Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants

Fabiola Greve, Pierre Sens, Luciana Arantes, Véronique Martin

To cite this version:

Fabiola Greve, Pierre Sens, Luciana Arantes, Véronique Martin. Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants. [Research Report] RR-6088, INRIA. 2011, pp.29. inria-00122517v4

HAL Id: inria-00122517

https://hal.inria.fr/inria-00122517v4

Submitted on 30 Mar 2011

Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants

Fabíola Greve — Pierre Sens — Luciana Arantes — Véronique Martin

N° 6088 — version 4

version initiale Janvier 2007 — version révisée Février 2011

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6088--FR+ENG

Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants

Fabíola Greve [*] , Pierre Sens , Luciana Arantes , Véronique Martin [†]

Thème COM — Systèmes communicants
Projets Regal

Rapport de recherche n° 6088 — version 4 — version initiale Janvier 2007 — version révisée Fvrier 2011 — 29 pages

Abstract: The distributed computing scenario is rapidly evolving for integrating self-organizing and dynamic wireless networks. Unreliable failure detectors are classical mechanisms which provide information about process failures and can help systems to cope with the high dynamism of these networks. A number of failure detection algorithms has been proposed so far. Nonetheless, most of them assume a global knowledge about the membership as well as a fully communication connectivity; additionally, they are timer-based, requiring that eventually some bound on the message transmission will permanently hold. These assumptions are no longer appropriate to the new scenario. This paper presents a new failure detector protocol which implements a new class of detectors, namely $\diamond S^{\mathcal{M}}$, which adapts the properties of the $\diamond S$ class to a dynamic network with an unknown membership. It has the interesting feature to be time-free, so that it does not rely on timers to detect failures; moreover, it tolerates mobility of nodes and message losses.

Key-words: Unreliable failure detector, dynamic distributed systems, wireless mobile networks, asynchronous systems

[*] DCC - Computer Science Department / Federal University of Bahia
[†] LIP6 - University of Paris 6 - INRIA

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Implémentation asynchrone de détecteurs de fautes sans connaître les participants et en présence d'une connectivité partielle

Résumé : L'informatique répartie intègre de plus en plus des réseaux sans fil dynamiques et auto-organisant. Les détecteurs de fautes non fiables sont un mécanisme classique fournissant des informations sur les processus défaillants. Ils peuvent être particulièrement utiles pour gérer le dynamisme important de ces réseaux. De nombreux algorithmes de détection de fautes ont déjà été proposés. Cependant, la plupart d'entre eux considèrent un ensemble connu de processus interconnectés par un réseau complètement maillé. De plus, ces détecteurs reposent sur des temporisateurs et supposent à terme des bornes sur les délais de transmission des messages. Des telles hypothèses ne sont pas réalistes dans les environnements dynamiques. Cet article présente un nouveau protocole pour détecter les fautes qui implémente une nouvelle classe de détecteurs, appelé $\diamond S^{\mathcal{M}}$, qui adapte les propriétés de la classe $\diamond S$ aux réseaux dynamiques avec l'absence de la connaissance des participants. Notre détecteur ne repose sur aucun temporisateur ; de plus, il tolère la mobilité des nœuds et la perte de messages.

Mots-clés : détecteurs de fautes non fiables, sytèmes distribués dynamiques, réseaux sans fil mobiles, systèmes asynchrone

1 Introduction

The distributed computing scenario is rapidly evolving for integrating unstructured, self-organizing and dynamic systems, like mobile wireless networks [1]. Nonetheless, the issue of designing reliable services which can cope with the high dynamism of these systems is a challenge. Failure detector is a fundamental service, able to help in the development of fault-tolerant distributed systems. *Unreliable failure detectors*, namely FD, can informally be seen as a per process oracle, which periodically provides a list of processes suspected of having crashed [2]. In this paper, we are interested in the class of eventually strong FDs, denoted $\diamond S$. Those FDs can make an arbitrary number of mistakes; yet, there is a time after which some correct process is never suspected (*eventual weak accuracy* property). Moreover, eventually, every process that crashes is permanently suspected by every correct process (*strong completeness* property). $\diamond S$ is the weakest class allowing to solve consensus in an asynchronous system (with the additional assumption that a majority of processes are correct). Consensus allows a set of processes to agree upon a common value, among the proposed ones, and it is in the heart of important middleware, e.g., group communication services, transactions and replication servers.

This paper focuses on FDs for mobile and unknown networks, such as WMNs (wireless mesh networks) [3], WSNs (wireless sensor networks) [4]. These kind of networks share the following properties: (1) a node does not necessarily know all the nodes of the network; (2) message transmission delay between nodes is highly unpredictable; (3) the network is not fully connected, thus a message sent by a node might be routed through a set of intermediate nodes until reaching its destination; (4) a node can move around and thus change of neighborhood.

The nature of wireless mobile networks creates important challenges for the development of failure detection protocols. The inherent dynamism of these environments prevents processes from gathering a global knowledge of the system's properties. The network topology is constantly changing and the best that a process can have is a local perception of these changes. Global assumptions, such as the knowledge about the whole membership, the maximum number of crashes, full connectivity or reliable communication, are no more realistic.

A number of failure detection algorithms has been proposed so far. Nonetheless, most of current implementations of FDs are based on an all-to-all communication approach where each process periodically sends "I am alive" messages to all processes [5, 6, 7]. As they usually consider a fully connected set of known nodes, these implementations are not adequate for dynamic environments. Furthermore, they are usually timer-based, assuming that eventually some bound of the transmission will permanently hold. Such an assumption is not suitable for dynamic environments where communication delays between two nodes can vary due to mobility of nodes. In [8], Mostefaoui *et al.* have proposed an asynchronous implementation of FDs which is time-free. It is based on an exchange of messages which just uses the values of f (the maximum number of faults in the system) and n (the total number of nodes). However, their computation model consists of a set of fully connected initially known nodes. Some works [9, 10] focus on the heartbeat FD for sparsely connected networks with unknown membership. The heartbeat FD is a special class of FD which is time-free and is able to implement quiescent reliable communication. But, instead of lists of suspects, it outputs a vector of unbounded counters; if a process crashes, its counter eventually stops increasing. It is worth remarking that none of these works tolerate mobility of nodes. Few implementations of unreliable FDs focus on wireless mobile networks [11, 12, 13]. The fundamental difference between these works and ours is the fact that all of them are timer-based. The only exception is [14], but it does not tolerate node mobility.

1.1 Contributions

This paper has three contributions: *(i)* the definition of the $\diamond S^M$ class of FDs, which adapts the properties of the $\diamond S$ class to a dynamic system with an unknown membership; *(ii)* the identification of sufficient assumptions to implement it, and *(iii)* a new time-free FD algorithm that implements the class $\diamond S^M$ under a wireless mobile network. These contributions have been briefly announced in [15].

In order to implement unreliable FDs in an asynchronous dynamic system of mobile nodes, some assumptions about the underlying system should be done. Due to arbitrary arrivals and departures, moves and crashes, dynamic systems can be characterized by the succession of unstable periods followed by stable periods. During the unstable periods, certain situations could block the computation. For example, the rapid movement of nodes or, numerous joins or leaves along the execution, could prevent any useful computation. Thus, the system should present some stability conditions that when satisfied for longtime enough will be sufficient for the computation to progress and terminate. In the classic model of distributed computation, these stable conditions are related mainly to synchrony requirements on process speed and message delays [2]. For the protocol proposed herein, since the computation model is based on a message exchange pattern and additionally the system composition is unknown, the stable conditions relate to some behavioral properties that nodes should satisfy in the network. For example, a mobile node should interact at least once with some others in order to be known in the system. Moreover, a node should stay within its neighborhood for a sufficient period of time, the time for the exchange of messages in order to be able to update its state and the state of the network with recent information. Thus, the second contribution of this paper is to identify behavior assumptions able to implement the properties of $\diamond S^M$ in mobile networks with unknown membership.

The third contribution is the proposition of a FD algorithm that implements $\diamond S^M$. It is suitable for wireless mobile networks and has the following innovative features that allow for scalability and adaptability: (i) it is conceived for a network whose membership is unknown and whose communication graph is not complete; (ii) it tolerates node mobility, beyond arbitrary joins and leaves; (iii) the failure detection uses local information (for the membership of the neighborhood), instead of traditional global information, such as n and f; (iv) the failure detection is time-free, thus the satisfaction of the properties of the FD does not rely on traditional synchrony assumptions, but on a message exchange pattern followed by the nodes; (v) the message exchange pattern is based on local exchanged information among neighbors and not on global exchanges among nodes in the system. Initially, each node only knows itself. Then, it periodically exchanges a QUERY-RESPONSE pair of messages with its neighbors, that is, those nodes from which it has received a message previously. Then, based only on the reception of these messages and the partial knowledge about the system membership (i.e., its neighborhood), a node is able to suspect other processes or revoke a suspicion. This information about suspicions and mistakes is piggybacked in the QUERY messages and eventually propagated to the whole network.

As far as we are aware of, this paper brings the first time-free FD algorithm for networks with unknown membership that tolerates mobility of nodes. Correctness proofs are given that the algorithm can implement FDs of class $\diamond S^M$ when behavioral properties are satisfied by the underlying system. Performance experiments of the proposed FD show that it exhibits a good reactivity to detect failures and revoke false suspicions, even in presence of mobility. We believe that our FD of class $\diamond S^M$ may be successful adopted to implement coordination protocols in a dynamic set, such as the one proposed by Greve *et al.*[16], who present a solution for the fault-tolerant consensus in a network of unknown participants with minimal synchrony assumptions.

The rest of the paper is organized as follows. Section 2 defines the model and specifies the $\diamond S^M$ FD class. Section 3 identifies assumptions to implement those FDs. Section 4 presents a time-free FD of the $\diamond S^M$ class and Section 5 its correctness proofs. Simulation performance results are shown in Section 7. Some related work are described in Section 6. Finally, Section 8 concludes the paper.

2 Model and Problem Definition

The wireless mobile network is a dynamic system composed of infinitely many processes; but each run consists of a finite set Π of $n > 1$ mobile nodes, namely, $\Pi = \{p_1, \ldots, p_n\}$. Contrarily to a static network, the membership is unknown, thus processes are not aware about Π and n, because, moreover, these values can vary from run to run; this coincides with the *finite arrival model* [17]. This model is suitable for long-lived or unmanaged applications, as for example, sensor networks deployed to support crises management or help on dealing with natural disasters. There is one process per node; each process knows its own identity, but it does not necessarily knows the identities of the other processes. Nonetheless, nodes communicate by sending and receiving messages via a packet radio network and may make use of the broadcast facility of this communication medium to know one another. There are no assumptions on the relative speed of processes or on message transfer delays, thus the system is *asynchronous*; there is no global clock, but to simplify the presentation, we take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers. A process may fail by *crashing*, *i.e.*, by prematurely or by deliberately halting (switched off); a crashed process does not recover.

The network is represented by a communication graph $G = (V, E)$ in which $V = \Pi$ represents the set of mobile nodes and E represents the set of logical links. The topology of G is dynamic due to arbitrary joins, leaves, crashes and moves. A bidirectional link between nodes p_i and p_j means that p_i is within the wireless transmission range of p_j and vice-versa.

If this assumption appears to be inappropriate for a mobile environment, one can use the strategy proposed in [18] for allowing a protocol originally designed for bidirectional links to work with unidirectional links. Let R_i be the transmission range of p_i, then all the nodes that are at distance at most R_i from p_i in the network are considered 1-hop *neighbors*, belonging to the same *neighborhood*. We denote N_i to be the set of 1-hop *neighbors* from p_i; thus, $(p_i, p_j) \in E$ *iff* $(p_i, p_j) \in N_i$. Local broadcast between 1-hop neighbors is *fair-lossy*. This means that messages may be lost, but, if p_i broadcasts m to processes in its neighborhood an infinite number of times, then every p_j in the neighborhood receives m from p_i an infinite number of times, or p_j is faulty. This condition is attained if the MAC layer of the underlying wireless network provides a protocol that reliably delivers broadcast data, even in presence of unpredictable behaviors, such as fading, collisions, and interference; solutions in this sense have been proposed in [19, 20, 21].

Nodes in Π may be mobile and they can keep continuously *moving* and *pausing* in the system. When a node p_m moves, its neighborhood may change. We consider a *passive mobility* model, i.e., the node that is moving does not know that it is moving. Hence, the mobile node p_m cannot notify its neighbors about its moving. Then, for the viewpoint of a neighbor, it is not possible to distinguish between a moving, a leave or a crash of p_m. During the neighborhood changing, p_m keeps its state, that is, the values of its variables.

2.1 Stability Assumptions

In order to implement unreliable failure detectors with an unknown membership, processes should interact with some others to be known. If there is some process in the system such that the rest of processes have no knowledge whatsoever of its identity, there is no algorithm that implements a failure detector with weak completeness, even if links are reliable and the system is synchronous [22]. In this sense, the characterization of the *actual membership* of the system, that is, the set of processes which might be considered for the computation is of utmost importance for our study. We consider then that after have joined the system for some point in time, a mobile process p_i must communicate somehow with the others in order to be known. Afterwards, if p_i leaves, it can re-enter the system with a new identity, thus, it is considered as a new process. Processes may join and leave the system as they wish, but the number of re-entries is bounded, due to the finite arrival assumption. One important aspect concerns the time period and conditions in which processes are connected to the system. During unstable periods, certain situations, as for example, connections for very short periods, the rapid movement of nodes, or numerous joins or leaves along the execution (characterizing a churn) could block the application and prevent any useful computation. Thus, the system should present some stability conditions that when satisfied for longtime enough will be sufficient for the computation to progress and terminate.

Definition 1. ***Membership*** *Let $t, t' \in \mathcal{T}$. Let $UP(t) \subset \Pi$ be the set of mobile processes that are in the system at time t, that is, after have joined the system before t, they neither leave it nor crash before t. Let p_i, p_j be mobile nodes. Let the $known_j$ set denotes the partial knowledge of p_j about the system's membership. The membership of the system is* *the* KNOWN *set.*

STABLE $\overset{def}{=} \{p_i : \exists t, t', s.t.\ \forall t' \geq t,\ p_i \in UP(t')\}$.

FAULTY $\overset{def}{=} \{p_i : \exists t, t',\ t < t',\ p_i \in UP(t) \wedge p_i \notin UP(t')\}$.

KNOWN $\overset{def}{=} \{p_i : (p_i \in \text{STABLE} \cup \text{FAULTY}) \wedge (p_i \in known_j, p_j \in \text{STABLE})\}$.

The actual membership of the system is in fact defined by the KNOWN set. A process is *known* if, after have joined the system, it has been identified by some stable process. A *stable* process is thus a mobile process that, after had entered the system for some point in time, never departs (due to a crash or a leave); otherwise, it is *faulty*. A process is faulty after time t, when, after had entered the system at t, it departs at $t' > t$. The STABLE set corresponds to the set of *correct* processes in the classical model of static systems.

Assumption 1. ***Connectivity*** *Let $G(\text{KNOWN} \cap \text{STABLE}) = G(S) \subseteq G$ be the graph obtained from the stable known processes. Then, $\exists t \in \mathcal{T}$, s.t., in $G(S)$ there is a path between every pair of processes $p_i, p_j \in G(S)$.*

This connectivity assumption states that, in spite of changes in the topology of G, from some point in time t, the set of known stables forms a *strongly connected component* in G. This condition is frequently present in the classical model of static networks and is indeed mandatory to ensure dissemination of messages to all stable processes and thus to ensure the global properties of the failure detector [2, 22, 23, 24].

2.2 A Failure Detector of Class $\Diamond S^M$

Unreliable failure detectors provide information about the liveness of processes in the system [2]. Each process has access to a local failure detector which outputs a list of processes that it currently suspects of being faulty. The failure detector is *unreliable* in the sense that it may erroneously add to its list a process which is actually correct. But if the detector later believes that suspecting this process is a mistake, it then removes the process from its list. Failure detectors are formally characterized by two properties: (i) *Completeness* characterizes its capability of suspecting every faulty process permanently; (ii) *Accuracy* characterizes its capability of not suspecting correct processes. Our work is focused on the class of *Eventually Strong* detectors, also known as $\Diamond S$. Nonetheless, we adapt the properties of this class in order to implement a FD in a dynamic set. Then, we define the class of *Eventually Strong Failure Detectors* with *Unknown Membership*, namely $\Diamond S^M$. This class keeps the same properties of $\Diamond S$, except that they are now valid to known processes, that are stable and faulty.

Definition 2. ***Eventually Strong FD with Unknown Membership*** *($\Diamond S^M$) Let $t, t' \in \mathcal{T}$. Let p_i, p_j be mobile nodes. Let $susp_j$ be the list of processes that p_j currently suspects of being faulty. The $\Diamond S^M$ class contains all the failure detectors that satisfy:*

Strong completeness $\overset{def}{=} \{\exists t, t', s.t.\ \forall t' \geq t,\ \forall p_i \in \text{KNOWN} \cap \text{FAULTY} \Rightarrow p_i \in susp_j,\ \forall p_j \in \text{KNOWN} \cap \text{STABLE}\}$.

Eventual weak accuracy $\overset{def}{=} \{\exists t, t', s.t.\ \forall t' \geq t,\ \exists p_i \in \text{KNOWN} \cap \text{STABLE} \Rightarrow p_i \notin susp_j,\ \forall p_j \in \text{KNOWN} \cap \text{STABLE}\}$.

3 Towards a Time-Free Failure Detector for the $\Diamond S^M$ Class

None of the failure detector classes can be implemented in a purely asynchronous system [2]. Indeed, while completeness can be realized by using "I am alive" messages and timeouts, accuracy cannot be safely implemented for all system executions. Thus, some additional assumptions on the underlying system should be done in order to implement them. With this aim, two orthogonal approaches can be distinguished in the literature: the timer-based and the time-free failure detection [25]. The timer-based model is the traditional approach and supposes that channels in the system are eventually timely; this means that, for every execution, there are bounds on process speeds and on message transmission delays. However, these bounds are not known and they hold only after some unknown time [2]. An alternative approach suggested by [8] and developed so far by [14, 23] considers that the system satisfies a message exchange pattern on the execution of a *query-based* communication and is time-free. While the timer-based approach imposes a constraint on the physical time (to satisfy message transfer delays), the time-free approach imposes a constraint on the logical time (to satisfy a message delivery order). These approaches are orthogonal and cannot be compared, but, they can be combined at the link level in order to implement hybrid protocols with combined assumptions [25].

3.1 Stable Query-Response Communication Mechanism

Our failure detector is time-free and based on a local QUERY-RESPONSE communication mechanism [23] adapted to a network with unknown membership. At each *query-response* round, a node systematically broadcasts a QUERY message to the nodes in its neighborhood until it possibly crashes or leaves the system. The time between two consecutive queries is finite but arbitrary. Each couple of QUERY-RESPONSE messages are uniquely identified in the system. A process p_i launches the primitive by sending a QUERY(m) with a message m. When a process p_j delivers this query, it updates its local state and systematically answers by sending back a RESPONSE() to p_i. Then, when p_i has received at least α_i responses from different processes, including a stable one, the current QUERY-RESPONSE *terminates*. Without loss of generality, the response for p_i itself is among the α_i responses. An implementation of a QUERY-RESPONSE communication over fair-lossy local channels can be done by the repeated broadcast of the query by the sender p_i until it has received at least α_i responses from its neighbors. The value associated to α_i directly relates to the neighborhood density of process p_i and a discussion about its assignment is done in Section 4.2 after the protocol's explanation.

Formally, the stable QUERY–RESPONSE primitive has the following properties:
(i) QR-Validity: If a QUERY(m) is delivered by process p_j, it has been sent by process p_i;

(ii) QR-Uniformity: A QUERY(m) is delivered at most once by a process;
(iii) QR-Stable-Termination: If a process p_i is not faulty (it does not crash nor leave the system) while it is issuing a query, that query generates at least α_i responses, moreover, at least one of the responses is from a stable known node p_j, $p_j \neq p_i$.

For the failure detection problem, the *stable termination* is important for the diffusion of the information to the whole network and consequent satisfaction of the accuracy and completeness properties. Moreover, it ensures that the first QUERY issued by p_i, when it joins the network, will be delivered by at least one stable process in such a way that p_i may take part to the membership of the system.

3.2 Behavioral Properties

Node p_i can keep continuously moving and pausing, but, infinitively often, p_i should stay within its neighborhood for a sufficient period of time in order to be able to update its state with recent information regarding suspicions and mistakes; otherwise, it would not update its state properly and thus completeness and accuracy properties of the failure detector would not be ensured. Recent information is gathered by p_i from its neighbors via the delivery of a QUERY message. Hence, the following *mobility property*, namely $\mathcal{M}obi\mathcal{P}$, has been defined and should be satisfied by all nodes. It ensures that, after reaching a new neighborhood at t', there will be a time $t > t'$ at which process p_i should have received QUERY messages from at least one stable neighbor p_j, beyond itself. Since channels are fair-lossy, the QUERY sent by p_j will be received by p_i, except if p_i is faulty.

Property 1. Mobility Property ($\mathcal{M}obi\mathcal{P}$). *Let $t', t \in \mathcal{T}, t' < t$. Let p_i be a node. Let t' be the time after which p_i has changed of neighborhood. Let SQ_i^t be the set of processes from which p_i has received a* QUERY *message after t' and before or at t. Process p_i satisfies $\mathcal{M}obi\mathcal{P}$ at time t if:*

$$\mathcal{M}obi\mathcal{P}^t(p_i) \overset{def}{=} \exists p_{j,j\neq i} \in SQ_i^t, t > t' : p_j \in \text{KNOWN} \cap \text{STABLE} \ \vee p_i \textit{ is faulty after } t'.$$

Instead of synchrony assumptions, to ensure the accuracy of the detection, the time-free model establishes conditions on the logical time the messages are delivered by processes. These are unified in the *stabilized responsiveness property*, namely $\mathcal{SRP}$. Thus, $\mathcal{SRP}(p_i)$ states that eventually, for any process p_j (who had received a response from $p_i$in the past), the set of responses received by p_j to its last QUERY always includes a response from p_i, that is, the response of p_i is always a winning response [25].

Property 2. Stabilized Responsiveness Property ($\mathcal{SRP}$). *Let $t'', t', t \in \mathcal{T}$. Let p_i be a stable known node. Let $rec\_from_j^{t'}$ ($rec\_from_j^{t''}$) be the set of processes from which p_j has received responses to its last* QUERY *that terminated at or before t' (t''). Process p_i satisfies $\mathcal{SRP}$ at time t if:*

$$\mathcal{SRP}^t(p_i) \overset{def}{=} \forall t' \geq t, \forall t'' > t', p_i \in rec\_from_j^{t'} \Rightarrow p_i \in rec\_from_j^{t''} \vee p_j \textit{ is faulty after } t.$$

This property denotes the ability of a stable known node p_i to reply, among the first α_i nodes, to a QUERY sent by a node p_j, who had received responses from p_i before. It should hold for at least one stable known node p_i in the system; thus preventing p_i to be permanently suspected. As a matter of comparison, in the timer-based model, this property would approximate the following: there is a time t after which the output channels from a stable process p_i to every other process p_j that knows p_i are eventually timely.

Assumption 2 summarizes the behaviors that nodes should satisfy in order to implement a failure detector of class $\diamond S^M$ in a mobile unknown network.

Assumption 2. ***Behavioral Assumptions.***
(1) $\forall p_i \in$ KNOWN : $\mathcal{M}obi\mathcal{P}^t(p_i)$ *holds after* p_i *moves and changes of neighborhood;*
(2) $\exists p_i \in$ KNOWN $\cap$ STABLE : $\mathcal{SRP}^t(p_i)$ *eventually holds.*

A discussion about how these assumptions could be satisfied in practice is done in Section 4.2 after the protocol's explanation.

4 A Failure Detector Algorithm for the $\diamond S^M$ Class

4.1 Algorithm Description

Algorithm 1 describes our protocol for implementing a FD of class $\diamond S^M$ for a mobile network of unknown membership that satisfies the model and assumptions stated in Sections 2 and 3. Particularly, we consider a network composed of KNOWN mobile nodes (Definition 1) in which Assumptions 1 and 2 hold.

Notations. We use the following notations:

- $susp_i$: denotes the current set of processes suspected of being faulty by p_i. Each element of this set is a tuple of the form $\langle id, ct \rangle$, where id is the identifier of the suspected node and ct is the tag associated to this information.
- $mist_i$: denotes the set of nodes which were previously suspected of being faulty but such suspicions are currently considered to be a mistake. Similar to the $susp_i$ set, the $mist_i$ is composed of tuples of the form $\langle id, ct \rangle$.
- rec_from_i: denotes the set of nodes from which p_i has received responses to its last QUERY message.
- $known_i$: denotes the partial knowledge of p_i about the system's membership, i.e., it denotes the current knowledge of p_i about its neighborhood.
- $Add(set, \langle id, ct \rangle)$: is a function that includes $\langle id, ct \rangle$ in set. If an $\langle id, - \rangle$ already exists in set, it is replaced by $\langle id, ct \rangle$.

Description. The algorithm is composed of two tasks $T1$ and $T2$.

Task $T1$*: Generating suspicions.* This task is made up of an infinite loop. At each round, a QUERY($susp_i$, $mist_i$) message is sent to all nodes of p_i's neighborhood (line 5). Node p_i

Algorithm 1 Time-Free Implementation of a $\Diamond S^M$ Failure Detector

```
1   init:
2    $susp_i \leftarrow \varnothing; mist_i \leftarrow \varnothing$ ; $known_i \leftarrow \varnothing$
3   Task T1:
4   Repeat forever
5       broadcast QUERY($susp_i$, $mist_i$)
6       wait until RESPONSE received from at least $\alpha_i$ processes
7       $rec\_from_i \leftarrow$ all $p_j$, a RESPONSE is received in line 6
8       For all $p_j \in known_i \setminus rec\_from_i \mid \langle p_j, - \rangle \notin susp_i$ do
9           If  $\langle p_j, ct \rangle \in mist_i$
10              $Add(susp_i, \langle p_j, ct+1 \rangle)$
11              $mist_i = mist_i \setminus \{\langle p_j, - \rangle\}$
12          Else
13              $Add(susp_i, \langle p_j, 0 \rangle)$
14  End repeat
15
16  Task T2:
17  Upon reception of QUERY ($susp_j$,$mist_j$) from $p_j$ do
18  $known_i \leftarrow known_i \cup \{p_j\}$
19  For  all $\langle p_x, ct_x \rangle \in susp_j$ do
20  If $\langle p_x, - \rangle \notin susp_i \cup mist_i$ or ($\langle p_x, ct \rangle \in susp_i \cup mist_i$ and $ct < ct_x$)
21      If $p_x = p_i$
22          $Add(mist_i, \langle p_i, ct_x + 1 \rangle)$
23      Else
24          $Add(susp_i, \langle p_x, ct_x \rangle)$
25          $mist_i = mist_i \setminus \{\langle p_x, - \rangle\}$
26  For  all $\langle p_x, ct_x \rangle \in mist_j$ do
27  If $\langle p_x, - \rangle \notin susp_i \cup mist_i$ or ($\langle p_x, ct \rangle \in susp_i \cup mist_i$ and $ct < ct_x$)
28      $Add(mist_i, \langle p_x, ct_x \rangle)$
29      $susp_i = susp_i \setminus \{\langle p_x, - \rangle\}$
30      If ($p_x \neq p_j$)
31          $known_i \leftarrow known_i \setminus \{p_x\}$
32  send RESPONSE to $p_j$
```

waits for at least α_i responses, which includes p_i's own response (line 6). Then, p_i detects new suspicions (lines 8-13). It starts suspecting each node p_j, not previously suspected ($p_j \notin susp_i$), which it knows ($p_j \in known_i$), but from which it does not receive a RESPONSE to its last QUERY. If a previous mistake information related to this new suspected node exists in the mistake set $mist_i$, it is removed from it (line 11) and the suspicion information is then included in $susp_i$ with a tag which is greater than the previous mistake tag (line 10). If p_j is not in the *mist* set (i.e., it is the first time p_j is suspected), p_i suspected information is tagged with 0 (line 13).

Task $T2$: Propagating suspicions and mistakes. This task allows a node to handle the reception of a QUERY message. A QUERY message contains the information about suspected nodes and mistakes kept by the sending node. However, based on the tag associated to each piece of information, the receiving node only takes into account the ones that are more recent than those it already knows or the ones that it does not know at all. The two loops of task $T2$ respectively handle the information received about suspected nodes (lines 19–25) and about mistaken nodes (lines 26–31). Thus, for each node p_x included in the suspected (respectively, mistake) set of the QUERY message, p_i includes the node p_x in its $susp_i$ (respectively, $mist_i$) set only if the following condition is satisfied: p_i received a more recent information about p_x status (failed or mistaken) than the one it has in its $susp_i$ and $mist_i$ sets. Furthermore, in the first loop of task $T2$, a new mistake is detected if the receiving node p_i is included in the suspected set of the QUERY message (line 21) with a greater tag. At the end of the task (line 32), p_i sends to the querying node a RESPONSE message.

Dealing with mobility and generating mistakes. When a node p_m moves to another destination, the nodes of its old destination will start suspecting it, since p_m is in their *known* set and it cannot reply to QUERY messages from the latter anymore. Hence, QUERY messages that include p_m as a suspected node will be propagated to nodes of the network. Eventually, when p_m reaches its new neighborhood, it will receive such suspicion messages. Upon receiving them, p_m will correct such a mistake by including itself (p_m) in the mistake set of its corresponding QUERY messages with a greater tag (lines 21-22). Such information will be propagated over the network. On the other hand, p_m will start suspecting the nodes of its old neighborhood since they are in its $known_m$ set. It then will broadcast this suspected information in its next QUERY message. Eventually, this information will be corrected by the nodes of its old neighborhood, and the corresponding generated mistakes will spread over the network, following the same principle.

In order to avoid a “ping-pong” effect between information about failure suspicions and corrections (mistakes), lines 30–31 allow the updating of the *known* sets of both the node p_m and of those nodes that belong to the original destination of p_m. Then, for each mistake $\langle p_x, ct_x \rangle$ received from a node p_j, such that node p_i keeps an old information about p_x, p_i verifies whether p_x is the sending node p_j (line 30). If they are different, p_x should belong to a remote neighborhood, because otherwise, p_i would have received the mistake by p_x itself. Notice that only the process can generate a new mistake about itself (line 21). Thus,

p_x is removed from the local set $known_i$ (line 31). Notice, however, that this condition is not sufficient to detect the mobility, because, p_x can be a neighbor of p_i and due to an asynchronous race, the QUERY sent by p_x with the mistake has not yet arrived at p_i. In fact, the propagated mistake sent by p_j has arrived at p_i firstly. If that is the case, p_x has been unduly removed from $known_i$. Fortunately, since local broadcast is fair-lossy, the QUERY from p_x is going to eventually arrive at p_i, if p_i is stable, and, as soon as the QUERY arrives, p_i will once again add p_x to its $know_i$ set (lines 17–18).

4.2 Practical Issues

The *stable termination* of the QUERY-RESPONSE primitive and the $\mathcal{M}obi\mathcal{P}$ property may be satisfied if the time of pause, between changes in direction and/or speed, is defined to be greater than the time to transmit the QUERY and receive the RESPONSE messages. This condition is attained when for example, the most widely used Random Waypoint Mobility Model [26] is considered. The value associated to α_i (the number of responses that a process p_i should wait in order to implement the QUERY-RESPONSE) should correspond to the expected number of processes with whom p_i can communicate, in spite of moves and faults. Since communication is local, α_i is a local parameter and can be defined as the value of the neighborhood density of p_i (i.e., $|N_i|$) minus the maximum number of faulty processes in its neighborhood; let f_i be this number; that is, $\alpha_i = |N_i| - f_i$. This local choice for α_i changes from previous works which consider a global value either proportional to the number of correct processes [8] or the number of stable processes [23] or the global number of faults [14]. Moreover, it follows recent works on fault tolerant communication in radio networks which propose a "local" fault model, instead of a "global" fault model, as an adequate strategy to deal with the dynamism and unreliability of wireless channels in spite of failures [20]. To reliably delivery data in spite of crashes, the maximum number of local failures should be $f_i < |N_i|/2$ [27]. In practice, the value of α_i relates not only with the application density and the expected number of local faults, but also with the type of network considered (either WMN, WSN, etc.) and the current topology of the network during execution. Thus, it can be defined on the fly, based on the current behavior of the network.

Wireless Mesh Network (WMN), Wireless Sensor Network (WSN), and infra-structured mobile networks [14, 28] are a good examples of platforms who would satisfy the assumptions of our model, specially the $\mathcal{SRP}$. In a WMN, the nodes move around a fixed set of nodes (the core of the network) and each mobile node eventually connects to a fix node. A WSN is composed of stationary nodes and can be organized in clusters, so that communication overhead can be reduced; one node in each cluster is designated the cluster head (CH) and the other nodes, cluster members (CMs). Communication inter-clusters is always routed through the respective CHs which act as gateway nodes and are responsible for maintaining the connectivity among neighboring CHs. An infra-structured mobile network is composed of mobile hosts (MH) and mobile support stations (MSS). A MH is connected to a MSS if it is located in its transmission range and two MHs can only communicate through MSSs, but, due to mobility, an MH can leave and enter the area covered by other MSSs. The system is composed of N MSSs but infinitely many MHs. However, in each run the protocol has only finitely many MHs. There are some works to implement a leader oracle [14] and to solve consensus in this type of network [28].

For all these platforms, special nodes (the fixed node for WMN, CHs for WSN or MSSs for infra-structured networks) eventually form a strongly connected component of stable nodes; additionally, they can be regarded as fast, so that they will always answer to a QUERY faster than the other nodes, considered as slow nodes (the mobile node for WMN, CMs for WSN or MHs for infra-structured networks). Thus, one of these fast nodes may satisfy the $\mathcal{SRP}$ property. The $\mathcal{SRP}$ may seem strong, but in practice it should just hold during the time the application needs the strong completeness and eventual weak accuracy properties of FDs of class $\diamondsuit S^M$, as for instance, the time to execute a consensus algorithm.

5 Correctness Proof

We present a proof that Algorithm 1 satisfies both the strong completeness and eventual weak accuracy properties, characterizing a $\diamondsuit S^M$ FD. We consider a mobile network of unknown membership that satisfies the model and assumptions stated in Sections 2 and 3. Particularly, we consider a network composed of KNOWN nodes (Definition 1) in which Assumptions 1 and 2 hold. Let us first make the following remarks.

Observation 1. The *last status* about a process p_x concerning failures is stored in a $susp_i$ or $mist_i$ set of some process p_i and is represented by the tuple $\langle p_x, ct_x \rangle$ which has the greatest counter ct_x in the network at some point t in time.
Observation 2. If process $p_j \in susp_i$ then $p_j \notin mist_i$ and similarly if $p_j \in mist_i$ then $p_j \notin susp_i$. This follows directly by the fact that when p_i adds p_j to $susp_i$, p_j is removed from $mist_i$ (lines 10–11 and 24–25) and vice-versa (lines 28–29). The only exceptions occur in lines 13 and 22, but in both cases, due to predicates of lines 9 and 21, respectively, the observation is ensured.
Observation 3. Only p_i can generate a new mistake about itself (lines 21–23). Moreover, the counter associated with the mistake is strictly increasing (this comes from predicate of lines 20 and 22). Finally, p_i never removes its own mistake $\langle p_i, - \rangle$ from $mist_i$ (lines 8, 11, 25).
Observation 4. Process $p_i \notin susp_i$ is always true. This follows from predicate of line 8, which is never satisfied, and predicate of line 21, which is always satisfied for $\langle p_i, - \rangle$; thus p_i is never included in $susp_i$ in lines 10 and 24.

Proof for the Specific Case of Pausing Nodes.

Let us notice that a mobile node is either *moving* (to reach a neighborhood, possibly different form its present one) or *pausing* (in this case, its neighborhood does not change). For the sake of comprehension, we first prove the FD properties hold for a network composed only of *pausing* nodes. Afterwards, we prove they hold for the generic case.

Lemma 1. *Let p_i, p_j be stable known nodes. Consider that, at time t, p_i owns the last status about p_x in the network ($\langle p_x, ct\rangle$) in its $susp_i$ set (respectively, $mist_i$ set). If no new information $\langle p_x, ct'\rangle$, $ct' > ct$, is generated after time t, then eventually every stable known node p_j will include $\langle p_x, ct\rangle$ in its $susp_j$ set (respectively, $mist_j$ set).*

Proof. By Assumption 1, there is a path P in the time consisting of stable known nodes between p_i and p_j, $P : p_i = p_0, p_1, \ldots, p_{k-1}, p_k = p_j$. Let us proof the claim by induction on k. The basis, $k = 0$, is true by assumption. By the induction step, the claim is valid for process p_{k-1}. So, p_{k-1} includes $\langle p_x, ct\rangle$ in $susp_{k-1}$ (respectively, $mist_{k-1}$). Let us show that p_k also includes $\langle p_x, ct\rangle$ in $susp_k$ (respectively, $mist_k$). Since p_{k-1} is stable known, it will execute line 5 and broadcast a QUERY(m) after time t to its neighbors, m contains $\langle p_x, ct\rangle$ in the $susp_{k-1}$ (respectively, $mist_{k-1}$) set. Since *QR-Stable-Termination* is satisfied, this QUERY message terminates and is received by at least a stable known process p_k in the neighborhood of p_{k-1} (line 17). Thus, p_k will execute lines 19-25 (respectively, lines 26-31). Since, by assumption, ct is the greatest counter associated with p_x in the network, p_k executes line 24 (respectively, line 28) and adds $\langle p_x, ct\rangle$ to its own $susp_k$ set (respectively, $mist_k$ set). Thus, the claim is valid for every stable known p_j and the lemma follows. □

Lemma 2. *Infinitely often, for every process p_i there is a stable known process p_j, such that $p_i \in known_j$.*

Proof. Since *QR-Stable-Termination* is satisfied, there is at least one stable known process p_j in the neighborhood of p_i which receives the last QUERY message sent from p_i; thus, $p_i \in known_j$ (lines 17–18). However, p_i can be removed from $known_j$ in line 31 during the treatment of a last mistake raised by p_i. From Observation 3, only p_i can generate a mistake about itself. Moreover, line 31 is the only point in which p_i can be removed from $known_j$. Thus, regarding this removal, two situations are possible:

Situation (1). Assume the last mistake raised by p_i arrives at p_j in a QUERY sent by p_i, that is $\langle p_i, ct\rangle \in mist_i$. In this case, the predicate of line 27 is satisfied, and lines 28–29 are executed, but not lines 30–31. Afterwards, if a QUERY from a process p_k arrives at p_j containing the same mistake over p_i, and such that $p_k \neq p_i$, then, since this mistake has already been taken into account and its counter is not greater than the existing one, the predicate of line 27 will no more be satisfied and lines 30–31 are not executed. Thus p_j will not remove p_i from the $known_j$ set, $p_i \in known_j$.

Situation (2). Assume that, due to an asynchronism, the last mistake raised by p_i arrives at p_j in a QUERY sent by $p_k \neq p_i$, that is $\langle p_i, ct\rangle \in mist_k$. Since this query from p_k arrives at p_j before the own QUERY from p_i, the predicate of line 27 is satisfied and lines 30–31 are executed. Thus p_j removes p_i from $known_j$. Nonetheless, later on, the original QUERY sent by p_i in which $p_i \in mist_i$ arrives to p_j at line 17. This holds because, by assumption, p_j is a stable known neighbor which receives the last QUERY message from p_i. In this case, process p_j will execute line 18 including p_i in $known_j$. Moreover, since this mistake has already been taken into account and its counter is not greater than the existing one, the predicate of line 27 will no more be satisfied and lines 30–31 are no more executed. Thus, p_j will keep p_i in its $known_j$ set, $p_i \in known_j$. This concludes the proof. □

Lemma 3. *Let p_f be a faulty known process and p_i a stable known process. Eventually, p_f is permanently included in $susp_i$ of every p_i.*

Proof. Let us consider that p_f is faulty at time t.

Remark 1. Since p_f is a known process and *QR-Stable-Termination* is satisfied, p_f has sent at least one QUERY message before it crashed at t, and there is at least a stable known process p_i which has received this last QUERY (lines 17–18). From Lemma 2, $p_f \in known_i$. After the crash of p_f at t, p_i will never receive a RESPONSE message from p_f in line 6, thus $p_f \notin rec\_from_i$ and $p_f \in known_i$. In this case, if p_f was not already suspected by p_i (line 8), it will add $\langle p_f, ct' \rangle$ to its $susp_i$ set. From Observation 3, no new information regarding a mistake over p_f is generated after t; moreover, since p_i receives the last QUERY from p_f, it gathers the last mistake over p_f with the greatest counter, if it exists. Thus, if $\langle p_f, ct \rangle \in mist_i$, p_i executes lines 9–11 and adds $\langle p_f, ct+1 \rangle$ in $susp_i$. Otherwise, p_i executes line 13 and adds $\langle p_f, 0 \rangle$ in $susp_i$. Finally, a process p_j in p_f's neighborhood can generate a new suspicion over p_f but only when $p_f \notin susp_j$ (line 8).

Remark 2. Thus, in the network, after t, the last status about p_f can only be a suspicion. In this case, following Lemma 1 and *Observation 2*, all stable known processes will eventually include p_f in their respective suspected sets. Thus, $p_f \in susp_i$ is always true for every stable known p_i. □

Lemma 4. *Let p_i, p_j be stable known processes. Assume that the property $\mathcal{SRP}^t(p_i)$ holds for p_i at time t. Eventually, p_i is never included in $susp_j$ of every p_j.*

Proof. Remark 1. According to $\mathcal{SRP}^t(p_i)$, there is a time t after which every process p_k, who had communicated with p_i, receives a RESPONSE message from p_i in reply to its QUERY. Thus, after time t, $p_i \in rec\_from_k$ is always true. Moreover, since a process is suspected only if its reply is not received (line 8), after time t, p_k never adds p_i in $susp_k$. From Observation 4, $p_i \notin susp_i$ always holds. From Observation 3, a mistake regarding p_i is only generated by p_i itself whether it is in a *suspected* set.

Remark 2. Thus, in the network, after time t, the last status about p_i can be (1) an old suspicion, generated before t; (2) a mistake or (3) none of the previous cases. In Case (1), following the propagation Lemma 1, p_i will eventually receive a QUERY message from a stable known process p_j with $\langle p_i, ct \rangle \in susp_j$. This will cause p_i to generate a new mistake with a greater tag ($\langle p_i, ct+1 \rangle \in mist_i$) (lines 21-23). The last status about p_i is now a mistake and we fall in Case (2). In Case (2), following Lemma 1 and *Observation 2*, the mistake will be propagated to all stable known processes. Then, eventually, lines 28-29 are executed by p_j and $p_i \notin susp_j$. From Remark 1, no new information about p_i is generated and $p_i \notin susp_j$ is always true. Case (3) follows directly from Remark 1 and the lemma follows. □

Theorem 1. *Algorithm 1 implements a failure detector of class $\diamond S^M$, assuming a network of* KNOWN pausing *nodes that satisfies Assumptions 1 and 2.*

Proof. To satisfy the strong completeness property, we must prove that eventually a faulty known node p_f is permanently included in $susp_i$ set of every stable known node p_i. This claim follows directly from Lemma 3. To satisfy the eventual weak accuracy property, we must prove that there is a time after which a stable known node p_i is not included in the $susp_j$ set of any stable known node p_j. This claim follows directly from Lemma 4 and the theorem follows. □

Proof for the Generic Case of Mobile Nodes.

Now, let us extend our proof to the generic case of a network composed of mobile nodes, knowing that a mobile node is either *moving* or *pausing*.

Lemma 5. *Lemma 1 holds for every stable known mobile node.*

Proof. The lemma follows directly from Lemma 1 for all pausing nodes. To take into account moving nodes, we should consider two cases.

Case (1): Assume that p_m is a stable known moving node which has not yet the last status $\langle p_x, ct_x \rangle$ about process p_x. Let us assume that, due to Lemma 1, every pausing stable known node p_i has added the last status $\langle p_x, ct_x \rangle$ in its $susp_i$ (respectively, $mist_i$) set before or at time t''. As soon as p_m reaches the new neighborhood at time $t' \geq t''$, since $\mathcal{M}obi\mathcal{P}^t(p_m)$ is satisfied at $t \geq t'$, p_m will receive a QUERY message from a stable known process p_i, carrying the last status about p_x. Thus, on the execution of task T2 (lines 17-32), since, by assumption, ct_x is the greatest counter associated with p_x, p_m executes line 24 (respectively, line 28) and adds $\langle p_x, ct_x \rangle$ to its own $susp_m$ set (respectively, $mist_m$ set), and the lemma follows.

Case (2): Assume that p_m is a stable known moving node which has the last status $\langle p_x, ct_x \rangle$ about process p_x. As soon as p_m reaches a new neighborhood at time t', it will execute line 5 and broadcast a QUERY message to all its neighbors. Since *QR-Stable-Termination* is satisfied at $t > t'$, there will be at least one stable known node p_i in the neighborhood of p_m, which receives the QUERY with the last status about p_x. Thus, by executing task T2, p_i adds $\langle p_x, ct_x \rangle$ in its $susp_i$ (line 24) (respectively, $mist_i$ set, line 28). Thus, following Lemma 1 and knowing that Case (1) holds, eventually all stable known nodes will include $\langle p_x, ct_x \rangle$ in their suspected set (respectively, mistake set) and the lemma follows. □

Lemma 6. *Lemma 2 holds for every stable known mobile node.*

Proof. Since *QR-Stable-Termination* is satisfied, there is one stable known process p_j in the neighborhood of p_i which receives its last QUERY, no matter if p_i or p_j are pausing or moving. Then, following the same arguments of Lemma 2, $p_i \in known_j$ and the lemma holds. □

Lemma 7. *Let p_f be a faulty known process and p_i a stable known process. Eventually, p_f is permanently included in $susp_i$ of every p_i.*

Proof. Let us consider that p_f is faulty at time t. The lemma follows directly from Lemma 3 for all pausing nodes p_i. To take into account moving nodes, two cases are possible. Let us first observe that, from Lemma 6, since *QR-Stable-Termination* holds at $s \leq t$, there is a stable known process p_i (either moving or pausing), $p_f \in known_i$.

Case (1): Assume that p_m is a stable known moving node which has not yet the last status about process p_f after t; and, assume that by Lemma 3, every stable known pausing node p_j has added p_f in its $susp_j$ set after t. In this case, due to Lemma 5 (Case 1), p_m will add p_f in its $susp_m$.

Case (2): Assume that p_m is a stable known moving node which has the last status about process p_f. Possibly, $p_m = p_i$. We should consider the following two situations. *Situation (1)*: Assume that the last status is a mistake, $\langle p_f, ct\rangle \in mist_m$. When p_m reaches its new neighborhood after t, from the same arguments of Lemma 3 (Remark 1), p_i will add $\langle p_f, ct+1\rangle \in susp_m$ and remove p_f from $mist_m$. Then, we fall in Situation (2). *Situation (2)*: Assume that the last status is a suspicion, $\langle p_f, -\rangle \in susp_m$. This follows from Lemma 3 (Remark 1). When p_m reaches its new neighborhood after t, due to the propagation Lemma 5 (Case 2), this information about the suspicion of p_f will be propagated to all stable known nodes in the network.

Finally, for every Case, from the same arguments of Lemma 3 (Remark 2) and considering Lemma 5, p_f is permanently included in every $susp_i$ of a stable known node p_i. □

Lemma 8. *Let p_i, p_j be stable known nodes. Assume that $\mathcal{SRP}^t(p_i)$ holds for p_i at time t. Eventually, p_i is never included in $susp_j$ of every p_j.*

Proof. The lemma follows directly from Lemma 4 for all pausing nodes p_j. Since $\mathcal{SRP}^t(p_i)$ is satisfied, there is a time t after which every process p_k, who had communicated with p_i, receives a RESPONSE message from p_i in reply to its QUERY. Thus, after time t, $p_i \in rec\_from_k$ is always true, and due to same arguments of Lemma 4 (Remark 1), after t, these nodes cannot generate a new suspicion over p_i in the network. Assume that p_m is a stable known moving node. Let us consider that p_m reaches its new neighborhood at time $t' \geq t$. By hypothesis, if $p_i \in rec\_from^{t'}_m$ then $p_i \in rec\_from^{t''}_m, t'' \geq t'$ is always true, thus p_m never adds p_i in $susp_m$. If $p_i \notin rec\_from^{t'}_m$, two cases are possible:

Case (1): p_m has the last status about p_i. The following situations are possible. *Situation (1)*: p_m suspects p_i. This can be an old suspicion, generated before t or a new one, generated after t due to p_m's move. In this case, $p_i \in known_m$, but p_m will no longer receive response messages from p_i, since p_m moves. Thus, p_m will suspect p_i (executing lines 8–13). According to Lemma 4 (Remark 2, Case 1), this suspicion is going to be revoked by p_i, by the generation of a mistake message with a greatest counter that, due to Lemma 5, is propagated along the network. Now, $p_i \in mist_m$ and $p_i \notin susp_m$. So, the last status of p_m is a mistake and we fall in Situation 2. *Situation (2)*: p_m does not suspect p_i. Following Lemma 4 (Remark 2, Cases 2 and 3) and Lemma 5, every stable known process p_j (including p_m) will permanently remove p_i from its respective $susp_j$ set.

Case (2): p_m has not yet the last status about process p_i. Due to Lemma 5, after time t', p_m will update its state with the last information about p_i. Due to Lemma 4 and 5, eventually $p_i \notin susp_m$ is always true. Thus, the lemma follows for every stable known node p_j.

□

Theorem 2. *Algorithm 1 implements a failure detector of class $\Diamond S^M$, assuming a network of* Known *mobile nodes that satisfies Assumptions 1 and 2.*

Proof. The strong completeness property follows directly from Lemma 7. The eventual weak accuracy property follows directly from Lemma 8 and the theorem follows. □

6 Related Work

As in the approach followed in our work, some scalable FD implementations do not require a fully connected network. Larrea *et al.* proposed in [5] an implementation of an unreliable failure detector based on a logical ring configuration of processes. Thus, the number of messages is linear, but the time for propagating failure information is quite high. Some works base the detection on the use of an adaptive heartbeat or follow the gossiping style communication, choosing only a few members or neighbors to disseminate information [29, 30]. Practically, the randomization makes the definition of timeout values difficult. In [31], a scalable hierarchical failure adapted for Grid configurations is proposed. However, the global configuration of the network is initially known by all nodes. Aguilera *et al.* [9] proposes the heartbeat FD which does not assume a network of fully connectivity and tolerates message losses. Tucci *et al.* [10] implements a heartbeat FD for the infinite arrival model and shows how to use it to implement a FD of the Ω class. The Ω class ensures that eventually each correct process is going to trust the same correct process, considered as the *leader*. The solution considers fair-lossy channels, but for a synchronous environment. Hutle [32] proposes a $\Diamond P$ FD with strong completeness (eventually, every node failure will be reported to every correct node) and eventual strong accuracy (after some point in time no correct node will be suspected by another correct node) properties. The solution considers sparsely connected unknown networks, subject to partitions; nonetheless, it assumes some knowledge about the neighborhood, which has a bounded number of processes, and about the jitter of the communication between direct neighbors. It is worth remarking that none of these previous works tolerate mobility of nodes.

Few implementations of unreliable FD found in the literature focus on wireless mobile networks [11, 12, 13]. The fundamental difference between these works and ours is the fact that all of them are timer-based. As soon as we are aware of the only work to follow a time-free detection strategy has been proposed by [14] in order to implement a *leader* failure detector of the Ω class. Nonetheless, it does not tolerate node mobility.

Friedman and Tcharny [11] propose a simple gossiping protocol which exploits the natural broadcast range of wireless networks to delimit the local membership of a node in a mobile network. A node periodically sends heartbeat messages to its neighbors. When receiving a vector, a node updates its vector to the maximum of its local vector and the former. Thus, if a node does not receive a new heartbeat information about a node after a certain time, it considers that the latter has failed. Contrarily to our approach, this work assumes a known number of nodes and provides probabilistic guarantees for the FD properties.

Tai *et al.* [12] exploit a cluster-based communication architecture to propose a hierarchical gossiping FD protocol for a network of non-mobile nodes. The FD is implemented both via intra-cluster heartbeat diffusion and failure report diffusion across clusters, i.e., if a failure is detected in a local cluster, it will be further forwarded across the clusters. It implements a FD of the class $\Diamond P$ providing probabilistic guarantees for the completeness and accuracy properties. Unlike our solution, this work considers a cluster-based communication architecture and provides probabilistic guarantees for the accuracy and completeness properties; moreover, it does not consider mobility.

Sridhar [13] adopts a hierarchical design to propose a deterministic local FD. He introduces the notion of *local failure detection* and restraints the scope of detection to the neighborhood of a node and not to the whole system. The FD is composed of two independent layers: a local one that builds a suspected list of crashed neighbors and a second one that detects mobility of nodes across network and able to correct possible mistakes. It advocates the use of this local detection as an appropriate abstraction to deal with mobility and resources lack in wireless sensor networks. While our approach allows the implementation of a $\Diamond S^M$ FD, this work implements an eventually perfect *local* failure detector of the class $\Diamond P$, i.e., it provides perfect failure detection, but with regard to a node's neighborhood.

Cao *et al.* [14] propose a time-free query-based implementation of a deterministic *leader* failure detector. It considers an infra-structured mobile network composed of mobile hosts (MH) and mobile support stations (MSS) (see Section 4.2). A MH is considered stable if, once it entered the system, it does not crash or gets disconnected. Both MSSs and MHs can crash and the maximum number of MSSs that can crash (f) is known a priori. Differently from ours, this work consider a hybrid network of mobile and static nodes; moreover, it implements an Ω FD. It provides an eventual accuracy property, which ensures that eventually at least one stable MH is continuously trusted by the MSSs. The completeness property ensures that an MH that crashes or permanently leaves the system is eventually no longer trusted by an MSS.

Table 1 shows a panorama of the FDs for Manets presented in this section considering a number of criteria: (1) type of nodes in the network, (2) knowledge about the number of nodes, (3) number of failures considered, (4) the connectivity of the communication network, (5) failure model considered, (6) strategy followed to detect failures, (7) the use of timers to detect failures, (8) the characterization of a membership for the network, (9) the use of local communication to make detection, (10) the FD class provided. The work presented herein exhibits the most generic features. It implements a time-free query-based deterministic FD suitable for any MANET topology.

FD Application. We believe that our $\Diamond S^M$ FD will be of great interest to implement consensus algorithms, such as the one proposed by Greve *et al.*[16], who present a solution for the fault-tolerant consensus in a dynamic system of unknown participants, with minimal synchrony assumptions (i.e., a FD of the class $\Diamond S$).

Table 1: Failure Detectors Comparison

Protocol	*Node Type*	*Number of Nodes*	*Number of Failures*	*Network Connectivity*	*Failure Model*
Friedman *et al.* [11]	Mobile	Known	Arbitrary	Connected graph	Crash Message omission
Tai *et al.* [12]	Static	Known	Arbitrary	Static connected graph of cluster heads	Crash Message omission
Sridhar [13]	Mobile	Known	Arbitrary	Connected graph	Crash
Cao *et al.* [14]	Static and Mobile	Known (static) Unknown (mobile)	f (fixed)	Complete graph of static nodes	Crash (only mobile) Reliable channels
Our protocol	**Mobile**	**Unknown**	**Arbitrary**	**Connected graph**	**Crash** **Message omission**

Protocol	*Detection Strategy*	*Timer Based*	*Membership Set*	*Local Detection*	*FD Class*
Friedman *et al.* [11]	Heartbeat	Yes	Yes	No	-
Tai *et al.* [12]	Heartbeat	Yes	Yes (cluster head) No (mobile nodes)	Yes	Probabilistic $\Diamond P$
Sridhar [13]	Generic	Yes	Yes	Yes	Local $\Diamond P$
Cao *et al.* [14]	Query-Response	No	Yes (static nodes) No (mobile nodes)	No	Ω Leader
Our protocol	**Query-Response**	**No**	**Yes**	**Yes**	$\Diamond S^M$

7 Performance Evaluation

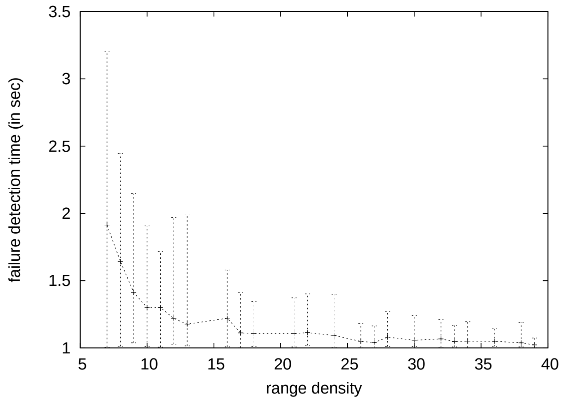
In this section we study and evaluate the behavior of our asynchronous failure detector. The performance experiments were conducted on top of the OMNeT++ discrete event simulator [33]. We assume 2 two-dimensional regions: the first one is a square of $600m$x$600m$ and the second one is a rectangle of $100m$x$1800m$

We consider that $\forall p_i$, the range density d_i is equal to d and the maximum number of failure f_i is equal to f. Therefore, $\forall p_i$, α_i has the same value. The number of nodes N is fixed to 100 uniformly distributed over the region and each simulation lasts 30 minutes. In the rectangle configuration the number of hops of the longest path is higher than in the square. The one-hop network delay δ is equal to $1ms$ in average.

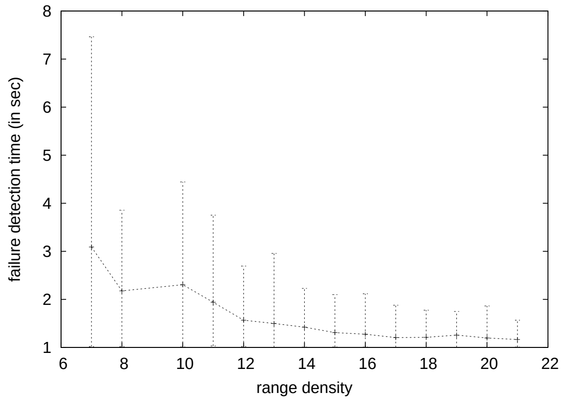
Concerning the implementation of our FD, it is not feasible that a node continuously broadcasts a QUERY message since the network would be overloaded with messages. To overcome this problem, we have included a delay of $\Delta = 1s$ between lines 6 and 8 of the Algorithm 1. However, by adding this waiting period, process p_i may receive more than α_i replies. Therefore, the extra replies will also be included in the rec_from set of this process (line 8), reducing then the number of false suspicions. It is worth remarking that this improvement does not change the protocol correctness.

7.1 Failure Detection

In order to evaluate the completeness property of our FD, we have measured the impact of the range density d on the failure detection time. To this end, the transmission range r varied from $100m$ to $380m$ which results in the variation of the range density d as shown in Figure 1. The maximum number of faults f is equal to 5 and they are uniformly inserted during an experiment at every 110s starting at 10s (10s, 120s, 230s, 340s, and 450s). For each range density, we have measured the average, maximum and minimum failure detection time considering the 95 correct nodes.



(a) 600x600 region



(b) 100x1800 region

Figure 1: Failure detection time vs. range density

We observe that there is no false suspicion. Furthermore, the propagation of failure detections is quite fast because the diameter of the network is relatively small in both configurations.

The failure detection time decreases with the range density. This happens because failure detection information is included in QUERY messages which spreads faster over the network when the density increases. We can notice that for values of d greater than 22, the failure detection time is uniform and equals around $\Delta + \delta$. The maximum failure detection time characterizes, for each range density, the time for all nodes to detect a failure (strong completeness).

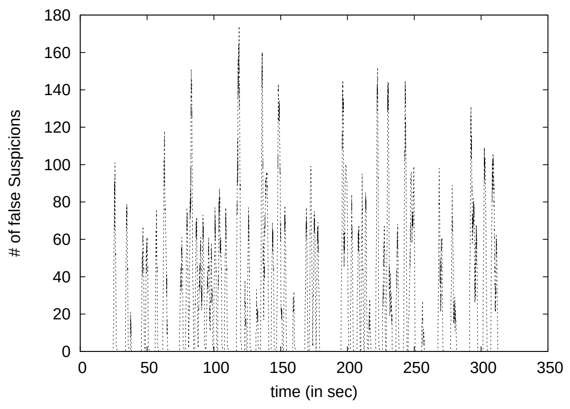
7.2 Impact of mobility

We have evaluated the accuracy property when both one and ten nodes located at one boundary of the network moves at a speed of 2m/s. The range transmission r is set to 100m, the density d equals to 7 and f equals to 5 ($\alpha_i = 2$, $\forall p_i$). When just one node moves, it starts moving at time 20s while when 10 nodes move, the first one starts moving at 100s and at every 5s a new one starts moving. A moving node stops when it arrives at the opposite border of the region. We consider that while moving, a moving node p_m continues to interact with the other nodes. Furthermore, all neighbors p_j of p_m must have at least $f + 1$ neighbors. Such restriction is necessary to guarantee that at least $\alpha_j = d - f$ nodes will reply to the query of p_j after p_m moves.

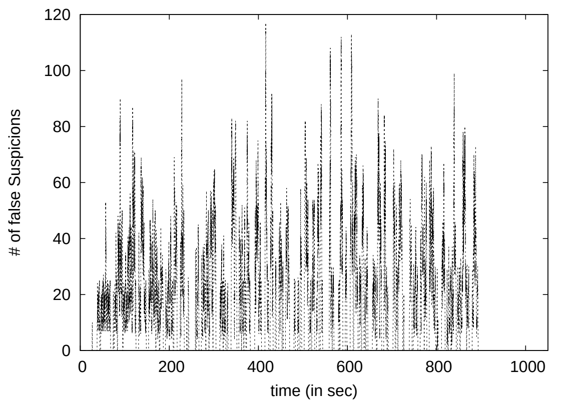
For each experiment, the total number of false suspicions has been measured. Figure 2.(a) (respectively, 3.(b)) shows the number of false suspicions between the moment that just one node (respectively, 10 nodes) stars moving at 20s (respectively, 100s and every 5s) and stops moving at time 315s (respectively, 880s, all nodes) for both the square and rectangle region configurations.

We see in both figures that false suspicions are rather ponctuel: the number of false suspicions increases very fast but decreases very fast too. This happen because *moving* nodes are suspected by nodes of its range and this information is spread very fast over the network since the diameter of the network is relatively small in both configurations. On the other hand, false suspicions of *moving* nodes start being corrected by the latter that generate mistakes which are analogously propagated very fast over the network.

Figures 4.(a) and 4.(b) respectively show the distribution of mistake duration and the number of false suspicions for all the 100 nodes in the rectangle configuration when 10 nodes move. We can observe that the duration of mistakes is quite small and stable for all of them. The average mistake duration is smaller than 1s. However, the number of false suspicions presents a more significant variation. In fact, for a given node, this number depends on its position in the region. On the other hand, the greater the number of mobile nodes that a node meets, the greater the number of false suspicions that it generates. In any case, the mistake is always corrected very fast: in less than 0.01s when the suspected node is close to the node that generates the suspicion and in 4s at maximum, otherwise.

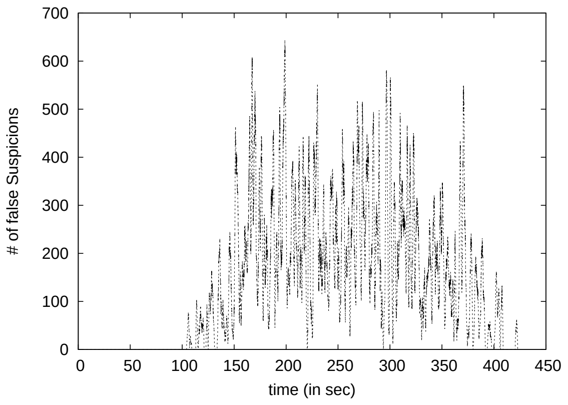


(a) 600x600 region

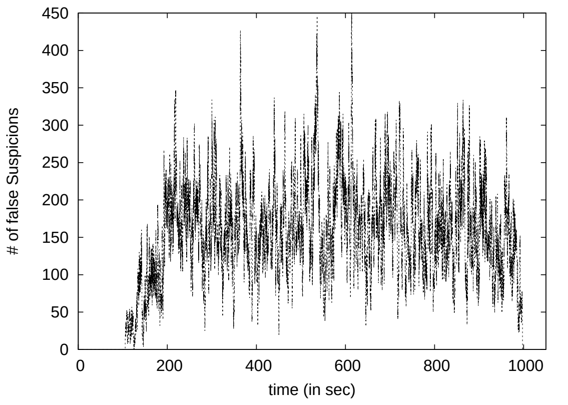


(b) 100x1800 region

Figure 2: Total number of false suspicions when one node moves

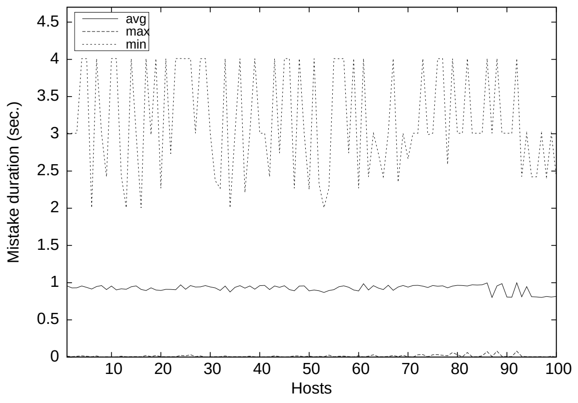


(a) 600x600 region

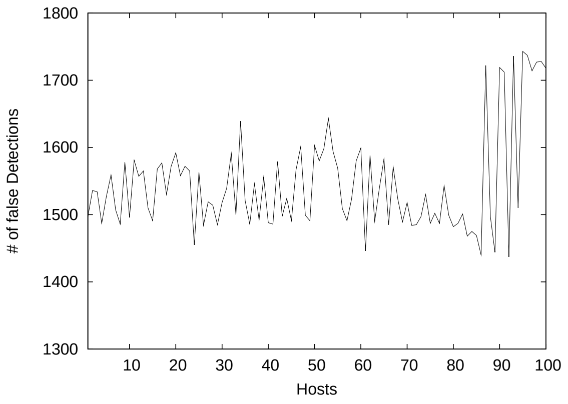


(b) 100x1800 region

Figure 3: Total number of false suspicions when ten nodes move



(a) Distribution of mistakes in 100x1800 region



(b) Distribution of false suspicions in 100x1800 region

Figure 4: Distribution of mistakes when ten nodes move

8 Conclusion

This paper has presented a new algorithm for an unreliable failure detector suitable for mobile wireless networks, such as WMNs or WSNs. It implements failure detectors of class $\diamondsuit S^M$ (eventually strong with unknown membership) when the exchanged pattern of messages satisfies some behavioral properties. As a future work, we plan to adapt the algorithm and properties to implement other classes of failure detectors.

References

[1] Conti, M., Giordano, S.: Multihop ad hoc networking: The theory. Communications Magazine, IEEE **45**(4) (2007) 78–86

[2] Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM **43**(2) (March 1996) 225–267

[3] Akyildiz, I.F., Wang, X., Wang, W.: Wireless mesh networks: a survey. Computer Networks **47**(4) (2005) 445–487

[4] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks **38** (2002) 393–422

[5] Larrea, M., Fernández, A., Arévalo, S.: Optimal implementation of the weakest failure detector for solving consensus. In: Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing. (2000) 334–334

[6] Sotoma, I., Madeira, E.: Adaptation - algorithms to adaptative fault monitoring and their implementation on corba. In: Proc. of the IEEE 3rd Int. Symposium on Distributed Objects and Applications. (2001) 219–228

[7] Devianov, B., Toueg, S.: Failure detector service for dependable computing. In: Proc. of the 1st Int. Conf. on Dependable Systems and Networks. (2000) 14–15

[8] Mostefaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: Proc. of Int. Conf. on Dependable Systems and Networks. (2003)

[9] Aguilera, M.K., Chen, W., Toueg, S.: Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: Proc. of the 11th International Workshop on Distributed Algorithms. (1997) 126–140

[10] Tucci-Piergiovanni, S., Baldoni, R.: Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In: Dependable Computing Conference (EDCC), 2010 European. (2010) 127 –134

[11] Friedman, R., Tcharny, G.: Evaluating failure detection in mobile ad-hoc networks. Int. Journal of Wireless and Mobile Computing **1**(8) (2005)

[12] Tai, A., Tso, K., Sanders, W.: Cluster-based failure detection service for large-scale ad hoc wireless network applications. In: Proc. of the Int. Conf. on Dependable Systems and Networks. (2004) 805–814

[13] Sridhar, N.: Decentralized local failure detection in dynamic distributed systems. The 25th IEEE Symp. on Reliable Distributed Systems **0** (2006) 143–154

[14] Cao, J., Raynal, M., Travers, C., Wu, W.: The eventual leadership in dynamic mobile networking environments. In: Proc. of the 13th Pacific Rim International Symposium on Dependable Computing. (2007) 123–130

[15] Sens, P., Arantes, L., Bouillaguet, M., V.Simon, Greve, F.: An unreliable failure detector for unknown and mobile networks. In: OPODIS. (2008) 555–559

[16] Greve, F., Tixeuil, S.: Knowledge conectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In: Proc. of the Int. Conf. on Dependable Systems and Networks. (2007) 82–91

[17] Aguilera, M.K.: A pleasant stroll through the land of infinitely many creatures. SIGACT News **35**(2) (2004) 36–59

[18] Ramasubramanian, V., Chandra, R., Mossé, D.: Providing a bidirectional abstraction for unidirectional adhoc networks. In: Proc. of the 21st IEEE International Conference on Computer Communications. (2002)

[19] Min-Te, S., Lifei, H., A. Arora, A., L.Ten-Hwang: Reliable mac layer multicast in ieee 802.11 wireless networks. In: Proc. of the International Conference on Parallel Processing. (August 2002) 527–536

[20] Koo, C.Y.: Broadcast in radio networks tolerating byzantine adversarial behavior. In: Proc. of the 23th annual ACM symposium on Principles of distributed computing. (2004) 275–282

[21] Bhandari, V., Vaidya, N.H.: Reliable local broadcast in a wireless network prone to byzantine failures. In: The 4th ACM SIGACT/SIGMOBILE International Workshop on FOUNDATIONS OF MOBILE COMPUTING. (2007)

[22] Jiménez, E., Arévalo, S., Fernández, A.: Implementing unreliable failure detectors with unknown membership. Inf. Process. Lett. **100**(2) (2006) 60–63

[23] Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., Abbadi, A.: From static distributed systems to dynamic systems. In: Proc. of the 24th IEEE Symposium on Reliable Distributed Systems. (2005) 109–118

[24] Bhandari, V., Vaidya, N.H.: Reliable broadcast in radio networks with locally bounded failures. IEEE Transactions on Parallel and Distributed Systems **21** (2010) 801–811

[25] Mostefaoui, A., Raynal, M., Travers, C.: Time-free and timer-based assumptions can be combined to obtain eventual leadership. IEEE Trans. Parallel Distrib. Syst. **17**(7) (2006) 656–666

[26] Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications & Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications **2** (2002) 483–502

[27] Bhandari, V., Vaidya, N.H.: On reliable broadcast in a radio network. In: Proc. of the 24th annual ACM symposium on Principles of distributed computing, ACM (2005) 138–147

[28] Wu, W., Cao, J., Yang, J., Raynal, M.: Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. IEEE Trans. Comput. **56**(8) (2007) 1055–1070

[29] Gupta, I., Chandra, T.D., Goldszmidt, G.S.: On scalable and efficient distributed failure detectors. In: Proc. of the 20th annual ACM symposium on Principles of distributed computing. (2001) 170–179

[30] Van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. Technical report, Ithaca, NY, USA (1998)

[31] Bertier, M., Marin, O., Sens, P.: Performance analysis of a hierarchical failure detector. In: Proc. of the Int. Conf. on Dependable Systems and Networks. (2003)

[32] Hutle, M.: An efficient failure detector for sparsely connected networks. Proc. of the IASTED International Conference on Parallel and Distributed Computing and Networks (2004) 369–374

[33] : OMNet++ Discret Event Simulation System. http://www.omnetpp.org.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399