy Inference System (ANFIS) is developed to predict the cut-off frequencies of the symmetric and the anti-symmetric circumferential waves (Ai, i=1) propagating around an elastic aluminum cylindrical shell of various radius ratio b/a (a: outer radius and b: inner radius) for an infinite length cylindrical shell excited perpendicularly to its axis. The Wigner-Ville distribution (WVD) is used like a comparison tool between the cut-off frequencies calculated by the analytical method and that predicted by the neuro-fuzzy techniques for aluminium or a stainless steel, or cooper tube. A good agreement is obtained between the output values predicted using ANFIS and those computed by the proper modes theory.

Keywords: Fuzzy logic, ANFIS, time-frequency, Acoustic scattering, acoustic circumferential waves, cut-off frequency, cylindrical shell.

1 Introduction

The present paper is especially concerned with the soft computing technique such as Adaptative Neuro-Fuzzy Inference System (ANFIS). The ANFIS model is able to predict the cut-off frequencies of the symmetric and the anti-symmetric circumferential waves (S_i and A_i, i=1,…) for aluminum cylindrical shell of various radius ratio b/a. The radius ratios used, in this paper, are between 0.4 and 0.99. The cut-off frequencies values determined using the ANFIS model are compared with those determined from the time-frequency images of Wigner-Ville to validate the robustness of the model proposed[1].

2 Backscattering Response from a Cylindrical Shell

Fig. 1 shows the cylindrical coordinate orientation and the direction of a plane wave incident on an infinitely long cylindrical shell in a fluid medium. The fluid (1) inside

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 268–273, 2013.

the shell has a density of ρ1 and propagation velocity $c1$. In general, the outer fluid (2) will be different and is described by the parameters ρ and c see table 1.

The backscattered complex pressure P_{diff} by a cylindrical shell in a faraway field ($r>>a$) is the summation of the incident wave, the reflective wave①, surface waves tell shell waves ② (whispering Gallery, Rayleigh, ...) and Scholte waves (A) ③ connected to the geometry of the object (figure 1). The waves ② and ③ are the circumferential waves. For these waves one distinguishes the waves A, the symmetric wave S1, and the anti-symmetric wave A1.

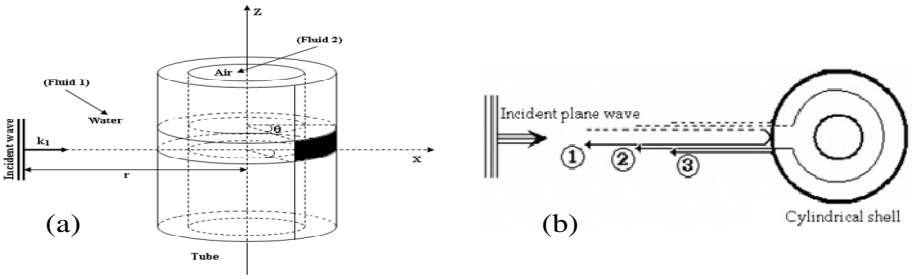


Fig. 1. (a) Geometry used for formulating the sound backscattering from a cylindrical shell. (b) Mechanisms of the formation of echoes showing the specular reflection ① and shell waves ② and Scholte wave (A)③.

The module of the backscattered complex pressure in a faraway field is called form function. This function is obtained by the relation [2]

$$\left|P_{diff}(\omega)\right| = \frac{2}{\sqrt{\pi k r}}\left|\sum_{n=0}^{N_{\max}} \varepsilon_n (-1)^n \frac{D_n^{(1)}(\omega)}{D_n(\omega)}\right| \tag{1}$$

where $\omega = 2\pi f$ is the angular frequency, k the wave number with respect to the wave velocity in the external fluid and. $D_n^1(\omega)$ and $D_n(\omega)$ are determinants computed from the boundary conditions of the problem . The function $H_n^{(1)}$ is the Hankel function of the first kind.

The Neumann factor (ε_n =1, if n=0; ε_n =2, if n>0), k=ω/c is the incident wave number and c is the phase velocity in water.

The physical parameters used in the calculation of the backscattered complex pressure are illustrated in table 1.

Table 1. Physical parameters

	Density ρ (kg/m^3)	Longitudinal Velocity c_L (m/s)	Transverse Velocity c_T (m/s)
Aluminum	2790	6380	3100
Water	1000	1470	-
Air	1.29	334	-

The figure 2 shows the module of the backscattered complex pressure in function of the reduced frequency *ka* (without unit) given by:

$$ka = \frac{\omega a}{c} = \frac{2\pi}{c(1-\frac{b}{a})} f\, d \tag{2}$$

where$d=a-b$ is the thickness of a cylindrical shell and f is the frequency of resonance of a wave in Hz.

The temporal signal response $P(t)$ of a cylindrical shell is computed by taking the Inverse of Fourier Transform of the module of the backscattered complex pressure:

$$P(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} h(\omega) P_{diff}(\omega)\, e^{-i\,\omega\, t} d\omega \tag{3}$$

Where $h(\omega)$ is a smoothing window.

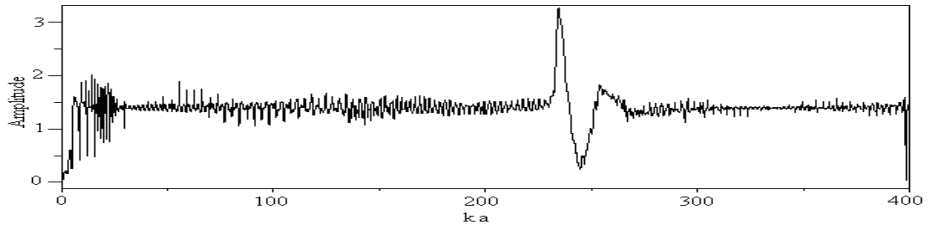


Fig. 2. Module of the backscattered pressure for an infinite aluminum cylindrical shell with air-filled cavity of radii radio b/a=0.95

3 Dispersion and Cut-off Frequency Determined Using the Proper Modes Theory

The cut-off frequencies values are given by equation (4) [1]:

For *A1* mode:
$$(ka)_c^{A1} = \frac{\pi}{c(1-\frac{b}{a})}.c_T \tag{4}$$

The calculated values, of the cut-off frequencies are given in table 2.

4 Dispersion Analysis Using Time-Frequency Image

The Wigner-Ville distribution (*WVD*) of the real signal $x(t)$ is defined by [3,4] :

$$WV_x(t,\nu) = \int_{-\infty}^{+\infty} x(t+\frac{\tau}{2}) x^*(t-\frac{\tau}{2}) e^{-i2\pi\nu\tau} d\tau \tag{5}$$

The Smoothed Pseudo Wigner-Ville (*SPWV*) of the real signal $x(t)$is defined by[1]:

$$SPWV_x(t,f) = \int_{-\infty}^{+\infty} \left|h(\frac{\tau}{2})\right|^2 \int_{-\infty}^{+\infty} g(t-u)\;\; x(u+\frac{\tau}{2}) \;\;\; x^*(u-\frac{\tau}{2}) \;\; \exp(-2\,j\pi) \tag{6}$$

The smoothing windows $g(t)$ and $h(t)$ are introduced into the *SPWV* definition in order to allow a separate control of interferences either in time (g) or in frequency (h).

Figure 3 represent the time-frequency image for example of the anti-symmetric circumferential wave *A1* for aluminum cylindrical shell of radius ratio *b/a=0.95*. The time-frequency image show the cut-off frequency as the intersection point of the asymptotic trajectory of the anti-symmetric wave *A1* and the axis of frequencies figures 3. The values of the cut-off frequency $(ka)_c$ obtained by SPWV are presented in table 2 for various radius ratios.

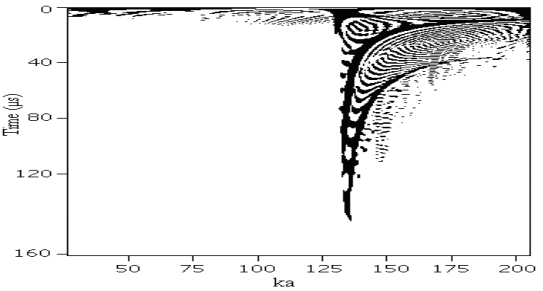


Fig. 3. SPWV of backscattered signal for the aluminium cylindrical shell of radii radio *b/a=0.95* (Anti-symmetric circumferential wave *A1*, *130<ka<200*)

5 Materials and Method

5.1 Adaptive Neuro-Fuzzy Inference System Architecture

The Adaptive Network-based Fuzzy Inference System (ANFIS) is developed by Jang in 1993 [5]. This model use neuro-adaptive learning technique that is a combination of Last squar error and the back-propagation gradient descent algorithm. Given an input/output data set, this allows Fuzzy system to learn from the data they are modelled.

The corresponding equivalent ANFIS architecture is as shown in figure 4. The system architecture consists of five layers, namely; fuzzy layer, product layer, normalized layer, fuzzy layer and total output layer. the theoretical studies show a relationship between input and output of each layer in ANFIS.

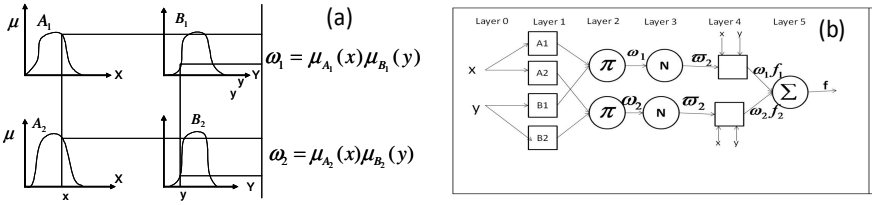


Fig. 4. (a) First-order Sugeno fuzzy model, (b) ANFIS architecture

The constructed adaptive network in figure 4 is functionally equivalent to a fuzzy inference system. The basic learning rule of *ANFIS* is a combination of last squar error and the back-propagation gradient descent.

6 Results and Discussion

The performance of *ANFIS* models for training and testing data sets were evaluated according to statistical criteria such as, coefficient of correlation *R*, *MAE*, *MRE*, *SE*.

In this work we tried to play on the number of rules and the number of epochs we have observed that the error values of our models decrease more than the number of rules, and the number of epochs is increases. The results of the measured errors are presented in figures 5 for the circumferential wave *A1* Table 2 show that the results obtained by the fuzzy system method are in good agreement with those determined from the results calculated using the proper modes theory of resonances, and they are better to those determined manually from the time-frequency of Wigner-Ville images.

Table 2. Results of the cut-off frequencies of mode *A1*obtained by the ANFIS model, the proper modes theory and by the time-frequency of Wigner-Ville images

Cylindrical shell	Cut-off frequencies $(ka)_c$ Computed using	Determined using	Determined using
	PMT	*ANFIS*	*SPWV*
b/a=0.9	66.21	66.16	66.0±0.3
b/a=0.95	132.43	132.59	132.0±0.3
b/a=0.97	220.72	221.32	221.0±0.2

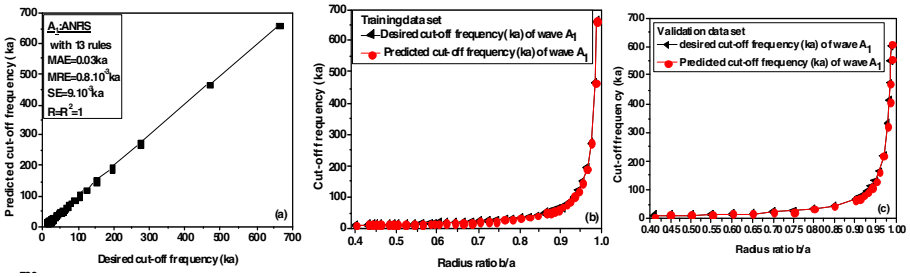


Fig. 5. (a) Correlation of desired versus *ANFIS* values of cut-off frequency of anti-symmetric wave A_1 and with training data set, (b) Cut-off frequency as a function of radius ratio of aluminum cylindrical shell on training data set and (c) Cut-off frequency as a function of radius ratio of an aliminum cylindrical shell on validation data set.

7 Conclusion

The main aim of this work was to train an ANFIS model to predict cut-off frequency with the minimum of input data. Results show that the trained model can be used as

an alternative way in the modelling behaviour system. This fuzzy logic model taking into account some characteristics of the tube is developed in order to predict the cut-off frequency for various types of circumferential waves A_1. In this article, this model is applied to aluminum tubes, can be used also to predict the evolution of the group and phase velocities according to the frequency. It also can constitute a help for the estimate of various parameters of a tube starting from the characteristics of which it is disposed. This article can be used as a new tool for characterization of an elastic tube. The use of the fuzzy logic allows one to determine automatically and with good precision the reduced cut-off frequency of an antisymmetric wave propagating around the tube. The R^2 value in fig is about 1, which can be considered as very satisfactory.

References

1. Latif, R., Aassif, E., Moudden, A., Decultot, D., Faiz, B., Maze, G.: Determination of the cut-off frequency of an acoustic circumferential wave using a time-frequency analysis. J. NDT&E Int. 33, 373–376 (2000)
2. Maze, G., Ripoche, J., Derem, A., Rousselot, J.L.: Diffusion d'une onde ultrasonore par des tubes remplis d'air immergés dans l'eau. Acustica 55, 69–85 (1984)
3. Haumesser, L., Décultot, D., Léon, F., Maze, G.: Experimental identification of finite cy-lindrical shell vibration modes. Journal of the Acoustical Society of America 111(5), 2034–2039 (2002)
4. Aassif, E., Latif, R., Decultot, D., Maze, G., Faiz, B., Moudden, A.: Time-frequency analysis of the complex pressure scattered by immersed tubes. In: 3rd Int. Conf., Acoust. Vibratory Surveillance Methods Diagnostic Techniques, Centre Technique des Industries Mécaniques
5. Jang, J.-S.R.: ANFIS: Adaptive-network-based fuzzy inference systems. IEEE Trans. Syst. Man Cybern. 23(3), 665–685 (1993)

Bitbox: Eventually Consistent File Sharing

Erwan Le Merrer, Nicolas Le Scouarnec, and Gilles Straub

Technicolor, France

Abstract. Bitbox is an application that synchronizes distributed repositories of data. It can be used as a backup or sharing application similarly to popular cloud-based storage systems. Bitbox supports arbitrary and changing topologies, thus allowing residential gateways to be used as caches for synchronizing nomadic devices that connect only periodically. In this article, we describe the data-structure and algorithms powering Bitbox. We prove its correctness by showing that its synchronization scheme achieves *strong eventual consistency*.

1 Introduction

To share, backup and access their content from all their devices (e.g., smartphones, tablets, laptops), people often rely on manual management of files (*e.g.*, sending photos by email or backing them up manually). However, such process is error-prone (*e.g.*, overwriting the newest files with older ones or forgetting files during backups). Sharing, backing up and providing pervasive access can all be boiled down to the problem of synchronizing multiple copies. Indeed, if a user updates some content, she wants her backups to be updated, her friends to receive updates of the shared content, and all her other devices to be updated with the new content in a transparent manner [1].

Current systems (*e.g.*, Dropbox [2]) require to synchronize against a central set of tightly synchronized servers. All synchronizations rely on these servers, thus requiring the clients to be frequently connected to them. A decentralized synchronization scheme furthermore allows to take into account the specificities of the network topology. For example, in home networks, the Internet gateway could be used as a cache since it stands between the slow Internet connection and the fast LAN; two devices can then synchronize their updates way faster than by leveraging remote servers.

In this paper we propose Bitbox, a distributed synchronization application. Bitbox relies on the concept of *convergent replicated data types* [3] for correctness. We describe the application and its core algorithms in Section 3 and give a proof that it correctly achieves strong eventual consistency in Section 4.

2 Background on Related Data-Synchronization Tools

Dropbox [2] partially leverages local connections by a feature called LanSync that allows downloading (*i.e.* reading) content from local devices instead of downloading from the cloud. However, this functionality does not support uploading

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 274–278, 2013.

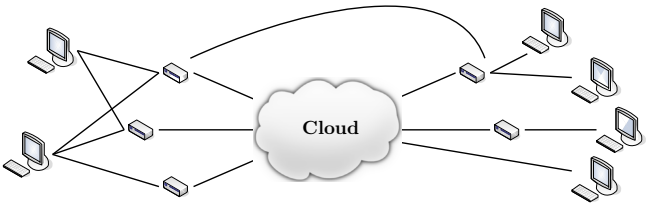


Fig. 1. Bitbox allows arbitrary synchronization topologies

(*i.e.* writing) since the central server is mandatory for it is in charge of maintaining the reference state.

To allow for complex replication policies facing device heterogeneity, Anzere [4] leverages a centralized and well-provisioned node among the user's devices. That special node is in charge of hosting the system's state and runs a conflict solver. We design symetric roles in our system, for less advanced policies.

Bitbox is related to distributed version control tools (*e.g.*, Git [5] or Mercurial), which are replacing traditional central version control (*e.g.*, SVN or CVS). These new tools are designed to allow arbitrary workflow departing from the single centrally managed version. They keep track of the history including all the successive versions of the content of each file, and assist the developer in performing semi-automatic merges between divergent versions. Bitbox differs in that it stores only the latest content of files (which are assumed to be large binary files such as videos or photos), and is designed so as to allow frequent automatic and seamless merges of divergent repositories.

3 The Bitbox Core

Bitbox is an application that keeps various repositories in sync. To this end, whenever a repository synchronizes against another, it lists the changes that occurred remotely since the repository was last synchronized, and applies them locally to reach an equivalent state.

To be able to list these changes, Bitbox keeps track of past states of the repository. A particular state is identified by a version number and consists in a set of file descriptors $\langle$*file name*, *file hashes*, *unique file id*$\rangle$, which is later referred as *content*. This is a compact representation since file hashes are much smaller than the actual file content. The unique file id is generated by the device which detects that a file has been added (*e.g.*, device identifier concatenated with a local timestamp). The state's version number allows distinguishing two states having the same content but having observed modifications (*e.g.*, a file is removed and then added back). The version number is also a compact representation of a state, allowing Bitbox to trivially detect whether a remote repository does not contain new changes.

Whenever a repository synchronizes against another, Bitbox needs to detect their last common state. To this end, the states are organized in a directed

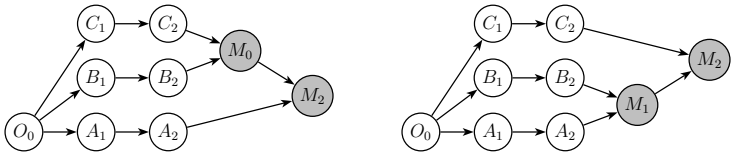


Fig. 2. The Merge operation builds a version number not directly from previous versions but from the previous update (P_U) of versions being merged. Here, $\mathrm{P}_U(M_0) = \{C_2, B_2\}$ and $\mathrm{P}_U(A_2) = \{A_2\}$. As a consequence, the order of Merges has no impact on the version number (*i.e.*, $A2 \oplus (B_2 \oplus C_2) = (A_2 \oplus B_2) \oplus C_2$).

acyclic graph (DAG) as depicted on Figure 2. An edge from state A_1 to state A_2 exists if and only if A_2 results directly from a user's action on A_1, or A_2 results directly from the merge of A_1 with some other state. The last common state of two repositories is their common ancestor in the DAG. Bitbox only considers deletion and addition of files since renaming and updating files can be expressed as a combination of deleting and adding. If O_0 is the last common state between the local state B_2 and the remote state C_2, then Bitbox applies locally all deletions that occurred between O_0 and B_2, and all additions that occured between O_0 and B_2. There cannot be conflicts on a local addition and a remote deletion; indeed, a file is either in O_0 in which case it can be deleted, or not in O_0 in which case it can be added. Furthermore, two deletions applied to the same file in O_0 are necessarily identical.

In some rare cases, conflicts may be observed between additions. They are resolved seamlessly as explained hereafter. In the first scenario, it can happen that users concurrently add two different files under the same file name, at two different devices. The filename f is then associated with the hash h_1 on one device, and h_2 on the other device. In the second scenario, two users may concurrently add the same file under two different versions; this results in the filename f being associated with two different unique file id u_1 and u_2. Since a filesystem cannot store multiple files under a single filename, Bitbox performs a deterministic renaming: the file `fn.ext` of hash h_1 and unique id u_1 is renamed to `fn-`h_1`-`u_1`.ext`. The renames performed during a merge are saved in the DAG so that Bitbox is able to detect *hidden* conflicts and perform renames appropriately. An hidden conflict happens when three conflicting version $\{(\texttt{f}_1\texttt{.ext}, h_1)\}$, $\{(\texttt{f}_1\texttt{.ext}, h_2)\}$ and $\{(\texttt{f}_1\texttt{.ext}, h_3)\}$ are merged. After the first merge, we obtain $\{(\texttt{f}_1\text{–}h_1\text{–}u_1\texttt{.ext}, h_1), (\texttt{f}_1\text{–}h_2\text{–}u_2\texttt{.ext}, h_2)\}$ and $\{(\texttt{f}_1\texttt{.ext}, h_3)\}$. The second merge could be performed without renaming but to ensure that the merge operation is associative, the hidden conflict must be detected and the rename performed.

As the Bitbox synchronization operation is not atomic, changes on the local file system may occur in the middle of a synchronization. To deal with such a case, Bitbox takes a snapshot of the filesystem (*i.e.,* the list of file names, and file hashes) just before running the synchronization process. All changes to the local filesystem occuring during a merge or an update are subsequently treated as if they occurred afterwards (*i.e.*, they are considered at the next update).

Algorithm 1. Updating and Synchronizing

$\mathcal{G}_L$ local history graph.
v_L current local version.
c_L local content description.
$\mathcal{G}_R, v_R, c_R$ the remote ones.
c'_L new local content descr.
H `SHA` applied on the **sorted** set
$\cdot$ concatenation operator
$\mathrm{P}_U(x)$ returns x or the last ancestors of x resulting from an UPDATE (cf. Fig. 2).

1: **def** MERGE$((v_L, c_L, \mathcal{G}_L), (v_R, c_R, \mathcal{G}_R))$
2: $\quad \mathcal{G}'_L \leftarrow \mathcal{G}_L + \mathcal{G}_R$
3: $\quad$ **if** $v_L \geq_{\mathcal{G}_L} v_R$ **then**
4: $\quad\quad$ **return**$(v_L, c_L, \mathcal{G}'_L)$
5: $\quad$ **else if** $v_R >_{\mathcal{G}_R} v_L$ **then**
6: $\quad\quad$ **return**$(v_R, c_R, \mathcal{G}'_L)$
7: $\quad$ **else**
8: $\quad\quad c_A \leftarrow$ COMMONANC$(\mathcal{G}'_L, v_L, v_R)$
9: $\quad\quad c'_L \leftarrow$ MERGECHANGES(c_A, c_L, c_R)
10: $\quad\quad v'_L \leftarrow \mathrm{H}(\mathrm{P}_U(v_L) \cup \mathrm{P}_U(v_R)) \cdot \mathrm{H}(c'_L)$
11: $\quad\quad \mathcal{G}'_L \leftarrow \mathcal{G}_L + \{v'_L\} + \{v_L \rightarrow v'_L, v_L \rightarrow v_R\}$
12: $\quad\quad$ **return**$(v'_L, c'_L, \mathcal{G}'_L)$
13: $\quad$ **end if**
14: **end def**

15: **def** UPDATE$((v_L, c_L, \mathcal{G}_L), c'_L)$
16: $\quad v'_L \leftarrow H(\mathrm{P}_U(v_L)) \cdot \mathrm{H}(c'_L)$
17: $\quad \mathcal{G}'_L \leftarrow \mathcal{G}_L + \{v'_L\} + \{v_L \rightarrow v'_L\}$
18: $\quad$ **return**$(v'_L, c'_L, \mathcal{G}'_L)$
19: **end def**

4 Proof of Correctness

In this Section, we show that our algorithm ensures that repositories are strongly eventually consistent, thus achieving our protocol goals.

Definition 1 (strong eventual consistency [3]). *The following properties must hold. (i) eventual delivery: an update delivered at some correct replica is eventually delivered to all correct replicas. (ii) convergence: correct replicas that have delivered the same updates eventually reach equivalent state. (iii) termination: all method executions terminate. (iv) strong convergence: correct replicas that have delivered the same updates have equivalent state.*

Theorem 1 (convergent replicated data type [3]). *Assuming eventual delivery and termination, any **state-based object (SBO)** that satisfies the **monotonic semilattice property** is strongly eventually consistent.*

Theorem 2. *A Bitbox object is a convergent replicated data type.*

Proof (sketch). The SBO in Bitbox is the tuple $\langle version, content, graph \rangle$. Replicas are equipped with *query* (trivial and omitted due to space constraints), *update* and *merge* operations (Alg.1) that work on the SBO state. SBO payload is its current tuple state, while its initial state is defined to be a share with no content (*e.g.*, an empty repository).

We need to show that this SBO is a monotonic semilattice (definition follows).

(i) payloads of the SBO form a join-semilattice ordered by $\leq$: the order $\leq$ operates on version identifiers. As they are not directly meaningful (as constituted by a hash value returned by operation at Alg.1 l.10 & l.16), an explicit

order is available by maintaining a DAG of version numbers (as on Fig.2), starting from the SBO's initial state. A DAG forms a partial order on its vertices. Next, we need to show that the SBO's structure exhibits:

- **idempotency**: $\textsc{Merge}(x, x)$ returns x, same graph and version (Alg.1 l.4).
- **commutativity**: $\textsc{Merge}(x, y)$ or conversely results in the same tuple (due to the sort operations at the hash functions Alg.1 l.11, and as additions of graph edges and states are commutative).
- **associativity**. First, operations on $\mathcal{G}$ are associative, involving only edge additions. Second, we define operations on content (*e.g.* MergeChanges Alg.1 l.9) to be solely additions and deletions of files; a file modification is then a deletion of the original file and the addition of the modified version. In this light, by assuming unique additions (*i.e.* unique file ids) and deletion operations occurring causally after corresponding additions, those operations on files actually characterize a *U-Set*, which is itself a convergent replicated data type [6]. We then directly obtain associativity. Finally, we look at the merge operation minus the $H(c'_L)$ operation (discussed at last step) for simplicity: $\textsc{Merge}(x, m = \textsc{Merge}(y, z)) = H(\mathrm{P}_U(x) \cup \mathrm{P}_U(m)) = H(\mathrm{P}_U(x) \cup (\mathrm{P}_U(y) \cup \mathrm{P}_U(z)) = H(\mathrm{P}_U(x) \cup \mathrm{P}_U(y) \cup \mathrm{P}_U(z)) = \textsc{Merge}(\textsc{Merge}(x, y), z)$

With these three identities, the SBO object forms a join-semilattice structure.

(ii) State of the SBO is monotonically non-decreasing across updates: each update creates a new version (Alg.1 l.16) that depends on previous version. A new version is appended to the DAG after the sink state, then becoming the new sink state. Consequently, since the partial order on state is defined by the DAG, the SBO's state is monotonically-non decreasing. □

References

1. Strauss, J., Lesniewski-Laas, C., Paluska, J.M., Ford, B., Morris, R., Kaashoek, F.: Device transparency: a new model for mobile storage. SIGOPS Oper. Syst. Rev. 44(1), 5–9 (2010)
2. Dropbox, `http://www.dropbox.com`
3. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free Replicated Data Types. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 386–400. Springer, Heidelberg (2011)
4. Riva, O., Yin, Q., Juric, D., Ucan, E., Roscoe, T.: Policy expressivity in the anzere personal cloud. In: SOCC (2011)
5. Git: distributed source control management, `http://git-scm.com`
6. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: A comprehensive study of Convergent and Commutative Replicated Data Types. INRIA, Tech. Rep. 7506 (2011)

Improving Miller's Algorithm Using the NAF and the Window NAF

Siham Ezzouak, Mohammed El Amrani, and Abdelmalek Azizi

The Department of Mathematics and Computer Science, Faculty of Science,
University Mohammed First, Oujda, BP 60000 Morocco
{szouak,elamranimohammed001,abdelmalekazizi}@yahoo.fr

Abstract. The Miller's algorithm is the most commonly used algorithm for computing pairing. To efficiently implement the pairings, it is necessary to optimise the computation time for the Miller's algorithm and the numbers of iterations. In this paper, we attempt to improve the original Miller's algorithm by using Non Adjacent Form (NAF) and The window NAF (NAF_w). These representations allow one to reduce the number of iterations in the original Miller's algorithm from $\frac{l}{2}$ to $\frac{l}{3}$ (NAF) or $\frac{l}{w+1}$ (NAF_w) where w is the size of the window in the NAF. Our approach is to replace the binary representation for the key by the NAF presentation or the NAF_w presentation in the Miller's algorithm.

Keywords: Elliptic curves, Pairing, Miller's algorithm, NAF, NAF_w.

1 Introduction

The use of pairings in cryptography was developed at an extraordinary pace. On the one hand, It's allow us to simplify existing protocols for example: the tripartite Diffie-Hellman protocol of Joux ([2]) and the decision problem of Diffie-Hellman [1]. On the other hand, we can construct the new protocols such as an Encryption based on the identity ([3]). Moreover In cryptanalysis, we can reduce the elliptic curve discrete logarithm problem to a discrete logarithm problem over the finite field where some attacks known to be Sub-exponential.

In the majority of applications, one of the following pairings (The Weil Pairing, the Tate pairing, the reduced Tate pairing, the Ate pairing and the Twist-Ate pairing) is used to construct cryptosystems. For computing these pairings, we make use of the famous Miller algorithm. In this paper, we attempt to improve this algorithm firstly by using the NAF and secondly with the NAF window. The remainder of this paper is organized as follows: In section 2, we describe the original Miller algorithm. In section 3, we recall the definition of the NAF and the NAF_w representations, we replace the binary representation in Miller's algorithm by one of the two and we compare the number of iterations and the running time of both the methods. Finally, in section 4, we concludes the paper.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 279–283, 2013.

2 Miller's Algorithm

After defining the pairing, we need to compute the function $f_{s,P}$. In our case we use the Miller's algorithm.

Theorem 1. *([7]) Let $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^k})$ then the Miller's function $f_{s,P}$ is a rational function on E with s zeroes at P, one pole at $[s]P$ and $s-1$ poles at $\mathcal{O}$.*
We denote $(f_{s,P})$ the divisor of $f_{s,P}$: $(f_{s,P}) = s\,[P] - [sP] - (s-1)\,[\mathcal{O}]$ $\forall s \in \mathbb{Z}$.
We construct $f_{s,P}$ using the following iterative formula :
$f_{i+j,P}(Q) = f_{i,P}(Q) * f_{j,P}(Q) * \frac{l_{[iP,jP]}(Q)}{v_{[i+j]P}(Q)}$ $\forall i, j \in \mathbb{Z}$.

$v_{[i+j]P}$ is the equation of the vertical line through point [i+j]P.
$l_{[iP,jP]}$ is the equation of the line through points [i]P and [j]P.

Implementation. Let $Dl_{T,P}$,$Nl_{T,P}$ Dv_{2T},Nv_{2T} be the denominator and the numerator of the line through T and P and the vertical line through point 2T respectively. The Miller 's algorithm is described by the pseudo-code bellow:

Algorithm 1. Miller's algorithm

Require: r=$\sum_{i=0}^{l-1} r_i * 2^i$ where $r_i \in \{0,1\}$ $P \in E(\mathbb{F}_q)$ et $Q \in E(\mathbb{F}_{q^k})$
Ensure: $f_{s,P}(Q)$
1: $T \leftarrow P$
2: $f_1 \leftarrow 1$
3: $f_2 \leftarrow 1$
4: $tmp \leftarrow \mathcal{O}$
5: **for** $i = l-1$ to 0 **do**
6: $tmp \leftarrow [2]T$
7: $f_1 \leftarrow f_1^2 * Nl_{T,T}(Q) * Dv_{tmp}(Q)$;
8: $f_2 \leftarrow f_2^2 * Dl_{T,T}(Q) * Nv_{tmp}(Q))$;
9: $T \leftarrow tmp$
10: **if** $r_i = 1$ **then**
11: $tmp \leftarrow T + P$
12: $f_1 \leftarrow f_1 * Nl_{T,P}(Q) * Dv_{tmp}(Q)$;
13: $f_2 \leftarrow f_2 * Dl_{T,P}(Q) * Nv_{tmp}(Q)$;
14: $T \leftarrow tmp$
15: **end if**
16: **end for**
17: Return $\frac{f_1}{f_2}$.

3 A Modified Miller's Algorithm

3.1 Miller's Algorithm with the NAF

Instead of representing the key k with the binary representation in the original Miller's algorithm, we use the NAF representation known as the a canonical representation with the fewest number of non-zero digits. In fact, the number of addition points in the Miller's algorithm is linked to number of non-zero digits i.e the hamming weight of the key. if one decrease this last, the number of operations is reduced and the running time will be improved. Furthermore, In the NAF representation, one must compute $-P$ which not require any operation $\forall P = (x, y) \in E(\mathbb{F}_q), -P = (x, -y)$. Consequently, the cost of the additive operations in the NAF is ignored.

Definition 1. *([9]) A non-adjacent form (NAF) of a positive integer k is an expression* $k=\sum_{i=0}^{l-1} k_i 2^i$ *where* $k_i \in \{0, \pm 1\}$, $k_{l-1} \neq 0$ *and no two consecutive digits* k_i *are non-zero i.e* $\forall i\ k_i k_{i+1} = 0$. *The length of the NAF is l.*

If we use this presentation for computing kP, the expected running time will be $\frac{l}{3}A + lD$ such that A and D the cost of the addition and the doubling point respectively instead of $\frac{l}{2}A + lD$ in the binary representation. So including this representation in the Miller's algorithm will decrease the numbers of addition by approximately $\frac{l}{6}$.

3.2 Miller's Algorithm with the NAF_w

The windows NAF is an improved version of the NAF which processes w digits of k at a time instead of one digit with the NAF which reduce the hamming weight. On the one hand the running time can be decreased, on the other hand more memory are used to store the $k_i P$. If extra memory is available this presentation is advised.

Definition 2. *[3]*
Let $w \geq 2$ *be a positive integer. A width-w NAF of a positive integer k is an expression* $NAF_w(k) = \sum_{i=0}^{l-1} k_i 2^i$ *where* $k_i \in \{-2^{w-1}, 2^{w-1} - 1\}$ *where each non-zero coefficient* k_i *is odd,* $\mid k_i \mid < 2^w - 1$, $k_{l-1} \neq 0$ *and at most one of any w consecutive digits is non-zero. The length of the width-w NAF is l.*

Since the NAF_w representation reduce the hamming weight from $\frac{1}{2}$ to $\frac{1}{w+1}$ than it's possible to optimize the Miller's algorithm with this latter. However some precomputations (computing $k_i P$) and extra memory (storing $k_i P$) are needed in the NAF_w.

3.3 Comparison between Algorithms

To build security elliptic curves we use prime numbers with digit size between 6 and 16 and the w-value equals to 6. We implemented our algorithm on Intel

Table 1. Running time comparison of three methods in seconds

Size of p	Miller Original	Miller with NAF	Miller with NAF_w
6	0.0062	0.00473	0.0054
7	0.0056	0.0048	0.0060
8	0.0052	0.0064	0.0060
9	0.0064	0.0048	0.0072
10	0.0088	0.0056	0.0064
11	0.0104	0.0076	0.0056
12	0.0104	0.0068	0.0088
13	0.0092	0.0052	0.0096
14	0.0080	0.0064	0.0088
15	0.0072	0.0060	0.0096
16	0.0088	0.0080	0.0097

pentium dual core processor 1.86 GHz and 782 MHz and 512 MB of memory using SAGE (Software Algebra Geometry Experimentation)[12]. The following table show the comparison of running time with both the three methods :

From the table 1, we find that the Miller algorithm with the NAF method take the least time to compute the pairing comparing to binary method and the NAF_w method. The NAF_w method known to take less iterations than NAF and binary method but the consuming time to compute $k_i * P$ at the first of algorithm increase when the digit size of prime p increase that explain why the running time is the slower in NAF_w method.

The following table 2 show the comparison of number iterations about the three methods. The number of iterations are divided to the numbers of additions and the number of doubling.

Table 2. Number Iterations comparison of three methods in seconds

Size of p	Miller Original		Miller with NAF		Miller with NAF_w	
	Add	Double	Add	Double	Add	Double
6	5	11	5	11	2	8
7	6	12	5	13	2	12
8	6	12	5	12	2	10
9	6	12	4	13	2	10
10	6	12	5	13	2	10
11	6	12	5	13	2	11
12	7	13	4	14	2	10
13	7	13	5	13	2	12
14	6	13	5	13	2	11
15	7	12	5	13	2	10
16	7	14	6	15	2	13

4 Conclusion

Since the Miller's algorithm is the heart of the pairings, several optimizations are applied in this algorithm. In this paper, we present one of them. We have decrease the number of iterations for both representations with extra memory requirement in the NAF_w and the running time is the least with the NAF method. Our approach can be adapted with the recent versions of the Miller for leading other optimizations such as the mixed coordinate and the denominator elimination. Our future work will analysed these two possibilities of optimizations.

References

1. Joux, A., Nguyen, K.: Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, `http://eprint.iacr.org/2001/003`
2. Joux, A.: A one round protocol for tripartite DiffieHellman. Journal of Cryptology 17(4), 263–276 (2004)
3. Hess, F.: Exponent Group Signature Schemes and Efficient Identity Based Signature Schemes Based on Pairings. Cryptology ePrint Archive, `http://eprint.iacr.org/2002/012`
4. Vercauteren, F.: Optimal Pairings. IEEE Transactions on Information Theory Cryptology 56(1), 455–461 (2010)
5. Galbraith, S.D., Harrison, K., Soldera, D.: Implementing the Tate Pairing. In: Fieker, C., Kohel, D.R. (eds.) ANTS 2002. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
6. Hess, F., Smart, N., Vercauteren, F.: The Eta Pairing Revisited. IEEE Transactions on Information Theory 52, 4595–4602 (2006)
7. Miller, V.-S.: The Weil pairing, and its efficient calculation. Journal Cryptology 17(4), 235–261 (2004)
8. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010)
9. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography, pages 92. Springer (2004)
10. Boxall, J., El Mrabet, N., Laguillaumie, F., Le, D.-P.: A Variant of Miller's Formula and Algorithm. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 417–434. Springer, Heidelberg (2010)
11. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, p. 354. Springer, Heidelberg (2002)
12. Stein, W.: SAGE mathematical software, Version 4.6 (2010), `http://www.sagemath.org`

Runtime Adaptation of Component Based Systems

Sihem Loukil, Slim Kallel, and Mohamed Jmaiel

ReDCAD Laboratory, University of Sfax, Tunisia
sihem.loukil@redcad.org,
slim.kallel@fsegs.rnu.tn, mohamed.jmaiel@enis.rnu.tn

Abstract. The need for continuously available software systems and their ability to support runtime adaptation is increasingly considered as one key issue in the software development. In particular, the software architecture of dynamically adaptive component based systems must continuously adapt to varying environmental conditions and user requirements. Therefore, they propose a wide range of possible adaptations that can not all be foreseen at design time. In this context, we propose to combine the Architecture Description Languages and the Aspect-Oriented Software Development which allow to make the adaptation process easier to design, understand and possible to validate.

1 Introduction

Software architecture modeling using Architecture Description Languages (ADLs) is becoming increasingly popular in the early phases of system development. Such languages facilitate the construction of high-level models in which systems are described as compositions of components. They play an important role in developing software systems deployed in large number of domains (companies, banks, air-ports, etc). Such systems must be always available and continuously adapt to varying environmental conditions and user requirements even at runtime. Hence, they should be modified/maintained during their execution, for example to include new functionalities, without being obliged to stop the system. This dynamic reconfiguration to maintain the system available presents a tedious task. In fact, not all possible reconfigurations that will be applied to the system can be foreseen at the time it is initially built and deployed. Therefore, the system must be flexible to support new needs that may appear during execution.

Very recently, several approaches like [1] are proposed to synchronize high level models with the running system. Such approaches focus on managing the variability of the dynamically adaptive systems by building a causal connection between abstract design models and the running system. In such approaches, an application is modeled using a base model and a set of variant models (aspects that encapsulate the variation points) in order to manage the variability of the adaptive application. Hence, an adaptation model is established to specify which variants should be selected according to the adaptation rules and the current

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 284–288, 2013.

context of the executing system. The main problem of such approaches is the use of more than one formalism to represent the design model (model representing at a high level of abstraction the architecture of the initial system before its deployment) and the execution model (model representing the abstraction of the architecture of a system at runtime). This forces the use of transformation languages and/or comparison models which increases the risk errors as it can give place to problems related to the synchronization of the different models, or to the propagation of changes among the different views. Moreover, to support unforeseen adaptations at design time in such approaches, the designer should manually create the modified model from scratch.

To tackle these issues, we propose an approach to manage the runtime adaptation of component based systems that uses one formalism to represent the design model as well as the execution model. This approach supports the unforeseen adaptations at design time by manually editing the parts in question using a graphical editor without being obliged to design the modified model from scratch.

In this context, we aim at combining the Architecture Description Languages (ADLs) and the Aspect-Oriented Software Development (AOSD) [2] paradigm which allow to make the adaptation process easier to design, understand and possible to validate.

For this purpose, we selected the Architecture Analysis & Design Language (AADL) [3], as an ADL, for specifying the architecture of dynamically adaptive component-based systems. This language uses the same formalism to represent the runtime model and the design model. Moreover, a previous publication [4] has provided a general overview of AO4AADL language, an Aspect-oriented extension for AADL. This aspect-oriented modeling language extended AADL with aspect-oriented concepts to design the crosscutting concerns related to the non-functional and technical properties. It is used in this work to allow designer monitoring the running system and performing the corresponding adaptation.

Our approach supports two types of runtime adaptations. First, the runtime adaptations resulting from a change in the execution context of the running system. Second, the ones resulting from the apparition of new user requirements that requires the manual intervention of the designer on the architectural specification of the system. In both cases, the adaptation actions are performed first on the model representing the architecture of the system. Applying the adaptation actions at the model level before applying them to the running system has the advantage that we can test their effect when applied as a whole without actually changing the system. Thereby, it is always possible to jump back to the state before starting to apply the adaptation actions in case an error is detected saving costly executions of roll-back operations on the system.

The remainder of this paper is organized as follows. Section 2 presents an overview of the proposed approach. In Section 3, we briefly present the monitoring module of the adaptation process. Section 4 details the related work. Finally, Section 5 concludes this paper and presents future work.

2 Architectural Reconfiguration of Component-Based Systems

Figure 1 shows the architecture of the proposed approach for managing component-based systems at runtime. This architecture comprises four levels: declarative level, instance level, runtime level and adaptation level.

At the declarative level, the designer defines the types of the components that can be used in the specification of the system. At the instance level, he specifies the base model that contains instances of the declared types as well as the connections between them. At this level, architectural aspects can be integrated to specify crosscutting concerns. The runtime level contains the runtime machinery that supports the execution of the system. The executed code is generated from the instance model and eventually from the defined architectural aspects. Finally, the adaptation level, which represents our main contribution, is responsible for the management of runtime adaptations that may result from a change in the execution context or the apparition of new user requirements at runtime. Then, these adaptations can affect either the running system (change in the execution context) or the instance model (apparition of new user requirements).

For the adaptations resulting from changes in the execution context, we use the aspect-oriented technique to define a set of architectural aspects that are intended to intercept the execution context variables and perform the corresponding reconfigurations. For the other type of adaptations, we employ the Hook methods technique which is intended to capture any manual intervention of the designer on the architectural model of the system.

In both cases, the effect of the adaptation actions to perform should be checked before committing them to the running system. For this purpose, they are applied first on the model representing the architecture of the system and the obtained configuration is validated through a set of architectural constraints defined at the declarative level. If the new configuration is valid, the adaptation actions are applied on the running system. Otherwise, it is simply discarded which allows to save costly executions of roll-back operations on the system.

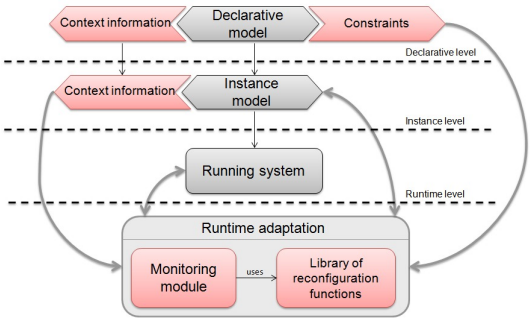


Fig. 1. The proposed architecture

As shown in figure 1, the runtime adaptation is composed of a main module called the monitoring module. This module checks if any change is occurred at the runtime level resulting from a change in the execution context or at the instance level. For this purpose, we employ an extended version of the declarative level that supports the specification of the context information. The context information and the adaptation rules are specified in a set of architectural aspects. To perform the adaptation actions, this module uses a library of reconfiguration functions that are included into the Ocarina tool [5].

At the declarative and instance level we use an extended version of AADL with aspect concepts described in AO4AADL language [4]. The choice of AADL was driven by many reasons. First, AADL is a standard and the resulting architecture enables simulation and analysis of architectural characteristics using precise execution and communication semantics. Second, AADL introduces two extension mechanisms (properties and annexes) which make the language much easier to extend. Moreover, it allows specifying architectural aspects using the AO4AADL extension. To achieve the step of designing the system, we developed our own graphical editor that integrates both AADL and AO4AADL concepts in order to make the job of the designer easier. The declarative model and the instance model are located at the server machine in the distributed application.

At the runtime level we use the RTSJ (Real Time Specification for Java) platform which allows executing RTSJ code generated from the AADL specification using Ocarina tool suite and easily weaving the AspectJ aspects generated from the AO4AADL ones using our AspectJ generator presented in [4].

3 Monitoring Module

The monitoring module is composed of two types of monitors: the running system monitor which looks at the changes that may be occurred at the runtime level and the instance model monitor which checks for changes at model level.

The running system monitor checks the changes in the context information through a set of AspectJ aspects woven into the code of the application. These aspects are generated from the AO4AADL aspects defined at the declarative level using our Aspect generator developed in the Ocarina tool suite [5]. The pointcut of such architectural aspect intercepts the execution of a port of a component or a parameter of a subprogram through which the information on the intercepted context variable is transferred. The advice code is fulfilled by the designer to specify the corresponding adaptation rules.

The instance model monitor is intended to check for changes on the instance model performed manually by the designer while the system is running. It is composed of a library of functions called Hook methods. These Hooks allow the designer to modify the functionalities of his software by realizing customized actions at well-defined times by inserting entry points to a list of actions. These methods include listeners to capture the evolution of the instance model.

4 Related Work

There are various points of view on how to reconfigure a system at runtime. Although this technique is recently introduced, several researchers have deepened their work in this area.

Similarly to our work, some approaches like [1] are based on aspect-oriented programming and model-oriented techniques to monitor and adapt application by building a causal connection between design models and the running system. Unlike our approach, the designer should manually create the modified model from scratch to support unforeseen adaptations. The authors present in [6] model-based traces as runtime models and traces analysis methods. However, the syntax and semantics of various types of the model-based traces in this work are not formally defined. Some other existing approaches show how runtime models can be derived efficiently from the specification, and how they support the designer in considering the execution of the application in the same formalism as the specification [7]. Unlike our approach, designers are required to consider the execution model when specifying any runtime adaptation, forcing them to understand the different formalisms of both the execution and the specification models.

5 Conclusion and Future Work

We have proposed an approach to manage the runtime adaption of component-based systems. Architectural aspects described in AO4AADL are used to catch the adaptations resulting from context information changes. These adaptations are foreseen at design time. For the unforeseen adaptations, we propose to manually act on the model to perform the adaptation.

In future work, we plan to extend our approach to support the detection of potential conflicts between runtime adaptation in a distributed system.

References

1. Morin, B., Barais, O., Jezequel, J.M., Fleurey, F., Solberg, A.: Models@ run.time to support dynamic adaptation. Computer 42, 44–51 (2009)
2. Filman, R.E., Elrad, T., Clarke, S., Akşit, M. (eds.): Aspect-Oriented Software Development. Addison-Wesley, Boston (2005)
3. SAE: Architecture Analysis & Design Language (2004), `http://www.sae.org`
4. Loukil, S., Kallel, S., Zalila, B., Jmaiel, M.: Toward an Aspect Oriented ADL for Embedded Systems. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 489–492. Springer, Heidelberg (2010)
5. Vergnaud, T., Zalila, B., Hugues, J.: Ocarina: a Compiler for the AADL. Technical report, Telecom Paristech - France (2006)
6. Maoz, S.: Using model-based traces as runtime models. Computer 42, 28–36 (2009)
7. Saudrais, S., Staikopoulos, A., Clarke, S.: Using specification models for runtime adaptations. In: International Workshop on Models@RunTime (2009)

Comparative Performance Analysis of AODV and AOMDV to Transmit H.264 Traffic

Adel Echchaachoui*, Ali Choukri, Ahmed Habbani, and Mohammed Elkoutbi

SIME laboratory, E.N.S.I.A.S, Med V – Souissi University, Rabat, Morocco
adel.echchaachoui@um5s.net.ma

Abstract. In this paper, we study and analyse the performances of two types of reactive routing protocols AODV that uses a single path and AOMDV using multipath to transmit a special video traffic based on the H.264.

This study will allow us to measure the impact of multipath routing on video packets transmission.

Keywords: AODV, AOMDV, H.264.

1 Introduction

A wireless ad-hoc is a dynamic network of mobiles nodes that communicate continuously with each other to hear and determine the evolution of the decentralized infrastructure of network [1].

In a Mobile Ad-hoc Network (MANET), the routing protocols allow to discover and route packets of data from the source to the destination. The choice of routing protocol depends mainly on its routing system and the type of transmission [2].

In this work, we investigate the transmission of H.264 video packet, and we determine which between the routing multipath and the routing single path offers the best performance for the delivery of these packets.

To make this comparison, we chose two reactive protocols: AODV and AOMDV.

2 Video and Routing Protocol

2.1 AODV

AODV (Ad-hoc On-Demand Distance Vector) is a reactive routing protocol used in mobile ad hoc networks: the paths are determined only if requested [3].

AODV establishes a single path from a source to a destination and deploys four messages in the routing system: Route Request (RREQ), Route Reply (RREP), Route Error (RERR) and Route Reply Acknowledgment (RREP-ACK).

It performs better than other protocols (like OLSR and DSR) when network has a small density [4, 5].

* Corresponding author.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 289–294, 2013.

2.2 AOMDV

AOMDV is a reactive routing protocol and an extension of AODV. It can discover multiple paths between the source node and the destination node. It is a multipath routing protocol.

Compared to AODV, AOMDV reduces packet loss and delay, and improves the process of route discovery [6].

In our study, we will measure the impact of this improvement on the performances of a traffic that requires a high quality infrastructure as H.264 traffic.

2.3 H.264

The H.264 standard was developed by ITU-T and ISO/IEC primarily to provide a very high compression of moving pictures for various applications such as video conferencing, communication and Internet streaming [7]. H.264 is the latest video codec standardized by ISO and ITU-T.

3 Related Work

Several comparisons of the performance of AODV and AOMDV has already been made and had the main aim of studying the behavior of these two reactive routing protocols in a normal Ad-hoc environment.

In [8] the authors studied the three protocols AODV, DSR and AOMDV to analyze the impact of multipath routing on a static network. They have deduced through this comparison that AOMDV react better than others in improving throughput and reducing the rate of packet loss.

W. Deepinder Singh and P. Tripatjot Singh showed in their comparison work that AOMDV provides more performance than AODV in terms of speed and Gigue, especially when the mobility of nodes is high [9].

To evaluate the performances of Ad-hoc routing protocols to route traffic MPEG-4, K. Kunavut and T. Sanguankotchakorn studied compared three protocols: OLSR, AODV and DSR. They showed that AODV is more appropriate than the others to deliver MPEG-4 traffic in a network with high mobility [10].

In another study and comparison, it has been shown that AODV is better for the transfer of MPEG-4 video packets than OLSR and DSDV. It generates a low traffic routing and a low packet jitter and provides good bandwidth [11].

4 Contribution

4.1 Simulation Environment

We used the network simulator NS-2 to achieve our simulations and evaluate the effect of multi-path routing on the performance of H.264 traffic [12]. To measure the maximum impact, we tried to create a highly mobile environment and changing each time the density of the network. We set the speed of the nodes to 30m/s without pause time and changed the number of nodes from 10 to 100.

4.2 Simulation Parameters

The parameters used in our simulation are presented in the following table (Tab.1):

Table 1. Simulation parameters

Simulator	NS 2
Physical layer radio type	802.11b
MAC protocol	802.11, CSMA/CA
Channel frequency (Ghz)	2.4
Antenna model	Omni-directional
Packet size (bytes)	512
Data rate (Mbps)	2
Simulation area (m^2)	1000 x 1000
Routing protocols	AODV, AOMDV
Traffic type	H.264
Mobility model	Random Waypoint
Speed (m/s)	30
Pause time (sec)	0
Number of nodes	Changing from 10 to 100

4.3 Metrics

To compare and study the behavior of AODV and AOMDV in the transmission of H.264 packets, we used the following metrics: End to end delay, PDR (Packet Delivery Ratio) and Throughput.

4.4 Results and Analysis

About End to End Delay

In Fig.4, average end to end delay of AODV and AOMDV delivering H.264 are showed relatively to the network density. Values are substantially identical throughout the simulation. They increased when the network density increased, except between 50 and 80 nodes where they changed very slightly. This can be explained by the fact that using one or more paths does not affect the average End to-End delay of packets.

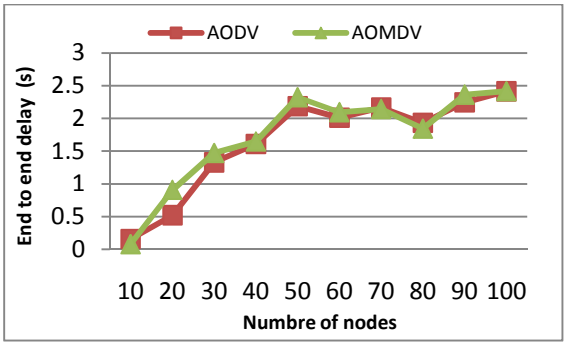


Fig. 1. Delay vs Network density

About PDR

Packet delivery ratio of AODV and AOMDV are illustrated in Fig.5. The number of packets delivered by AOMDV is greater than the number of packets delivered by AODV during all phases of the simulation. This is due because AODV is a single-path routing protocol, if a link is broken, the packet will not be delivered. However, because AOMDV is a multipath routing protocol, even if the link is broken, the network will find another route giving another chance for the delivery of the packet.

On the other hand, we note that the difference between the values of the two routing protocols decrease as the network density increases. Indeed, in a highly mobile (nodes speed = 30m/s and pause time = 0) and dense environment, finding an alternate path is more difficult. Hence, less number of alternate paths will be found by AOMDV when network become denser.

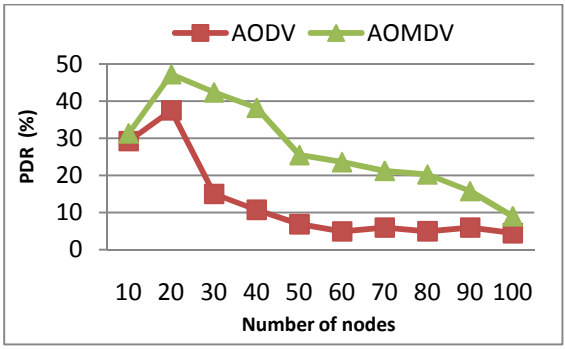


Fig. 2. PDR vs Network density

About Throughput

In Fig.6, throughput of AODV and AOMDV delivering H.264 are illustrated. During simulation, the throughput of AOMDV is better than AODV independently of network density when H.364 packets are generated to the network. When number of nodes increase, throughputs of AODV decrease linearly. By cons, values of AOMDV are irregulars but much higher than AODV. AODV is a single path routing protocol, once a link breaks the packet delivery along that route stops. Due to AOMDV being a

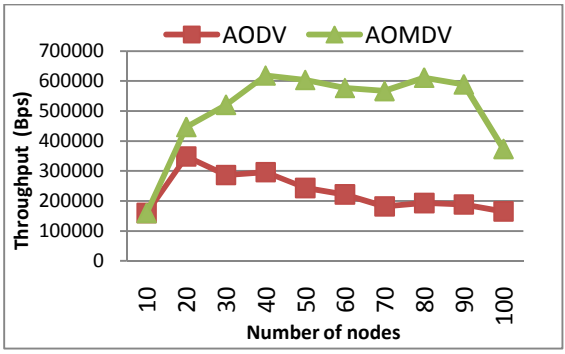


Fig. 3. Throughput vs Network density

multipath routing protocol, it searches for alternate paths if the current route breaks. This mechanism allows AOMDV to offer more throughput than AODV to transmit H.264 packets independently of network density.

5 Conclusions and Perspectives

This paper assesses and analyzes the performance of AODV and AOMDV to route video packets H.264, using NS-2 as simulation tool. The comparison study was based on three metrics: average End to end delay, Packet delivery Ratio and throughput. We found that AOMDV provides better results than AODV especially when the network density is not high. AOMDV outperforms AODV due to its mechanism that tries to find an available route when the primary link is broken.

We conclude that AOMDV is better than AODV to ensure a quality traffic of H.264 video packets, except when the density and mobility of the network becomes very important, the difference between the performances of two routing protocols are slight. Our next work will involve the optimization of AOMDV routing protocol to offer a good alternation of routes even when the network is very dense.

References

1. Royer, E.M., Chai-Keong, T.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications 6, 46–55 (1999)
2. Chenna Reddy, P., ChandraSekhar Reddy, P.: Performance Analysis of Adhoc Network Routing Protocols. In: International Symposium on Ad Hoc and Ubiquitous Computing, ISAUHC 2006, pp. 186–187 (2006)
3. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC (2003)
4. Kanthe, A.M., Simunic, D., Prasad, R.: Comparison of AODV and DSR on-demand routing protocols in mobile ad hoc networks. In: 2012 1st International Conference on Emerging Technology Trends in Electronics, Communication and Networking (ET2ECN), pp. 1–5 (2012)
5. Rahman, M.A., Anwar, F., Naeem, J., Abedin, M.S.M.: A simulation based performance comparison of routing protocol on Mobile Ad-hoc Network (proactive, reactive and hybrid). In: 2010 International Conference on Computer and Communication Engineering (ICCCE), pp. 1–5 (2010)
6. Marina, M.K., Das, S.R.: Ad hoc on-demand multipath distance vector routing: Research Articles. Wirel. Commun. Mob. Comput. 6, 969–988 (2006)
7. Tamhankar, A., Rao, K.R.: An overview of H.264/MPEG-4 Part 10. In: 4th EURASIP Conference Focused on Video/Image Processing and Multimedia Communications, vol. 1, pp. 1–51 (2003)
8. Manveen Singh, C., Rambir, J., Sandeep: Simulation and Comparison of AODV, DSR and AOMDV Routing Protocols in MANETs. International Journal of Soft Computing & Engineering 2, 375–381 (2012)
9. Deepinder Singh, W., Tripatjot Singh, P.: Performance Comparison of Single and Multipath Routing Protocols in Adhoc Networks. International Journal of Computer Technology and Applications 02, 1486–1496 (2011)

10. Kunavut, K., Sanguankotchakorn, T.: Performance evaluation of ad hoc routing protocols to deliver MPEG-4 traffic. In: 2010 12th IEEE International Conference on Communication Technology (ICCT), pp. 207–210 (2010)
11. Chowdhury, M.U., Perera, D., Pham, T.: A performance comparison of three wireless multi hop ad-hoc network routing protocols when streaming MPEG4 traffic. In: Proceedings of the 8th International Multitopic Conference, INMIC 2004, pp. 516–521 (2004)
12. Peter, W.: The VINT Project, The Network Simulator - ns-2, `http://www.isi.edu/nsnam/ns/` (accessed 2008)

Large Scale 3D Shape Retrieval Based on Multi-core Architectures

El Wardani Dadi and El Mostafa Daoudi

University of Mohammed First, Faculty of Sciences, LaRi Laboratory,
Oujda, MOROCCO
wrd.dadi@gmail.com, m.daoudi@fso.ump.ma

Abstract. Despite of the variety of approaches proposed in the literature in order to improve the execution time of the 3D shape retrieval [14,15], the challenge that still remains is to design a 3D shape retrieval method that allows the large scale retrieval and, in the same time, respects the relevance of the obtained results. In this work, we deal with the problem of the large scale of 3D shape retrieval by proposing new implementations on multi-core environment. At our knowledge, a few partial works based on HPC (High Performance Computing), have been proposed in the literature [1,2]. The proposed solutions are designed for the GPU (Graphical Processing Unit) and concern only the step of the extraction of the SIFT salient local features. In order to optimally exploit the potential of the multi-core architectures, we have studied different data distributions. Experimental results, under OpenMP environment, show that the large scale retrieval can be achieved using the multi-core environment.

Keywords: 3D Content-based Shape Retrieval, Large Scale Retrieval, multi-core architecture, OpenMP, accelerate 3D shape retrieval.

1 Introduction

Currently, there are an increasing number of 3D objects on the web, leading to large databases, thanks to recent digitizing and modeling technologies. The need of efficient methods for 3D shape-content based retrieval, in order to ease navigation into related large databases, and also to structure, organize and manage this new multimedia type of data, has become an active topic in various research communities such as computer vision, computer graphics, mechanical CAD, and pattern recognition. The 3D shape retrieval is the processing of retrieving visual similar objects to given 3D object query.

Various 3D shape retrieval methods have been proposed in the literature [3,4,5,6,7]. A good 3D shape retrieval method must satisfy at least two conditions simultaneously [3]:

- the relevance (the first 3D objects returned by the method must be the most similar to the query)
- The retrieval results should be fast

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 295–299, 2013.

In most of existing methods, it is unlikely to satisfy the two conditions simultaneously. Moreover, for the large database, the retrieval process becomes increasingly difficult and needs more computational times; which does not allow the large scale retrieval.

In this paper we propose new implementations on multi-core environment in order to improve the execution time of the retrieval process in the large databases of three-dimensional models (large scale). At our knowledge, a few partial works based on HPC (High Performance Computing), have been proposed in the literature [1, 2]. The proposed solutions are designed for the GPU (Graphical Processing Unit) and concern only the step of the extraction of the SIFT salient local features. To exploit the potential of the multi-core architecture at their maximum performance and improve the load balancing between different cores, we have used dynamic scheduling for data distribution.

For the experimental tests, we have chosen to parallelize the CMBOF method proposed by Zhouhui et al. [8], since it gives the best result comparing to many other methods in particular the view based methods [9,10,11,12]. Two objectives are explored: the first one consists in designing highly optimized algorithms that fully exploit all the available resources; the second one is the scalability of the algorithms on a large number of cores.

The rest of the paper is organized as follows. The parallelization on multi-core is presented in section 2. Section 3 is devoted to experimental results. We provide some perspectives and we conclude the paper in section 4.

2 Parallelization on Multi-core

Current machines offer microprocessors composed of multiple cores (processors) and Graphical processing Units (GPU). The major challenge is how to exploit the potential of these architectures at their maximum performance.

The aim of our work is to propose parallel solutions to accelerate the retrieval process and therefore allow the large scale retrieval (the retrieval in large databases) by exploiting the potential of the multi-core architectures.

We recall that the retrieval process, using the CMBOF method, is performed into essentials phases. These two phases are done online:

- Indexing of the 3D query-object: computing the descriptor of the shape of query
- Shape matching: comparing the descriptor of the query-object with the descriptor of each 3D objects of the database.

Our goal in this paper consists to parallelize each phase.

2.1 Parallel Shape Indexing

To compute the descriptor of a given 3D object, the CMBOF method [8] proposes to characterize the object by several word histograms (descriptors) where each descriptor corresponds to a 2D view (2D image) captured around the shape of this object.

Since the computation of the descriptors of the views is independent, we can compute simultaneously several 2D view descriptors on different processors (cores). For the data distribution, we propose two strategies: a static and then a dynamic schedule. We assume that we have p cores denoted by «Pi» for 0≤i<p and n views denoted by «Vi» for 0≤i<n.

In order to optimally balance the load between different cores, a dynamic schedule approach is adopted; as follows:

- Assign to each core « Pi » an initial workload (a number k of views with k≤). The remaining block of views will be shared between all cores.
- Each core computes in parallel the descriptors of the 2D views that are assigned to it.
- As soon as a core completes its work, it takes one 2D view from the shared block (the remaining views). This process is repeated as long as it remains untreated views.

2.2 Parallel Shape Matching

After calculating the descriptor of the 3D query object, the second phase of the retrieval process is the shape matching. In this phase, the descriptor of the query is compared with descriptors of each 3D object belonging to the database. Note that the descriptors of objects in the database are computed offline.

Sequentially shape matching in a large database is time consuming. The advantage of using multi-core is to compare simultaneously the query-objects with p objects; where p is the number of cores. For the data distribution, we use the same strategies studied in the previous section.

We assume that the database is composed of m 3D objects denoted by «Oi» for 0≤i<m. For the dynamic distribution we proceed as follows:

- Assign to each core « Pi » an initial workload (a number k of objects with k≤). The remaining block of objects will be shared between all cores.
- Each core executes in parallel the comparison of the query object with the objects it assigned.
- As soon as a processor (core) completes its work, it takes one objects from the shared block (the remaining objects). This process is repeated as long as it remains untreated objects.

3 Experimental Results

Experimental results are performed on a Dell PowerEdge 2900 Server Quadra-Core Intel® CPU E5310 160 GHz, 2GB RAM. All programs are implemented in C/OpenMP in Visual Studio 2010 environment (VC2010).

- To extract SIFT salient feature, we are used the C version of SIFT [16].
- To capture 66 views around a 3D object, we are used the executable provided by Zhouhui [8]

– We are used Princeton 3D Shape Benchmark database [17] composed of 1810 - 3D Objects.

Figure 1 and figure 2 show if we increase the number of cores, the parallel time remains close to the sequential time divided by the number of cores.

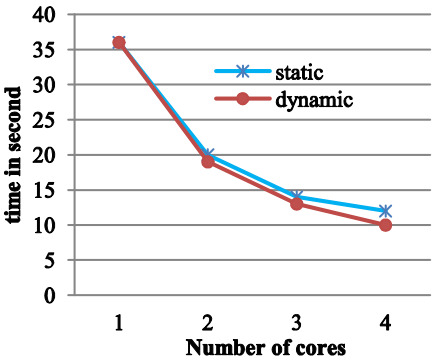


Fig. 1. Execution time for computing the descriptor of a 3D object on a multi-core

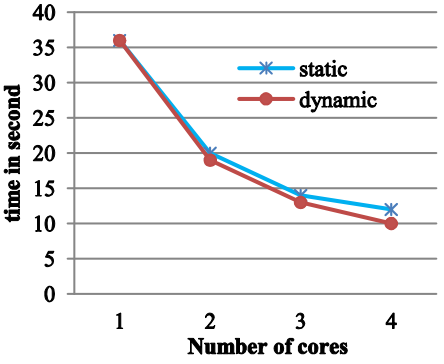


Fig. 2. Execution time for the shape matching process

In Figure 3, we measure the speed up (sequential_time / parallel_time) of the shape matching process based on dynamic scheduling and executed on 4 cores. The obtained speed up is close to the ideal one (94%). This shows that the proposed algorithms are scalable,

In Figure 4, we compare the execution time of the retrieval process for different sizes of databases. In this test we report the evolution of the parallel time on 4 cores versus sequential time. The results show that the size of the database does not affect the speed up. This shows that if we use large databases using p cores, the parallel time remains close to the sequential time divided by p. We conclude that the large scale can be achieved by using a great number of cores.

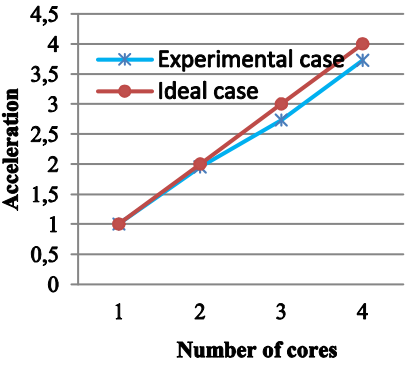


Fig. 3. Comparison of the experimental with the ideal acceleration

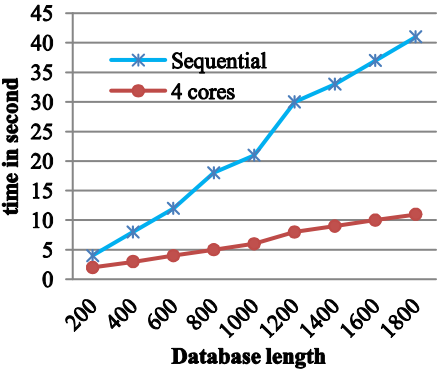


Fig. 4. Execution time of the retrieval process on databases with different sizes

4 Conclusion

In this paper we have proposed a new implementation on multi-core architecture, to accelerate the 3D shape retrieval in large databases. The experimental results show that the large scale can be achieved for a large number of cores. In future work, we propose to combine multi-core architectures with GPU accelerators in order to improve our results.

References

1. Ohbuchi, R., Furuya, T.: Accelerating Bag-of-Features SIFT Algorithm for 3D Model Retrieval. In: SAMT 2009 (2009)
2. Kuang, Q., Zhao, L.: A Practical GPU Based KNN Algorithm. In: Proceedings of the Second Symposium International Computer Science and Computational Technology(ISCSCT 2009), Huangshan, P. R. China, December 26-28, pp. 151–155 (2009)
3. Tangelder, J.W.H., Veltkamp, R.C.: A survey of content based 3D shape retrieval methods. Multimedia Tools and Applications 39(3), 441–471 (2008)
4. Shilane, P., Kazhdan, M., Min, P., Funkhouser, T.: The princeton shape benchmark. In: Proc. Shape Modeling International 2004, pp. 157–166 (2004)
5. Zaharia, T., Preteux, F.: 3D versus 2D/3D shape descriptors: A comparative study. In: SPIE Conf. on Image Processing: Algorithms and Systems III - IS & T/ SPIE Symposium on Electronic Imaging, Science and Technology 2003, San Jose, CA, vol. 5298 (January 2004)
6. Bustos, B., Keim, D.A., Schreck, T., Vranic, D.: An experimental comparison of feature-based 3D retrieval methods. In: 2nd Int. Symp. on 3D Data Processing, Visualization, and Transmission (3DPVT 2004), Thessaloniki, Greece (September 2004)
7. Del Bimbo, A., Pala, P.: Content-based retrieval of 3D models. ACM Trans. Multimedia Com
8. Lian, Z., Godil, A., Sun, X.: Visual Similarity based 3D Shape Retrieval. In: IEEE International Conference on Shape Modeling and Applications, SMI (2010)
9. Chen, D.-Y., Tian, X.-P., Shen, Y.-T., Ouhyoung, M.: On visual similarity based 3d model retrieval. In: Eurographics, Granada, Spain (September 2003)
10. Ohbuchi, R., Osada, K., Furuya, T., Banno, T.: Salient Local Visual Features for Shape Based 3D Model Retrieval. In: Proc. IEEE International Conference on Shape Modeling and Applications (SMI 2008), Stony Brook University, June 4 - 6 (2008)
11. Daras, P., Axenopoulos, A.: A 3D shape retrieval framework supporting multimodal queries. Int'l. Journal of Computer Vision (IJCV)
12. Chaouch, M., Verroust-Blondet, A.: A new descriptor for 2D depth image indexing and 3D model retrieval. In: Proc. ICIP 2007, vol. 6, pp. 373–376 (2007)
13. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. Int'l. Journal of Computer Vision 60(2) (November 2004)
14. Dadi, E.W., Daoudi, E.M., Tadonki, C.: Fast 3D shape retrieval method for classified databases. In: IEEE International Conference on Complex Systems, ICCS (2012)
15. Veltkamp, R.C., Giezeman, G.-J., Bast, H., Baumbach, T., Furuya, T., Giesen, J., Godil, A., Lian, Z., Ohbuchi, R., Saleem, W.: SHREC 2010 Track: Large Scale Retrieval. In: Eurographics Workshop on 3D Object Retrieval (2010)

New Forwarding Strategy for PROPHET Routing in Delay Tolerant Networks

Ahmed El Ouadrhiri[1], Mohamed El Kamili[1],
Mohammed Raiss El Fenni[2], and Lahcen Omari[1]

[1] LIM, Faculty of Science Dhar El Mahrez, USMBA,
30000 Fez, Morocco
[2] LIA, University of Avignon, France

Abstract. Delay tolerant networks (DTNs) are based on the concept of store-carry and forward protocols: a node may store a message in its buffer and carry it for a long period of time, until an appropriate forwarding opportunity arises. DTNs are able to provide communication services in areas where there is no guarantee that a fully connected path between source and destination exists at any time, and seek to address the technical routing issues in wireless networks. Traditional routing protocols are unable to deliver messages between hosts in such conditions. In this paper, we address the problem of routing in DTNs and propose a new forwarding strategy based on the predictability control of a probabilistic routing model. This strategy increase the number of received messages without increasing the rate of relayed messages, which is more suitable regarding the battery power limitation of mobile nodes. Simulation results show that our new forwarding strategy performs better than the PROPHET routing solution well known by the research community.

Keywords: DTNs, PROPHET, Predictability, ONE Simulator.

1 Introduction

In order to provide communication services in the absence of end-to-end paths, researchers have proposed a new networking paradigm, often referred to as Delay Tolerant Networking (DTN [4]), based on the *store-carry-and-forward routing principle* [2]. Several solutions have been proposed to handle routing in such networks. To increase the probability of delivery, messages replicate many times in the network and nodes must send a message to the appropriate node. For example, in PROPHET (Probabilistic ROuting Protocol using History of Encounters and Transitivity)[5], messages are sent to the nodes that have a big chance to encounter the destination.

Many improvements have been made for PROPHET. In [3], Boudguig et *al.* propose a new predictability equations by introducing a new optimization factor $\alpha \in [0, 1]$ in the original equations used for calculating the predictability in the PROPHET. In [1], Burns et *al.* propose a new predictability concept based on the recorded movement of the node during the last t rounds and suppose

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 300–305, 2013.

that movements of humans and vehicles are periodic. In [7], Quan Yuan et *al.* propose predict and relay (PER), an efficient routing algorithm for DTNs, where nodes determine the probability distribution of future contact times and choose a proper next-hop in order to improve the end-to-end delivery probability.

The paper is organized as follows: In section II, we introduce the routing protocols concept in DTNs. We describe our new proposed approach in section III. Simulation results and their interpretations are presented in section IV. Finally, we conclude the paper and give some perspectives.

2 Routing in DTNs

Routing is one of the very important challenges in DTNs, and the efficiency remains the first objective to reach. In literature, the researchers on this field have dealt with the problem by different methods [5–7]. Epidemic routing is based on flooding. It relies on neighbours to transmit messages through flooding. It is considered as the dominant choice due to its simplicity, and its optimal performance in terms of delay and delivery rate when resources are unbounded, even if it consumes the maximum of resources. Probabilistic routing transmits messages only to the most relevant nodes in the next hop, based on parameters such as the history of encounters, queues, remaining energy, etc...

PROPHET [5], is among the most widely used probabilistic protocols; it uses the history of previous encounters between nodes to calculate predictability. It uses three original equations to calculate the predictability:

1. ***Direct Contact:***

$$P(a,b) = P(a,b)_{old} + (1 - P(a,b)_{old}) \times P_{init}$$

   $P(a,b)$ is the predictability of the node b stored in the node a.
2. ***Transitivity:***

$$P(a,c) = P(a,c)_{old} + (1 - P(a,c)_{old}) \times P(a,b) \times P(b,c) \times \beta$$

   $\beta \in]0,1[$ is a constant that expresses the influence of transitivity.
3. ***Aging:***

$$P(a,b) = P(a,b)_{old} \times \gamma^k$$

   with $\gamma \in]0,1[$ is the aging constant, it is the number of time units that have elapsed since their last meeting.

Forwarding strategy: when two nodes meet, a message is sent from node 1 to node 2 if the delivery predictability of the destination of the message is higher for node 2. Node 1 does not delete the message after sending it as long as there is sufficient buffer space available. Systems management queue used when buffers are full. In our evaluations, we have used FIFO queues.

3 Proposed Approach

In forwarding strategy of PROPHET protocol [5], a message is sent to the encountered node only if it has a higher delivery predictability for the destination. The inconvenience of this approach is that if the creator node of the message has a higher predictability than the other encountered nodes, the message remains in the queue until being sent to its destination or deleted by the queue. Therefore, this message does not have a good opportunity to reach its destination. On the other hand, when a node encounters another one with a low delivery predictability, there is a case where the last node is more likely to meet the destination than the other nodes with high predictability. Moreover, if the message destination has a high predictability in all network nodes, this message remains exchanged many times between nodes (even if this message has already arrived at its destination) to the detriment of the other messages that their destinations have low predictability.

The new Forwarding Strategy: Our new approach avoids these problems and ensures fairness (between messages that have high predictability and others that have low predictability) in transmission of messages. The new approach is based on the sum of predictabilities of nodes that carry the message. In fact, we have proposed that each node manage its own messages. A message is sent to the encountered node if the predictability of this node for the destination strictly positive and the sum of predictabilities of all nodes carrying the message is lower than a Threshold *Tsh*. With this new strategy, we can send many copies of messages that have low predictability and few copies of messages that have high predictability. The other messages (messages created by other nodes) that carried by this node are sent only to their destinations if encountered.

The algorithm 1 summarizes iterations of the new forwarding strategy, when two nodes A and B meet (for example, node A execute the algorithm), we have:

Algorithm 1. New Forwarding Strategy

1: Send all messages destined to node B.
2: For all messages generated by node A: (node that execute the algorithm)
3: **if** $P(B, M.dest) > 0$ and $M.SumPredctInNetwork \leq Tsh$ **then**
4: $M.SumPredctInNetwork \leftarrow M.SumPredctInNetwork + P(B, M.dest)$
5: Select the message M for being sent to B.
6: **end if**

→ M: is a message of the node A.
→ M.dest : is the destination of the message M.
→ $P(B, M.dest)$: is the predictability of the node B for the destination of the message M.
→ M.SumPredctInNetwork: is the sum of the predictabilities of all nodes in the network that carrying the message M.

4 Simulation Results

We have used the ONE simulator (Opportunistic Network Environment) [9] to validate the performance of our new approach. We have used a similar configuration to the one used in [5]. This scenario consists of a $1500m * 300m$ area where 50 nodes are randomly placed. These nodes move according to the random waypoint mobility model [10] with speeds of $0 - 20m/s$. From a subset of 45 nodes, one message is sent every second for 1980 seconds of the simulation (each of the 45 nodes sending one message to the other 44 nodes), and the simulation is then run for another 2020 seconds to allow messages to be received.

We ran simulations varying the queue size. Table 1 shows the relayed and received messages for PROPHET and our new forwarding strategy with Tsh=2 and Tsh=4. While sending the same number of messages, it is easy to see that our new approach deliver more messages (34,8% for Tsh=4 and 36,'% for Tsh=2) by exchanging a small number of them. Compared with PROPHET, we improve the efficiency by delivering more messages with low energy consumption. Forwarding strategy of PROPHET protocol is based only on high predictability of messages, which is insufficient since we have to control the exchanged messages also.

Table 1. Received and relayed messages in buffer size = 10 messages

protocol	created	relayed	received	dropped	delivery rate	buffertime avg
PROPHET	2020	74286	453	75351	22,4%	11,975
Tsh = 2	2020	3752	736	4540	36,4%	192,771
Tsh = 4	2020	5992	704	6812	34,8%	132,149

Lets increase the queue size to 60 and see how it affects the network performances. Based on Table 2, PROPHET protocol exchange more messages than the new forwarding strategy for both thresholds, but still delivering less messages. The delivery rate has improved for all protocols, but our strategy performs very well compared to PROPHET. This behavior is resulting from the fact that messages reside for a long time in queues using our strategy. In fact, the main idea in our new forwarding strategy is to keep the message in the queue until meeting the destination, instead of forwarding the message to all nodes as in the PROPHET protocol.

Table 2. Received and relayed messages in buffer size = 60 messages

protocol	created	relayed	received	dropped	delivery rate	buffertime avg
PROPHET	2020	127559	793	125958	39,2%	38,13
Tsh = 2	2020	4584	1495	2164	74%	1140,54
Tsh = 4	2020	7018	1566	4487	77,5%	783,89

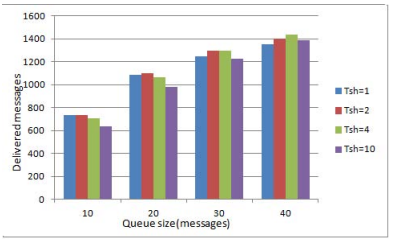


Fig. 1. Received messages

We have shown through simulations that our new forwarding strategy performs better than the one used in PROPHET protocol. Now, we will investigate the impact of the threshold *Tsh*. In Figure 1, we plot the delivery rate for different Thresholds varying the queue size. For example, Tsh=2 performs better than the other values if buffer size is between 10 and 20, whereas for buffer size 30 and 40, Tsh=4 is the optimal. It is interesting to see that the optimal threshold depends on the buffer size. Furthermore, we can not intuitively determine the optimal value of the threshold.

5 Conclusion

In this paper, we address the problem of routing in Delay Tolerant Networks. We have proposed a new forwarding strategy based on the predictability control of the PROPHET routing protocol, and a threshold on the number of retransmissions. We have shown through simulations that our new forwarding strategy performs better than PROPHET. This new strategy increase the number of received messages and decreases the number of relayed ones, which increases the overall throughput of the system and conserves the battery life by minimizing the energy consumption of mobile nodes. Learning the optimal threshold value will be an extension of this model in a future work. Using estimator algorithms we can determine the threshold that achieves the optimal performance with a low cost [11]. Since routing involves strategic decisions between DTN nodes, our future direction is to introduce game theory [8].

References

1. Burns, B., Brock, O., Levine, B.N.: MV routing and capacity building in disruption tolerant networks. In: Proc. IEEE INFOCOM (2005)
2. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: Proc. of ACM SIGCOMM (August 2004)
3. Boudguig, M., Abdali, A.: New predictability concept for routing in DTN. In: IEEE International Conference on Multimedia Computing and Systems (ICMCS), Tangiers Morocco, May 10-12 (2012), `http://ieeexplore.ieee.org`
4. Delay tolerant networking research group, `http://www.dtnrg.org`

5. Lindgren, A., Doria, A., Schelén, O.: Probabilistic routing in intermittently connected networks. Springer, Heidelberg (2004)
6. Vahdat, A., Becker, D.: Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University (2000)
7. Yuan, Q., Cardei, I., Wu, J.: An efficient prediction-based routing in disruption-tolerant networks. IEEE Transactions on Parallel and Distributed Systems 23(1), 19–31 (2012)
8. Laveen Sundararaj, L., Vellaiyan, P.: Delay tolerant networking routing as a game theory problem - An Overview. International Journal of Computer Networks 2(3), 159–172 (2010)
9. ONE Simulator, `http://www.netlab.tkk.fi/tutkimus/dtn/theone/`
10. Pelov, A.: Mobility models for wireless networks. Thesis, University of Strasbourg, Department of Mathematics and Computer Science (2009)
11. Thathachar, M.A.L., Sastry, P.S.: Estimator algorithms for learning automata. In: Proc. Platinum Jubilee Conf. on Systems and Signal Processing, Bangalore, India (1986)

New Validation Approach Based on Group MADM for Network Selection

Mohamed Lahby, Leghris Cherkaoui, and Abdellah Adib

Department of Computer Science, LIM Lab.
Faculty of Sciences and Technology of Mohammedia
{mlahby,cleghris,adib_adbe}@yahoo.fr

Abstract. In this paper, we propose a new validation approach for network selection access. The proposed approach is able to deal with the inconsistent ranking outcomes problem of the multi attribute decision making (MADM) methods by using the group MADM. Thus, we aim to select the appropriate weighting algorithm which can be combined with the ranking algorithm based on Mahalanobis distance. Simulation results are presented to illustrate the effectiveness of our new validation approach for background traffic and streaming traffic.

Keywords: Heterogeneous Multi-Access, Network Selection, Multi Attribute Decision Making, Group Decision Making.

1 Introduction

Nowadays, all users need to connect on the Internet anywhere and anytime, with the best quality of service (QoS) and the minimum of services' cost. The fourth generation (4G) of wireless communications are integrating a multitude of radio access technologies (Rats) such as WIFI (IEEE 802.11a, IEEE 802.11b, etc), WIMAX (IEEE 802.16), and LTE. Moreover, the mobile devices are equipped with multiple interfaces which offer multiple possibilities for the user to connect on the Internet.

The most important issue in radio access technologies is to maintain a seamless service continuity under the principle "Always Best Connected" (ABC) [1]. The network selection is intended to determine the most suitable network in terms of quality of service (QoS) for mobile users. However, this process is considered as a complex problem and mapped in NP-Hard problem [2]. To address this issue, multi attribute decision making (MADM) methods have been widely used in the context of network selection.

Indeed, due to nature of network selection problem, MADM methods represent a promising solution which can be applied to network selection problem. Several network selection algorithms based on MADM methods are utilized to choose the best access network among other available networks. In [3] the network selection decision is modeled using two MADM methods AHP and SAW. The AHP method is used to provide a weight for each criterion involved in the network selection. While the SAW algorithm is applied to provide a ranking

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 306–310, 2013.

of all alternatives. In [4] the network selection algorithm is based on AHP and TOPSIS. The AHP method is used to weigh each criterion and TOPSIS method is applied to determine the ranking of access network.

The authors [5] have proposed a new ranking algorithm based on Mahalanobis distance. This algorithm allows to select the best available access network and to provide the better performance than the classical MADM methods. However, there is no single MADM weighting algorithm which is considered favorable than other methods to be combined with Mahalanobis distance algorithm. In addition, there is no study examining the validity of MADM methods that can be used to choose the best access network.

The validity of MADM methods includes two main issues which are the selection of the most valid ranking algorithm and the determination of the most suitable weights for different criteria. To cope with these issues, the authors [6] have proposed a new validation approach based on group MADM methods. The proposed validation approach can select the group ranking outcome of an MADM method which has the highest consistency degree with it's corresponding individual ranking outcomes. The weakness of the proposed validation lies in fact that is do not takes into account the weighting algorithms, the approach assumes that all attributes are weighted equally by different decision makers.

In this work, we propose a validation scheme to select the suitable network service for mobile device. The proposed approach takes into account four weighting methods AHP[7], FAHP[8], ANP[9], and FANP[10] by using the group MADM.

The goal of this paper, is to determine a suitable weighting algorithm which can be combined with the Mahalanobis distance. The remainder of this paper is organized as follows. Section 2 presents our new validation approach based on group MADM. Section 3 includes the simulations and results. Section 4 concludes this paper.

2 New Validation Approach Based on Group MADM

The validation of the appropriate MADM algorithm which can be applied in the network selection decision remains an open issue. To deal with this issue, we propose a new validation approach which can take into account different weighting methods. Our approach is based on group MADM and it allows to select the group ranking outcome of the MADM method which has the highest consistency degree with it's corresponding individual ranking outcome.

According to [6], the group MADM problem involves a finite number of alternatives $A = \{A_i,\ for\ i = 1, 2...n\}$, which are to be evaluated by a group of p decision makers $DM = \{DM_k,\ for\ k = 1, 2...p\}$ with respect to a set of m attributes $C = \{C_j,\ for\ j = 1, 2...m\}$.

In order to deal with weakness of the proposed approach in [6], we assume that each decision maker DM_k decides the relative importance for each attribute C_j by using for weighting algorithms: AHP, FAHP, ANP and FANP. Our new validation approach can be categorized in eight steps.

1. Construct of the individual decision matrix: each decision maker DM_k evaluates the performance rating x_{ij}^k of alternative A_i with respect to attribute C_j in order to construct the decision matrix X^k. The individual decision matrix is expressed as

$$X^k = (x_{ij}^k) \quad i = 1, ..n, \quad j = 1, .., m \tag{1}$$

2. Construct of the individual weight vector: for that we construct four weight vectors T_1^k, T_2^k, T_3^k, T_4^k by using four weighting algorithms namely AHP, FAHP, ANP, and FANP respectively. The weight vector W^k combines the four vectors T_1^k, T_2^k, T_3^k and T_4^k by using the the arithmetic mean. The weight vector W^k, given by each decision maker can be calculated by:

$$W^k = [{w^k}_1, {w^k}_2, ...{w^k}_m], \quad where \quad {w^k}_i = \frac{\sum_{j=1}^{4} T_j^k}{4} \tag{2}$$

3. Construct of the group decision matrix: the decision matrices X^k are averaged to represent the group decision matrix X. The matrix X is given by:

$$X = (x_{ij}) \quad where \quad x_{ij} = \frac{\sum_{k=1}^{p} x_{ij}^k}{p}, \quad i = 1, ..n, \quad j = 1, .., m \tag{3}$$

4. Construct of the group weight vector: the weight vector W^k are averaged to represent the group weight vector W. The group vector W is given by

$$W = [w_1, w_2, ..., w_m] \quad where \quad x_{ij} = \frac{\sum_{k=1}^{p} x_{ij}^k}{p}, \quad i = 1, ..n, \quad j = 1, .., m \tag{4}$$

5. Apply the network selection algorithm based on group MADM: the Mahalanobis distance algorithm is applied to the group decision matrix X, by taking into account the weigh vector W. The ranking score V^i obtained is expressed by:

$$V^i = [a_{i1}, a_{i2}, ..., a_{in}] \tag{5}$$

6. Apply the individual Mahalanobis distance algorithm: each decision maker DM_k combines the Mahalanobis ranking algorithm with one of the weighting algorithms, in order to generate the vector V_i^k. This one, represents the score of the alternatives A_i. The vector V_i^k is defined by:

$$V_i^k = [{b^k}_{i1}, {b^k}_{i2}, ..., {b^k}_{in}] \quad where \quad i = 1, 2..., n \quad et \quad k = 1, 2...., p. \tag{6}$$

7. Calculate the consistency degree: in order to select the most valid ranking outcome which is the most acceptable by the all decision markers we introduce the consistency degree. This one, allows to measure the relationship between V_i and V_i^k. The consistency degree CD_i of the method between the group and individual ranking outcomes is given as follows:

$$CD_i = \frac{\sum_{k=1}^{p} C_i^k}{p} \tag{7}$$

The Pearson's correlation C_i^k between V^i and V_i^k can be calculated by:

$$C_i^k = \frac{n(\sum_{j=1}^{n} a_{ij} {b^k}_{ij}) - (\sum_{j=1}^{n} a_{ij})(\sum_{j=1}^{n} {b^k}_{ij})}{\sqrt{[n(\sum {a_{ij}}^2) - (\sum a_{ij})^2][n(\sum {b^k_{ij}}^2) - (\sum {b^k}_{ij})^2]}} \quad (8)$$

where n is the number of the available networks.

8. Ranking: the best group ranking has the highest value of CD_i.

3 Simulations and Results

3.1 The Simulation Scenario

In this simulation, the set A represents three candidates networks: UMTS, WIFI and WIMAX. The set C represents six attributes: cost per byte (CB), available bandwidth (AB), security (S), packet delay (D), packet jitter (J) and packet loss (L), and the set DM contains three decision makers: DM_1, DM_2, and DM_3. The measures of every criterion are given in table 2 and 3.

3.2 The Results of Simulation

We perform two simulations for two traffic classes namely background and streaming. We apply the four MADM methods Mahalanobis-W1, Mahalanobis-W2, Mahalanobis-W3 and Mahalanobis-W4 to the group decision matrix X and the group weight vector W, four group ranking outcomes are obtained which are G1, G2, G3 and G4 respectively.

By applying our new validation, the consistency degrees of G1, G2, G3 and G4 for background and streaming are given in table 1. We notice that G3 has the highest score, which means that the ANP is most suitable algorithm which should be used to weigh different criteria for background and streaming traffic.

Table 1. Pearson's correlation for group MADM for background and streaming

Group MADM	**G1**	**G2**	**G3**	**G4**
Consistency degree for background	0.99970	0.89990	0.99973	0.98360
Consistency degree for streaming	0.99590	0.78453	0.99763	0.94637

Table 2. The QoS metrics for the candidate networks

Network	**AB** $(mbps)$	**D** (ms)	**J** (ms)	**L**$(per10^6)$
UMTS	1.2	35	12	50
WIFI	6	110	15	60
WIMAX	8	100	20	80

Table 3. The values of cost and security for the candidate networks

	DM_1		DM_2		DM_3	
Network	**CB**(%)	**S**(%)	**CB**(%)	**S**(%)	**CB**(%)	**S**(%)
UMTS	60	70	65	60	70	80
WIFI	12	25	15	30	10	20
WIMAX	70	75	55	50	65	70

4 Conclusion

In this work, we have proposed new validation approach which can take into account different weighting methods and allows to select the group ranking outcome of Mahalanobis distance algorithm. The results of two simulations show that the ANP algorithm allows to assign the suitable relative importance value of each criterion, for background and streaming traffic.

References

1. Gustafsson, E., Jonsson, A.: Always best connected. IEEE Wireless Communications Magazine 10(1), 49–55 (2003)
2. Gazis, V., Houssos, N., Alonistioti, N., Merakos, L.: On the complexity of Always Best Connected in 4G mobile networks. In: 58th IEEE Vehicular Technology Conference, VTC, vol. 4, pp. 2312–2316 (2003)
3. Sheng-mei, L., Su, P., Zheng-kun, M., Qing-min, M., Ming-hai, X.: A simple additive weighting vertical handoff algorithm based on SINR and AHP for heterogeneous wireless networks. In: International Conference on Intelligent Computation Technology and Automation, ICICTA, pp. 347–350 (2010)
4. Lahby, M., Leghris, C., Adib, A.: A Hybrid Approach for Network Selection in Heterogeneous Multi-Access Environments. In: The Proceedings of the 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, pp. 1–5 (2011)
5. Lahby, M., Leghris, C., Adib, A.: A Novel Ranking Algorithm Based Network Selection For Heterogeneous Wireless Access. Journal of Networks 8(2), 263–272 (2013)
6. Yeh, C.-H., Chang, Y.-H.: Validating multiattribute decision making methods for supporting group decisions. In: IEEE Conference on Cybernetics and Intelligent Systems, pp. 878–883 (2008)
7. Saaty, T.L.: Decision Making for Leaders: The Analytic hierarchy Process for Decisions in a Complex World. RWS Publications (1998)
8. Mahmoodzadeh, S., et al.: Project Selection by Using Fuzzy AHP and TOPSIS Technique. In: Proceeding of World Academy of Science, Engineering and Technology, vol. 24 (October 2007)
9. Lee, J., Kim, S.: Using Analytic Network Process and Goal Programming for Interdependent Information System Project Selection. Computers and Operation Research 27(4), 367–382 (2000)
10. Mikhailov, L., Singh, M.G.: Fuzzy analytic network process and its application to the development of decision support systems. IEEE Transactions on Systems, Man and Cybernetics, 33–41 (2003)

Secured Geographic Routing Protocol for Vehicular Ad Hoc Networks (VANETs)

Mohammed Erritali[1], Bouabid El Ouahidi[1], and Daniel Bourget[2]

[1] Department of Computer Science
Mohamed V Agdal, University – L.R.I, Faculty of Sciences Rabat, Morocco
mederritali@yahoo.fr, ouahidi@fsr.ac.ma
[2] Telecom Bretagne, France
daniel.bourget@telecom-bretagne.eu

Abstract. A Vehicular ad hoc network called VANETs is a mobile network allowing to vehicles to communicate with each other in the absence of fixed infrastructure, with the aim of improving road safety through the exchange of alerts between neighborhood vehicles or to offer new comfort services to road users. The characteristics of these networks such as: shared wireless medium and the highly dynamic network topology pose a number of nontrivial challenges to security design.

In these networks each vehicle coordinates with every other vehicle in forwarding their packets to reach the destination. Since these vehicles operate in a physically insecure environment; they are vulnerable to different types of attacks such as the blackhole attack, Sybil attack, selective forwarding and altering routing information.

This paper proposes a security solution for VANETs using a pre-existing routing protocol Greedy Perimeter Stateless routing. In this solution, each node in a network has a list of its neighbor nodes including a shared secret key which is obtained by executing a key agreement Diffie Hellman, this key will be used by the AES symmetric encryption algorithm to generate a digital signature, after applying the MD5 hashing algorithm on the non-modifiable data of GPSR packets. Our idea consists that each vehicle verifies the integrity and authenticates the sender in the process of route discovery, Comparing with other recently proposed security routing protocols, our security solution needs less computation times in routing transactions because it use AES and does not need any centralized element in vehicular ad-hoc networks.

Keywords: Attack, Secure routing protocols, VANETs.

1 Introduction

Vehicular networks are a projection of intelligent transportation systems (ITS)[1,2] designating new technologies applied to transportation networks to improve the conduct and bring new services to road users by offering solutions that allow to:

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 311–315, 2013.

-Reduce road congestion
- Establish a system of traffic management that allows rapid intervention in case of incidents.
- Locate parking
- Report of a pedestrian passageway
-Notify violations of the Stop signal.

From examples of interesting solutions described above we can deduce that the vehicular communication ensures large improvement in terms of road safety, better utilization of resources such as time and fuel and new opportunities for entertainment applications to road users. The services offered in vehicular networks can distinguish several types of communication [2]: Vehicle to Vehicle communication (V2V), vehicle to infrastructure communications (V2I) and hybrid communications derived from the combination of these two types of communications.

Generally networks without infrastructure are called ad hoc networks, which is why V2V communications are called vehicular ad hoc networks (VANETs).

In these networks a successful attack against the road safety alerts could have catastrophic consequences such as loss of human lives. Therefore, making a vehicular communication network secure is not an extension but a primary concern. So far, only a few research efforts have addressed in VANETs security issues, focusing either on identification of their challenges, or proposing secure VANETs architectures.

In VANETs, routing is an important element designed to transmit road safety alerts to all vehicles in neighborhood, so it constitutes an ideal target for attacks which aims to prevent alert messages to reach their destinations.

The routing raises a significant number of problems which are not yet resolved such as packet modification, data injection and the generation of false messages, the rupture of packet forwarding, or deleting packets.

To remedy these vulnerabilities, several secure routing protocols for mobile networks [3, 4] have been proposed in which cryptographic primitives are involved, such as digital signatures, MACs (Message Authentication Code) or asymmetric encryption.

The other sections of the paper are structured as follows:

Section 2 briefly presents possible attack against routing protocols; section 3 describes secure routing protocols. Finally we present our extension to secure greedy perimeter Stateless routing.

2 Attack against Routing Protocols

Vehicular Ad hoc networks are dynamic and self-organized so any node can participate in routing and also uses a shared wireless medium to send packets .therefore there are no barriers to ensure that a malicious node cause disturbances in the circulating traffic. In the following section we will present some type of attack [9, 10, 11, 12] against routing protocols:

1- Blackhole attack: The black hole attack consists to insert a malicious node in the network. This node, by various means, will modify the routing tables to force the maximum of neighboring nodes to pass information through it. Then like a black hole in space, all the information that will go in it will never be retransmitted.

2- The type of attack Denial of Service DOS: The attacker sends an excessive amount of data to overload the network for the purpose to overflow the routing table of relay nodes.

3- Insertions of infinite loops: An attacker will modify the network routing mechanism with one or more nodes malicious, so that packets will be routed in infinite loops and will therefore consume network bandwidth.

4- Sybil attack: The "Sybil attack" consists that a malicious vehicle is masquerading as several vehicles. Thus it modifies the routing table that will become invalid. For example a malicious node witch pretend to be multiple nodes can gain a significant advantage for an election of master node.

5- Identity spoofing: A malicious node usurps address of a legitimate node as the source address in order to disseminate its messages on the VANETs network.

6- Wormhole attack: The attack of the wormhole requires the insertion of at least two malicious nodes A and B .These two nodes are connected by a link A-B. The goal of this attack is to trick the neighboring nodes about distances. Because in general a routing protocol seeks the shortest path in number of hops.

7- The alteration/modification of routing packets: A malicious node will retrieve a package and alter it, by adding false information (about the recipient, the sender or the data).

3 Secure Routing Protocols

SRP [6] is based on DSR protocol and requires a pre-existing secured association between source node and the destination node (secret key exchange) and it uses a MAC to ensure the confidentiality of the route learned from the source to the destination. These authors propose a mechanism to detect malicious behavior neighborhoods (Neighbor Lookup Protocol) and a mechanism for secure data transmission (Secure Message Transmission Protocol).

ARIADNE [7] is also based on DSR (Dynamic Source Routing) and it is used to authenticate messages routing with three mechanisms. The first is the use of a shared secret key between each pair of nodes, the second combines a secret shared between the nodes and the broadcast authentication (TESLA), and the third is the use of a digital signature.

SAODV [5] secures the modifiable routing data as the number of hops, which can for example be decremented by an attacker, and authenticates the fields that should not be changed using an RSA digital signature.

Figure 1 illustrates these secure routing protocols with used cryptographic techniques.

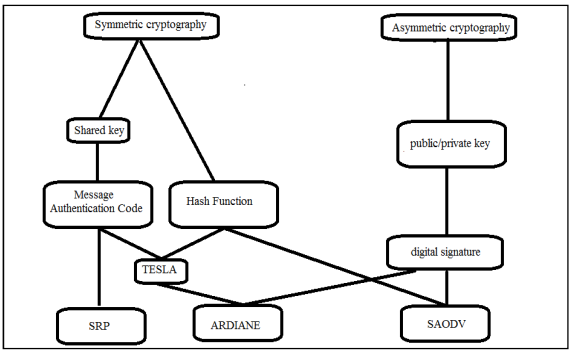


Fig. 1. Secure routing protocols with used cryptographic techniques

4 Adding Security Aspect in Greedy Perimeter Stateless Routing

Geographical routing protocols of vehicular networks have been developed without considering the security aspects against routing attacks. In this section we present our contribution to secure GPSR or generally geographical routing like it.

4.1 1st Extension: Establish Secret Keys

Our first contribution in this work is to propose an approach that allows to establish secret keys Diffie-Hellman between two neighboring vehicles (in a hop) when exchanging beacons packets to build direct neighbors tables of Greedy Perimeter Stateless Routing protocol [8]. The idea is to have neighbor's tables that contain secret keys that will be used as a symmetric encryption key.

4.2 2nd Extension: Adding a Symmetrical Digital Signature

In VANETs the digital signature will be the mechanism to ensure the integrity and the authentification of packets exchanged between two direct GPSR neighbors. Indeed, the mobility of nodes requires a minimal time of routing packet from the source to the destination, which is why we propose to use the AES encryption algorithm.

Figure 2 illustrates the process of creation of the digital signature.

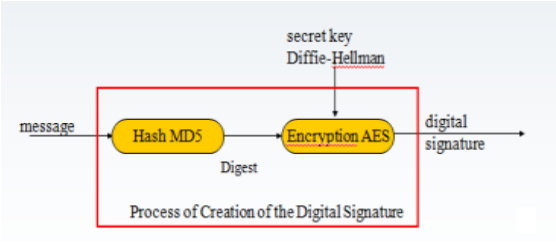


Fig. 2. GPSR digital signature

5 Conclusion and Perspectives

The problematic of communication and security in vehicular ad hoc networks attract more and more attention from research groups. Indeed, the ease of deployment of vehicular ad hoc networks and their spontaneous nature make them a compelling solution for the safety of drivers and their passengers. However, these networks require that all users work together to route information of other users on the same VANET network .This hypothesis and several other characteristics (mobility, bandwidth) make the problems of routing and security of communications in these networks a capital axis of research. In this paper we have presented GPSR protocol extensions. As perspectives of this work we will use to study how to define a level of trust between nodes and how to implement an intrusion detection system in a vehicular ad hoc network.

References

1. Khalfallah, S., Jerbi, M., Cherif, M.O., Senouci, S.-M., Ducourthial, B.: Expérimentations des communications inter-véhicules, Colloque Francophone sur l'Ingénierie des Protocoles (CFIP). Les Arcs, France (2008)
2. Jerbi, M.: Protocoles pour les communications dans les réseaux de véhicules en environnement urbain: Routage et GeoCast basés sur les intersections. Thèse, France (2008)
3. Idjiwa, A., Radhouane, B., Rebecca, B., Laurent, G.: Protocole de routage ad hoc sécurisé dans une architecture clusterisée, http://idjiwa.free.fr/wordpress/?cat=3
4. Hu, C., Perrig, A., Johnson, D.B.: Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols (2003)
5. Zapata, M.G.: Internet draft. In: Secure Ad hoc On-Demand Distance Vector (SAODV) Routing (September 15, 2005)
6. Papadimitratos, P., Haas, Z.J., Samar, P.: The Secure Routing Protocol (SRP) for Ad Hoc Networks (December 2002)
7. Hu, Y.-C., Perrig, A., Johnson, D.B.: Ariadne: A Secure On Demand Routing Protocol for Ad Hoc Networks
8. Karp, B., Kung, H.T.: Greedy Perimeter Stateless Routing for Wireless Networks. In: Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom 2000, Boston, MA (August 2000)
9. Douceur, J.R.: The Sybil Attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
10. Hu, Y.C., Perrig, A., Johnson, D.B.: Wormhole Detection in Wireless Ad Hoc Networks. Technical Report TR 01- 384, Department of Computer Science, Rice University (June 2002)
11. Karlof, C., Wagner, D.: Securing Routing in Wireless Sensor Networks: Attacks and Countermeasures. In: Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications, Anchorage, AK, USA, pp. 113–127 (May 2003)
12. Zhou, Z., Yow, K.C.: Geographic Ad Hoc Routing Security: Attacks and Countermeasures. Ad Hoc & Sensor Wireless Networks, 235–253 (March 2005)

Enhanced AntNet Protocol for Wireless Multimedia Sensor Networks

Ismail Bennis[1], Ouadoudi Zytoune[2], and Driss Aboutajdine[1]

[1] LRIT, unité associée au CNRST (URAC29), Faculté des Sciences, Université Mohammed V - Agdal, Rabat, Maroc
[2] Université Ibn Tofail, Kénitra, Maroc
i.bennis@fsr.um5a.ma, zytoune@gmail.com, aboutaj@fsr.ac.ma

Abstract. The field of wireless multimedia sensor networks (WMSN) attracts more and more the research community as being an interdisciplinary field of interest. This type of network becomes a low cost, multifunctional due to advances in micro-electromechanical systems, as well as proliferation and progression of wireless communications. However, transmitting collected multimedia information must meet the QoS criteria such as delay, bandwidth, packet loss rate, etc. Many routing protocols have been developed for WMSN. Recently, the most known ones are based on meta-heuristic, that show desirable properties of being adaptive, scalable, and robust. This paper presents a new routing protocol for WMSNs based on AntNet protocol which is inspired by the stigmergy-driven shortest path following behavior of biological ants. Our aims in this work is to provide as well as possible the best QoS in terms of delay and Packet Delivery Ratio(PDR). The results of simulations compared to the AODV show that our work has better delay and PDR.

Keywords: WMSN, AntNet, Delay, Routing protocol, QoS.

1 Introduction

WSNs consist of an amount of independent nodes equipped with sensing capabilities, wireless communication interfaces, limited processing and energy resources. Regarding applications of this type of network, they are characterized in general, with low bandwidth demands, and are usually delay tolerant. Recently the availability of CMOS cameras and microphones to capture multimedia content from the environment has allowed the arrival of new network type called WMSNs [1]. This novel network will not only enhance existing sensor network applications such as tracking or home automation, but they will also enable several new applications such as multimedia surveillance sensor networks, traffic avoidance[1].

When network size scales up, routing becomes more challenging and critical. Lately, biologically-inspired intelligent algorithms have been deployed to tackle this problem [2]. Using ants and other social swarms as models, software agents can be created to solve complex problems. One of the most successful swarm intelligence techniques is called Ant Colony Optimization (ACO) [3]. We can say

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 316–320, 2013.

that during these last two decades, this technique have served as an important source of inspiration for the design of novel algorithms and systems [4].

The remainder of the paper is organized as follows; in section II we discuss the implementation of AntNet for WMSN and propose our optimization for this protocol, the results of extensive simulations are shown in section III. Finally section IV, concludes the paper.

2 Implementation and Enhanced AntNet for WMSN

In this work we have made several optimizations of the initial version of the protocol (Antnet). These optimizations can be classified into two categories, the first categorie's is related to the adaptation of the protocol in the wireless and mobile environment in view that the AntNet protocole has been designed for fixed networks. Some of this modification was inspired from others works like [5] and [6]. Also we were led to make some change in the packet classes that can be divided into three different types:

Data packets represent the information which must be transmitted by the sonsor node, **Control packets** which contains two types : Forward ants used to search the path and Backward ants used to update the routing tables, and **Hello Control packets** that are used to have a list of available neighbor nodes.

Also each node has two data structures the first one is a routing table T_k containing triples of a destination address d, a nexthop n used to reach that destination d and a probability P_{nd}. This probability value P_{nd} stored in the routing table express the goodness of choosing the associated nexthop n to reach the destination d. The probabilities have to verify: $\sum_{n \in N_k} P_{nd} = 1$ where $d \in [1, N]$ and $N_k = neighbors(k)$. The routing table T_k is changed by incrementing (Eq 1) or decrementing (Eq 2) the probability P_{nd} according to the following way :

$$P_{nd} \leftarrow P_{nd} + r * (1 - P_{nd}) \tag{1}$$

$$P_{md} \leftarrow P_{md} * (1 - r), \forall m \in N_k, m \neq n \tag{2}$$

Where r is a reinforcement parameter in the interval [0,1].

The second categorie's comprises all functions and methods achieved and added in order to make the protocol more robust and more desirable for WMSNs. In what follows we will describe all these functions and methods incorporated into the protocol.

1. Optimization of the path.
   The goal here is to minimize the number of hops of the path that will be built by the forward ant , optimization is performed by removing redundant nodes and also nodes contained in the same neighborhood and are part of the path at its creation.
2. Initialisation of pheromone.
   The second change we made is to modify the manner that the next hop is chosen by the forward ant. As described in [7], a stochastic decision policy is applied to select the next node to move to, but if the destination is a neighbor, there is no need to calculate this probability. So we forward the ant directly to the neighbor.

3. Reactive mechanism.
   The basic version of the protocol is proactive, it tries to create a multitude of paths from fictitious sources to destinations, but this anticipation is not always useful because we can have paths that are not needed for the node source, and if we go to a large topology then it would be useless to create paths that will probably never be used. Therefore we planned to make the reactive protocol to avoid these problems. Also using the reactive manner will reduce congestion phenomena because the control packet are reduced.
4. List of ancestor.
   When a node needs to find a path to a destination, it generates the FANT which is responsible for finding the path. So, in order to increase the chance to find the path and accelerate research, we applied a mechanism such that each node receiving a FANT seeking a path to a particular destination, generates itself a FANT to this destination. But every node that participates in research must, in the case of success, reported its findings to the source node. To do this, it needs a list of history nodes which we called "list of ancestors", this list is optimized and assigned to each node receiving FANT before activating the process FANT at him.
5. List of destinations.
   In each node, we added a table that contains the state of this node towards each others node in the topology. we assume that the node can have three states: **Transmitter** :the node are the source of the trafic; **Intermediate** : the node is part of a path from source to destination; **None** :the node has no interaction with topology.
   We added also two lists in node : the first list will contain the destinations that the node should look the way by the FANT and this node participates as the transmitter, we call this list list_dest_src. The second list will contain the destinations that the node should look the way by the FANT and this node can participate as intermediate, we call this list list_dest_inter.
6. Update pheromone.
   The update of the pheromone value is governed by the equations 1 and 2, or this way of doing requires some time before converging to optimal solutions, and this becomes clearer in a large-scale topology. In order to accelerate the convergence we have changed the manner of the update, this is accomplished by penalizing the neighbors of a node that lead to the same destination but in greatest number of hops.

3 Simulation and Experimental Results

Our aims in this section is to compare our proposition withe AODV. The comparison regards the end-to-end delay,the packet delivery ratio (PDR) and the overhead. The simulation software used is NS2. For simplicity raisons we assum that a constant reinforcements model is used: $r = C, C \in [0, 1]$; and there is no mobility in this scenario. We have created many different problems and for each one we test the both protocols, the shown results are the average of this

problems. In the following subsection we describe the environment and scenario of the simulation.

Each simulation scenario is presented as follows : X nodes are randomly placed in an area of 1500*300 m^2, where:

$$X=[\![20;40;60;80;100;120;140;160;180;200;220;240;260;280;300]\!].$$

The data traffic is generated by 10% of nodes of topology as constant bit rate (CBR) sources sending four 512-byte packet per second. Each source starts sending at a random time between 20 and 180 seconds after the start of the simulation, and keeps sending till the end, the lenght of the simulation is 300s. At each number of nodes we repeat the simulation several times, and we calculate the average of the delays, overhead and the values of PDR found.

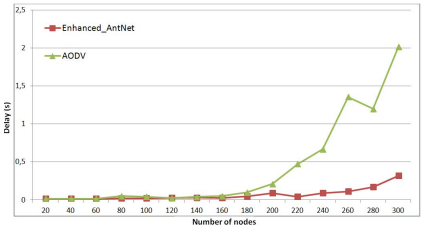


Fig. 1. Average delay VS Number of nodes

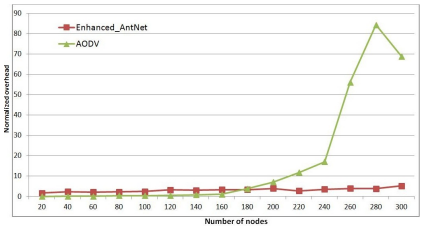


Fig. 2. Normalized overhead VS Number of nodes

Figure 1 shows the average delay experienced when using the both routing protocols under different number of nodes. We can see that we have the nearest delay when the number of nodes is less than 180. But, after that the dealy for AODV in scenario 1 and 2 grows up clearly to reach 2 seconds, and in case of our optimisation the delay does not exceed 0.4 seconds, this can be explained by the fact that the AODV broadcast a RRequest packet, or if the nodes have a significant number of neighbors, it will generate more packet control that will increase the congestion and thereby delaying the packet data.

In Figure 2 we can see also, in case of AODV, that the overhead becomes greater after 180 nodes, but for the Enhanced Antnet the overhead remains nearly constant for all situations; as explained above once the number of nodes in the neighborhood becomes greater, the traffic generated becomes greater too, which may explain the increase in the case of AODV.

Figure 3 shows clearly the gap between the value of the packet delivery ratio of the both protocols, we can perceive that the PDR for the Enhanced Antnet has some fluctuation but never drops below the 80%, in contrast to the AODV where PDR reaches very low values (up to 40%) when the number of node is 300.

So, all results show that we have almost the best delay, overhead and PDR, especially when the number of node increases, which is suitable for the WMSN.

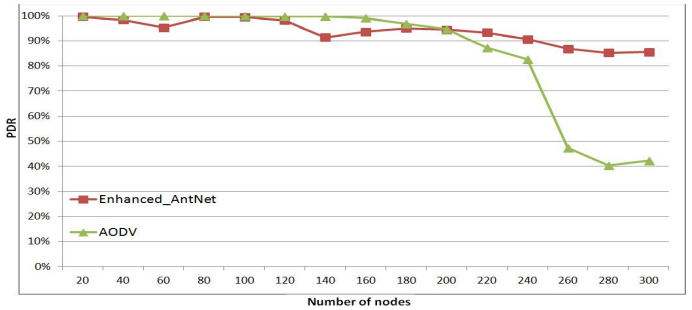


Fig. 3. PDR VS Number of nodes

4 Conclusion and Future Work

In this paper we have described an optimized AntNet routing algorithm for WMSN, our work enhances the basic AntNet protocol which is based on the most known swarm intelligence techniques ACO. Simulation results show that Enhanced AntNet has a performance advantage over AODV. The advantage exists in terms of packet delivery ratio, average end-to-end delay, and also the overhead. For future work, there is a point that we want to improve, is to introduce mobility in simulation scenario in order to have more realistic models and to compare our work with other protocol designed for WMSN.

References

1. Akyildiz, I.F., Melodia, T., Chowdhury, K.R.: A Survey on Wireless Multimedia Sensor Networks. Computer Networks 51, 921–960 (2007)
2. GhasemAghaei, R., Rahman, A., Gueaieb, W., Saddik, A.E.: Ant colony based reinforcement learning algorithm for routing in wireless sensor networks. In: IEEE Instrumentation and Measurement Technology Conference Proceedings (2007)
3. Dorigo, M., Blum, C.: Antcolonyoptimization theory: A survey. Theoretical Computer Science 344, 243–278 (2005)
4. Saleem, M., Di Caro, G.A., Farooq, M.: Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions. Information Sciences 181, 4597–4624 (2011)
5. Di Gianni, C., Ducatelle, F., Gambardella, L.M.: AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications 16, 443–455 (2005)
6. Camilo, T., Carreto, C., Silva, J.S., Boavida, F.: An energy-efficient ant-based routing algorithm for wireless sensor networks. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 49–59. Springer, Heidelberg (2006)
7. Di Caro, G.A., Dorigo, M.: AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, JAIR (1998)

Forest Fire Detection and Localization with Wireless Sensor Networks

Yassine Sabri[1] and Najib El Kamoun[2]

STIC Laboratory,
Chouaib Doukkali University, B.P: 20 , El Jadida MOROCCO
{sabriyassino,elkamoun}@gmail.com

Abstract. Location determination is an important problem for almost all WSN applications. However, obviously becomes an important target in the case of surveillance systems for forest fires do not have to support real-time monitoring of each point of an area at any time location early threat of fire.Solutions applying wireless sensor networks, on the other hand, can gather sensorial data values, like temperature and humidity, from all points of a area incessantly, day and night, and allow for fresh and precise data to the fire-fighting center rapidly.In this paper, we present the pattern and implementation of a model for the detection and localization of forest fires and the control environment. In order to know precisely and almost in real time the risk of forest fires, we design and are implementing a protocol type "Range-based" for a large-scale deployment based on convex hull. The effciency of the method is demonstrated by simulations, we show that our framework can supply quick response to forest fires while consuming energy effciently.

Keywords: Wireless sensor network (WSN), Localization, Monitoring System.

1 Introduction

The rapid progress of wireless communication and the availability of many lightweight, small-size, and portable computing devices have made a great impact on community. These technologies have made the dream of communication anytime and anywhere possible. People using a mobile device can surf the web as well as talk with their friends while they are moving all over the world. To provide more value-added applications, such as geographical navigation for tourists, advertising messages for local potential customers, 911 emergency service for subscribers, etc., we need to acquire the location information of a mobile user (MU) with a mobile device.

Among the WSNs applications, forest fire detection can be very helpful in avoiding human and material losses. For instance [4],the average annual number of forest fires throughout the Mediterranean basin is now close to 50 000,twice that during the 70.In Morocco only, almost 500 forest fires were registered in 2010.In the last five years the average of fire occurrences was 540 with an area 4500 ha, with an estimated annual loss of 2,902,667 US [3].

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 321–325, 2013.

Other, some proposals have been published that consider the use of WSNs to monitor and detect forest fires [1]. The use of multiple sensor sources, and the deployment of the sensor nodes in areas not visible to the satellite, increase the probability of a more accurate and early fire detection. WSNs event detection applications are in essence fusion information processes [3]. The rest of this document is organized as follows. The related work is presented in Section 2. In Section 3, we introduce the proposed algorithm based on a threshold method, and its evaluation. In Section 4, we discuss the proposed algorithmbased on the Dempster-Shafer theory, along with the results of the evaluation of the proposed algorithm. Finally, in Section 5 the conclusions and future work are presented.

2 Localization Technique

Initially, each anchor broadcasts its position. A node can therefore be deduced the distance between each of the anchors We use the technique SumDist (Savvides et al., 2002) for estimating distances adding the distances between separated sensor nodes of an anchor.Upon receiving the position of a anchor, a node considers the following cases:

- If it receives directly the position of the anchor, he deduces they are neighbors and therefore it located on the circle centered at the anchor or radius of a circle is r.
- If it receives the position by an intermediate node, it concluded that it is not neighbor of the anchor and therefore it is not inside the circle of radius r centered in anchor .

So,when a node u receives a position of an anchor A, it estimates the distance to this anchor with Sum-Dist and draws one or two circles. In fact, if $(A \in N_\Lambda(u))$, u knows d_{Au} and deduces that it is on the circle $\mathcal{C}_{Au}$ of radius equals to d_{Au} and centered in A. If $(A \notin N_\Lambda(u))$ then u knows that it is not inside the circle of center A and radius r otherwise A and u would be neighbors. Moreover, u knows the estimated distance to A, $\hat{d}_{Au}$ deduced by Sum-Dist. By triangular inequality, $\hat{d}_{Au} \leq d_{Au}$. u applies this technique to each received anchor position. So, u is inside the circle $\mathcal{C}_{\overline{Au}}$ of center A and radius $\hat{d}_{Au}$. Thus, the intersection of circles defines a cloud of points $\mathcal{S}_u$. the center of gravity of the convex hull of this cloud $conv(\mathcal{S}_u)$ represents the estimated position of u. The main design of the Slsng, which is a simple finite state machine. As shown in figure 1, a node running Slsng is in one of four states at any time: (i) Sensor not estimated, (ii) Sensor estimated, (iii) estimated Anchor,and (iv) improve the accuracy. Transitions between the states are triggered by events.

3 Method for Forest Fire Detection

Our algorithm uses a fusion information method known as the threshold method [3]. The algorithm is based on the state machine shown in Figure 2, which defines

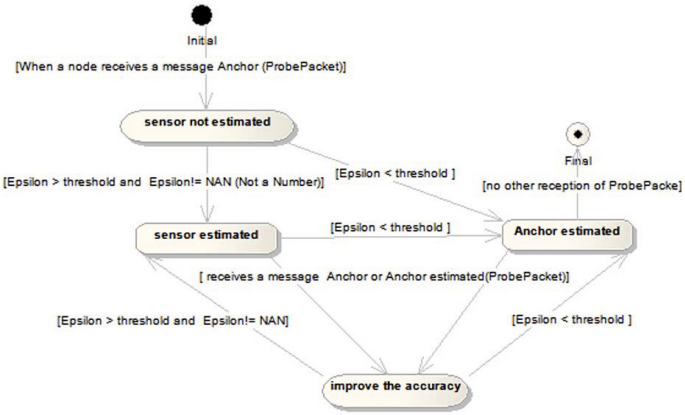


Fig. 1. State machine diagram for Sensor node not estimated

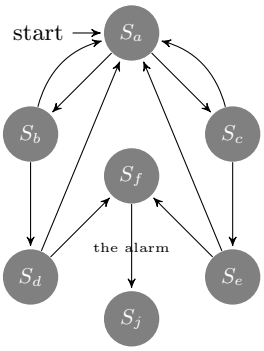


Fig. 2. *Our algorithm state machine*

five states. The transition from one state to another is generated when a relevant change in the values of temperature, light or relative humidity is detected, indicating the probable existence of a fire.

The initial state is the S_a and represents the normal (i.e. no fire) environmental conditions. The states State1 and S_c are transitional states, since they indicate the probable occurrence of a night fire or a day fire, respectively. The S_d may represent the sunrise, whereas the S_e may indicate that the mote was exposed to direct sunshine. Finally, the S_e represents the presence of a fire. In normal conditions, the state machine is in S_a. Even though the values of the temperature, light and relative humidity are collected, in the S_a only the temperature value is evaluated. Every time the temperature is registered, the ratio between the average of the values of a sliding window of size WT and the new temperature value, is calculated. The sliding window contains the most recent WT temperature values recorded. If this ratio is greater than `tr_threshold`,

it means a large change in the temperature value and a that possibly a fire has been detected. To determine if it is night fire, the rate of change of the light is evaluated, in a similar way as we did with the temperature. Therefore, if the ratio between the average of the values of the sliding window of size WL and the most recent light value is greater than `tl_threshold`, the machine changes to the S_b. Otherwise, it changes to the S_c. It is important to note that while in S_a, in $temp\_s_0$ we store the last stable temperature value; that is, the value before the machine moved from S_a. If the machine is in S_b, we calculate the ratio of the average of the values of the sliding window of size WH and the most recent value of the relative humidity. If this ratio is less than `th_threshold`, the machine moves to S_d; otherwise, it goes back to S_a. While the machine is in S_d, the ratios are still computed, and if they are greater (or smaller, in the case of humidity) than their respective thresholds (i.e., the temperature is still increasing while the relative humidity is decreasing), the machine moves to S_f to determine the localization of fire,and systematically moves to $S_j$and an alarm is triggered, indicating the probable occurrence of a fire.

4 Experiment and Results

4.1 Simulation Environment

Experiments were built upon the J-Sim simulator [9] dedicated to WSN simulations. It is a compositional, component-based simulation environment. It is built upon the concept of autonomous component programming model. J-Sim is developed entirely in Java. The signal attenuation due to obstacles or other factors (e.g. use of unidirectional antennas) is simulated in J-Sim.The positions to estimate are generated randomly on a surface $\mathcal{A} = L \times L$ with dimensions of experimentation varying between 100×100 to 800×800 and a density of sensor $d = 20$, each configuration obtained is repeated for each of the two methods.the range of the sensors was set at 14.

The simplest way to describe localization performance is to determine the residual error between the estimated and actual node positions for every node in the network, sum them and average the result. Broxton et al in [2] do this using the mean absolute error metric (MAE), which, for each of n nodes in the network, calculates the residual between the estimated nodes and actual coordinates.

$$MAE = \frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 - (y_i - \hat{y}_i)^2}{n} \tag{1}$$

with (x_i, y_i) the real position and $(\hat{x}_i, \hat{y}_i)$ the estimated postilions . Figure 3 shows the evolution the location accuracy convergence. depending on the size of networks. in first graph, the convergence time increases linearly with the dimension, and in the second graph represents the evolution of convergence time that is the time when the Metric MAE is stabilized over time. convergence time with our method in a dimension 400×400 corresponds to $65s$ in $\delta = 0\%$ and $190s$ with $\delta = 10\%$. In fact, the main particularity of our protocol is that the complexity does not depend on the dimension of networks,but the number of nodes constructing the convex hull.

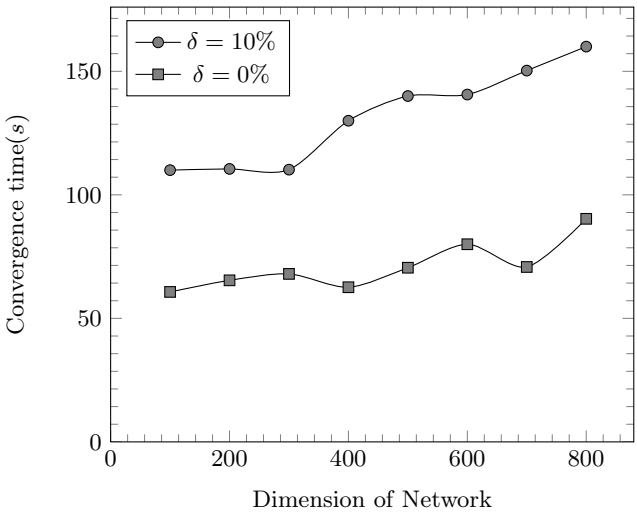


Fig. 3. *Convergence time with density of anchors* $\alpha = 20\%$

5 Conclusion

In this paper, we present a simple framework for a wireless sensor network to be used for forest fire localization and detection. As future work, we plan to develop a new algorithm to detect forest fi res based on the use of evolutionary programming.

References

1. Bahrepour, M., Meratnia, N., Havinga, P.J.M.: Automatic fire detection: A survey from wireless sensor network perspective (December 2008)
2. Broxton, M., Lifton, J., Paradiso, J.A.: Localization on the pushpin computing sensor network using spectral graph drawing and mesh relaxation. SIGMOBILE Mob. Comput. Commun. Rev. 10, 1–12 (2006)
3. Nakamura, E.F., Loureiro, A.A.F., Frery, A.C.: Information fusion for wireless sensor networks: Methods, models, and classifications. ACM Comput. Surv. 39(3) (September 2007)
4. FAO Forestry Paper. Réunion de la FAO sur les politiques nationales ayant une incidence sur les incendies de forêt. FO (1998)

On Ensuring End-to-End Quality of Service in Inter-Domain Environment

Sara Bakkali, Hafssa Benaboud, and Mouad Ben Mamoun

LRI, Faculty of Sciences at Rabat, Mohammed V-Agdal University,
Rabat, Morocco
bakkalisara@gmail.com,
{benaboud,ben_mamoun}@fsr.ac.ma

Abstract. Internet use is in permanent evolution, Internet traffic has become more and more diversified, and each type of traffic has its own Quality of Service (QoS) requirements. Different problems appear with this diversity. One of the main problems is the difficulty in ensuring QoS for traffics that cross multiple domains or Autonomous Systems (ASs). We propose in this paper a new method which ensures the end to end QoS requirements over multiple ASs. This method keeps the same values of QoS parameters required by the traffic, even during its passage across several ASs. This paper explains the problem of end-to-end QoS and gives a detailed description of our new approach.

Keywords: Inter-domain routing, QoS.

1 Introduction

Early, before the appearance of different types of traffic, which requires more bandwidth, less delay, and other necessary parametres, Internet traffic didn't have any quality of service constraints. However, today network traffics are very diverse, and each type of traffic has its own QoS requirements. Ensuring QoS has become an additional task for the network. Various models have been implemented to ensure QoS in intra-domain case. Nevertheless, in the inter-domain case the problem is not resolved yet.

Objective of this paper is to propose a new method that ensures the end-to-end QoS constraints for traffic services across multiple domains. Services involved in our approach include real time services such as voice and video telephony and conference, as well as services that require high capacity interconnections like links between scientific sites or cloud services, which are provided by different domains.

This paper is organized as follows. Section 2 gives related works and discusses the inter-domain problem. Next in section 3, we describe our approach that ensures end-to-end QoS over multiple domains, and finally, in section 4, we conclude this paper and give future works.

V. Gramoli and R. Guerraoui (Eds.): NETYS 2013, LNCS 7853, pp. 326–330, 2013.

2 Inter-Domain Problem and Related Works

Several solutions and technologies have been proposed and implemented to provide QoS within the same domain (AS), such as IntServ (Integrated Services) [1] model, DiffServ (Differentiated Services) [2] model or even MPLS [3].

However, a serious problem is posed when the traffic crosses another domain (AS). This problem is due to the fact that QoS constraints, required by the client and which the operator undertakes to provide (usually specified in the Service Level Agreement, SLA), are defined in the classes of service. While the definition of the classes of service is assured by the domain administrator, they are consequently specific to each domain, and are valid only within this domain. In this case, in the transition to another domain the QoS constraints offered to the traffic will not be the same as in the source domain, therefore the QoS required by the client at the beginning will not be provided from the end-to-end until its destination.

A various studies and several solutions have been proposed to ensure the quality of service (QoS) in inter-domain; each solution suggests a specific approach to treat the subject. All solutions proposed in the different works [4][5][6][7][8] treat the end-to-end inter-domain QoS problem by focusing on one of the two following aspects:

1. Paths computation: by proposing new algorithms,
2. Management functions: by proposing a new model based on new procedures and methods or on existing technologies (e.g. MPLS).

However, these inter-domain solutions do not provide to clients traffic the same required QoS as in its source domain. In this context, we introduce this paper which presents a solution that offer to clients traffic the same QoS constraints even in passing to another domain.

3 Proposed Solution Description

3.1 Approach Definition

To ensure continuity of QoS constraints offered to the client even after the transition to other domains, we introduce a new method that provides a new mechanism for inter-domain traffic treatment.

The basic idea in our approach is to designate in each domain a server responsible for the management of the different classes of service, named the **Class Manager** (CM). On this server we define a table, named **Class Table** (CT) that contains all information concerning the different classes defined in this domain (such as bandwidth, loss rate, delay, etc.).

Once the CM of each domain filled its CT, it sends it to a neighbouring domain. In this way, each CM has all the information about its neighbours classes of services, and then, upon receiving a packet from the neighbouring domain, the router in the current domain can classify it in a class that has the

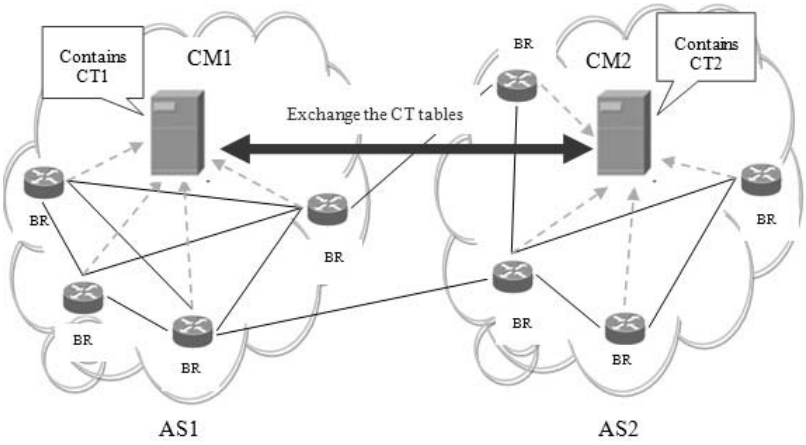


Fig. 1. Mechanism of the proposed Solution

same characteristics as the source class. In this manner, the client flow retains the same QoS constraints throughout its path to the destination, and receives the same treatment from end-to-end. The diagram in the figure [1] resumes this mechanism.

3.2 CT Table Structure

The class table is structured according the following fields:

1. AS number: to identify domain associated with the class.
2. Class number: to identify the class of service.
3. Bandwidth: to indicate the percentage of bandwidth allocated to the class.
4. Priority: to specify the priority level of the class.
5. Queue-limit: to specify the maximum number of packets that the queue can hold for this class.
6. Random-detect: to indicate whether the algorithm WRED is enabled on that class.

We note that, to ensure a certain correspondence between the CT tables of the different domains, we define in the CT table only class parameters common between various router's constructors, which are basic parameters used by the different constructors to characterize a class of service. However, the CT table fields can be adapted later to parameters used by the router's constructor implemented in the network. The parameters used in the CT table must be specified in the agreement established between the domains as we will explain later in this paper.

3.3 Sending Information from Routers to CM Sever

As we have already mentioned, routers receive the customer traffic and class it by applying mechanisms of adopted QoS intra-domain model. Information concerning parameters relatives to every class defined on a router is in the router configuration file. The border router sends this file to the CM server. Once received, the CM server executes a script to retrieve information concerning classes of service, and to place them in a file named CT, this file represents the class table which is responsible of storing information concerning all classes of service defined in the domain. However, sending the entire border routers configuration file to the CM server presents a serious security risk, this point will be discussed in future work.

3.4 Exchanging Tables between CM Servers

The communication between the CM servers of all domains uses the TCP protocol. Information exchanged between CM servers is included in TCP streams. So, before sending its CT table, each CM server establish a TCP session with the one holding in the neighbouring domain. Once the session TCP is established, the first message exchanged between both CM servers is the identification message, which allows each CM server to become identified by its neighbour, by sending its IP address and AS number. After the identification, CM servers exchange their CT tables by sending a set of messages to announce their classes of services, called Announcement Messages. Every message contains various parameters values relatives to every class defined in the domain.

When the CM server receives the message transporting the informations from its neighbour, it stores them in its CT table. In this way, when the CM server receives the totality of messages, it will have all information concerning all classes defined in the neighbouring domain.

The last type of message is the Update Message, which is sent by a CM server when there is an addition or modification of a class of service defined in its domain. The update message has the same structure as the announcement message.

3.5 Broadcasting CT Tables

Once a CM server receives its neighbour CT table, it diffuses it to the routers of its domain. Hence, all domain routers will possess all information about class of service defined in the neighbouring domain, and can use this information to create and configure classes of service which will have same values of QoS parameters.

According to these classes of service, the receiving router will classify packets comming from the neighbouring domain to be forwarded in the current domain with the same QoS constraints.

3.6 Agreements between Domains

The proposed solution is mainly based on agreements established between domains. Indeed, the information exchanged between domains in CT tables is very important and very sensitive, and the domain administrators have to negotiate and establish an agreement that will manage relations between domains. The agreement also defines how the tables exchange will be charged.

4 Conclusion and Future Work

Different problems appear with the evolution of Internet. One of the main problems aims the inter-domain routing with a guaranteed quality of service (QoS). Today, the Internet traffic is diversified, and each type of traffic has its own requirements. In this paper, we proposed a new mechanism which could ensure end-to-end QoS over multiple AS. We described it and we gave details of its operations and its components.

Our mechanism keeps the same QoS required and our objective is achieved. However, we can't confirm its performances before a study is done. So, our future work will focus on the performance evaluation of the new solution in order to show its advantages and limitations.

References

1. Wroclawski, J.: The Use of RSVP with IETF Integrated Services. IETF Standard Track, RFC 2210 (1997)
2. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. IETF Informational, RFC 2475 (1998)
3. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. IETF Informational, RFC 3031 (2001)
4. Farrel, A., Vasseur, J.-P., Ayyangar, A.: A Framework for Inter-Domain Multiprotocol Label Switching Traffic Engineering. IETF Informational, RFC 4726 (2006)
5. Howartha, P., Boucadairb, M., Flegkasa, P., Wanga, N., Pavloua, G., Morandb, P., Coadicb, T., Griffinc, D., Asgarid, A., Georgatsosen, P.: End-to-end quality of service provisioning through inter-provider traffic engineering. Computer Communications 29, 683–702 (2006)
6. Yampolskiy, M., Hommel, W., Danciu, A., Metzker, G., Hamm, K.: Management-aware Inter-Domain Routing for End-to-End Quality of Service. International Journal on Advances in Internet Technology 4, 60–77 (2011)
7. Frikha, A., Lahoud, S., Cousin, B.: Hybrid Inter-Domain QoS Routing with Crankback Mechanisms. In: Balandin, S., Koucheryavy, Y., Hu, H. (eds.) NEW2AN 2011 and ruSMART 2011. LNCS, vol. 6869, pp. 450–462. Springer, Heidelberg (2011)
8. Frikha, A., Lahoud, S.: Hybrid Inter-Domain QoS Routing based on Look-Ahead Information. IRISA's Interne Publications de, PI 1946 (2010)

Author Index