# Eventually Strong Failure Detector with Unknown Membership

Fabíola Greve[‡1], Pierre Sens[2], Luciana Arantes[2] and Véronique Simon[2]

[1]*Department of Computer Science, Federal University of Bahia (UFBA), Bahia, BA - Brazil*
[2]*LIP6, University of Paris 6, CNRS, INRIA, 4 - Place Jussieu, 75005, Paris, France*
*Email: fabiola@dcc.ufba.br, FirstName.LastName@lip6.fr*

**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 dynamics 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 time-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 $\diamondsuit S^{\mathcal{M}}$, which adapts the properties of the $\diamondsuit S$ 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.**

*Keywords: Unreliable failure detector, dynamic distributed systems, wireless mobile networks, asynchronous systems*

## 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 dynamics 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 $\diamondsuit S$. 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). $\diamondsuit S$ 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 dynamics 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 appropriate for dynamic environments. Furthermore, they are usually time-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 denoted 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 a list 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 time-based. The only exception is [14], but it does not tolerate node mobility.

### 1.1. Contributions

This paper has two contributions: *(i)* the proposition of a model and the definition of the $\diamondsuit S^M$ class of FDs; *(ii)* a new time-free FD algorithm that implements the class $\diamondsuit S^M$ under a wireless mobile network.

In order to implement unreliable FDs in an asynchronous dynamic system of mobile nodes, some assumptions about the underlying system should be made. 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, may 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 properties that nodes should satisfy in the network. For example, in order to be known in the system, a mobile node should interact at least once with some other node that never departs from the system. Thus, the first contribution of this paper is to propose a model and identify sufficient assumptions able to implement the properties of a new class of failure detectors suitable for mobile networks with unknown membership. The class of *eventually strong FD with unknown membership* (namely, $\diamondsuit S^M$) adapts the properties of the $\diamondsuit S$ class to a dynamic system with an unknown membership.

The second contribution is the proposition of a FD algorithm that implements $\diamondsuit S^M$. It is suitable for wireless mobile networks and has the following innovative features that provide 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. 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 QUERY and RESPONSE 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 $\diamondsuit S^M$ when some properties are satisfied by the underlying system. We believe that our FD of class $\diamondsuit S^M$ may be successful adopted to implement coordination protocols in a dynamic set, such as the one proposed by Greve *et al.* [15], who present a solution for the fault-tolerant consensus in a network of unknown participants with minimal synchrony assumptions.

A preliminary work with some of the contributions of this paper appeared firstly in [16] as a brief announcement. Afterwards, in [17], we have provided a first version of the algorithm to implement $\diamondsuit S^M$, but for a slightly different model. Then, in [18], a discussion about an appropriate model to implement unreliable failure detectors in dynamic systems is presented.

The rest of the paper is organized as follows. Section 2 defines the model and specifies the $\diamondsuit S^M$ FD class. Section 3 identifies assumptions to implement those FDs. Sections 4 and 5 present a time-free FD of the $\diamondsuit S^M$ class and its correctness proofs respectively.

Section 6 presents an experimental evaluation of the proposed FD protocol. A thorough related work is described in Section 7. Finally, Section 8 concludes the paper.

## 2. MODEL FOR FAILURE DETECTION IN DYNAMIC NETWORKS

We are particularly interested in systems deployed over a wireless mobile network, such as WMNs, WSNs, and MANETs. The system is a collection of nodes which communicate by sending and receiving messages via a packet radio network.

There are no assumptions on the relative speed of processes or on message transfer delays, thus the system is *asynchronous*. To simplify the presentation, we take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers. There is no global clock and processes do not have access to $\mathcal{T}$: it is introduced for the convenience of the presentation, to state properties and make proofs.

*Finite arrival model* [19]. The network is a dynamic system composed of infinitely many processes; but each run consists of a finite set $\Pi$ of $n$ nodes, namely, $\Pi = \{p_1, \ldots, p_n\}$. This model properly expresses dynamic networks where nodes join and leave the system as they wish. It 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.

*The membership is unknown.* Processes are not aware about $\Pi$ or $n$, because, moreover, these values can vary from run to run [19]. There is one process per node; each process knows its own identity, but it does not necessarily knows the identities of the others. Nonetheless, they can make use of the broadcast facility of the wireless medium to know one another. Thus, we consider that a process knows a subset of $\Pi$, composed of nodes with whom it previously communicated. A process may fail by *crashing*, *i.e.*, by prematurely or by deliberately halting (switched off); a crashed process does not recover. When a process leaves the network it can re-enter with a new identity, then it is considered as a new process. Until it possible crashes, a process behaves according to its specification. A process that does follow its algorithm specification and never crashes is said to be *correct*.

*Communication graph is dynamic.* Due to arbitrary joins, leaves and failures, the network is represented by a communication graph with a dynamic topology, thus the relations between nodes take place over a time span $\mathcal{T} \subseteq \mathbb{N}$. Following [20], we consider that the dynamics of the system is represented by a *time-varying graph*, namely TVG, $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta, \psi)$, where: (1) $V = \Pi$ represents the set of nodes, (2) $E \subseteq V \times V$ represents the set of logical links between nodes, (3) $\rho : E \times \mathcal{T} \to \{0, 1\}$ is an *edge presence* function, indicating whether a given edge $e \in E$ is available at a given time $t \in \mathcal{T}$, such that $\rho(e, t) = 1$ iff $e$ is present at $t$, otherwise $\rho(e, t) = 0$, (4) $\zeta : E \times \mathcal{T} \to \mathbb{N}$ is a *latency* function, indicating the time taken to cross a given edge $e$ if starting at a given date $t$; since the system is asynchronous, there is no bound for this time, thus, we consider that $\zeta$ exists but cannot be estimated, (5) $\psi : V \times \mathcal{T} \to \{0, 1\}$ is a *node presence* function, indicating whether a given process $p_i \in V$ is up at a given time $t \in \mathcal{T}$, such that $\psi(p_i, t) = 1$ iff node $p_i$ is up at $t$, otherwise $\psi(p, t) = 0$.

Let $R_i$ be the wireless transmission range of $p_i$ in the network, 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 $N_i^t$ to be the set of 1-hop *neighbors* from $p_i$ at time $t \in \mathcal{T}$. The neighborhood relationship establishes the edge set, in such a way that $p_j \in N_i^t$ *iff* $(p_i, p_j) \in E_i^t$, such that $\rho((p_i, p_j), t) = 1$. The *degree* of $p_i$ at time $t$ is defined to be $Deg_i^t = |E_i^t|$.

Given a TVG $\mathcal{G}$, the graph $G = (V, E)$ is called the *underlying graph* of $\mathcal{G}$. $G$ should be considered as a sort of *footprint* of $\mathcal{G}$ which flattens the time dimension and indicates only the pair of nodes that have relations at some time in $\mathcal{T}$. Formally, a sequence of couples $\mathcal{J} = \{(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)\}$, such that $\{e_1, e_2, \ldots, e_k\}$ is a walk in $G$, is a *journey* in $\mathcal{G}$ *if and only if* $\rho(e_i, t_i) = 1$ and $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i < k$. If a journey exists from $p_i$ to $p_j$, we say that $p_i$ *reaches* $p_j$ or more simply, $p_i \rightsquigarrow p_j$.

*Communication is fair-lossy.* 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 $p_j$ permanently in the neighborhood receives $m$ from $p_i$ an infinite number of times, or $p_j$ is faulty. That is, if $p_i$ starts to send $m$ at time $t$ an infinite number of times, then, if $\rho((p_i, p_j), t') = 1, \forall t' \in (t, \infty)$, $p_j$ receives $m$ an infinity number of times if $p_j$ is correct. 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 [21, 22, 23].

*Mobility model.* Nodes in $\Pi$ 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 and Connectivity Assumptions

One important aspect on the design of FDs for dynamic networks 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 or numerous joins or leaves along the execution (characterizing a churn) could block the application and prevent any useful computation. Thus, to implement any global computation, the system should present some stability conditions that when satisfied for long enough time will be sufficient to satisfy the requirements of the application and terminate. In order to implement FDs with an unknown membership, processes should interact with some other process that never departs from the system to be known. If there is some process such that the rest of processes have no knowledge whatsoever of its identity, there is no algorithm that implements a FD with weak completeness [24]. *Completeness* characterizes the FD capability of suspecting every faulty process permanently. 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.

We consider then that a process $p_i$ *joins* the network at some point $t \in \mathcal{T}$ in time. Subsequently, $p_i$ must somehow communicate with the others in order to be known. In a wireless network, this can be done by simply broadcasting its identity to the neighbors. Due to this initial communication, every process $p_j$ is able to gather an initial partial knowledge $\Pi_j \subseteq \Pi$ about the system's membership which increases over the time along $p_j$'s execution. Let $\Pi_j(t)$ be the partial knowledge of $p_j$ by time $t$. 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*. When $p_i$ *leaves* the network at time $t' > t$, 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.

Let us thus define the status that a process may exhibit along the system execution:

DEFINITION 2.1. *Process Status. A process $p_i$ may assume the following status in the system.*

$join^t(p_i) \Leftrightarrow \exists t \in \mathcal{T}, \forall s < t,\ \psi(p_i, s) = 0 \wedge \psi(p_i, t) = 1$

$stable^t(p_i) \Leftrightarrow \exists t \in \mathcal{T}, \forall t' \geq t,\ \psi(p_i, t') = 1$

$faulty^t(p_i) \Leftrightarrow \exists s, t \in \mathcal{T}, s < t,\ \psi(p_i, s) = 1 \wedge \forall t' \geq t, \psi(p_i, t') = 0$

$known^t(p_i) \Leftrightarrow \exists p_j, \exists t \in \mathcal{T}, stable^t(p_j) \wedge p_i \in \Pi_j(t)$

The *failure pattern* of the system, namely $F(t)$, is the set of processes that have failed in the system by time $t$. That is, $F(t) = \{p_i : faulty^t(p_i)\}$. Similarly, $S(t)$, is the set of processes that are stable in the system by time $t$. That is, $S(t) = \{p_i : stable^t(p_i)\}$.

DEFINITION 2.2. *Membership. The membership of the system is the* KNOWN *set.*

$\text{STABLE} \stackrel{def}{=} \bigcup_{t \in \mathcal{T}} S(t)$

$\text{FAULTY} \stackrel{def}{=} \bigcup_{t \in \mathcal{T}} F(t)$

$\text{KNOWN} \stackrel{def}{=} \{p_i : \exists t \in \mathcal{T}, p_i \in \text{STABLE} \cup \text{FAULTY} \wedge known^t(p_i)\}$

Let $V_{KS} = \text{KNOWN} \cap \text{STABLE}$ and $E_{KS} \subseteq V_{KS} \times V_{KS}$. The graph $G_{KS} = (V_{KS}, E_{KS}) \subseteq G$ is the graph induced from the stable known nodes in $\Pi$, defining the TVG $\mathcal{G}_{\mathcal{KS}} = (V_{KS}, E_{KS}, \mathcal{T}, \rho, \psi) \subseteq \mathcal{G}$.

We can identify classes of TVG based on the temporal properties established by the entities. The classes are important because they imply necessary conditions and impossibility results for distributed computations. Notably, Class 3 (Connectivity over time) [20] is important for our study. It means that the TVG is connected over time.

ASSUMPTION 1. *Network connectivity over time.* In the system, represented by the TVG $\mathcal{G}_{KS}$, $\exists t \in \mathcal{T}, \forall t' \geq t$, $\forall p_i, p_j \in V_{KS}, p_i \rightsquigarrow p_j$ ($p_i$ *reaches* $p_j$). That is, after $t$, there is a *journey* $\mathcal{J}$, $\forall p_i, p_j \in \text{KNOWN} \cap \text{STABLE}$. For a communication purpose, we assume that each edge $e_i$ of $\mathcal{J}$ remains available until a message is delivered, thus $\rho(e_i, t) = 1, \forall t \in [t_i, t_i + \zeta(e_i, t_i)]$.

The connectivity Assumption 1 states that, in spite of changes in the topology, from some point in time $t$, the TVG $\mathcal{G}_{KS}$ is connected over time. This is a common assumption, mandatory to ensure reliable dissemination of messages to all stable processes in a dynamic network [25, 20] and thus to ensure the global properties of the failure detector [2, 24, 26, 27].

Recent works about radio communication advocate a "local" fault model, instead of a "global" fault model, as a suitable strategy to deal with the dynamics and unreliability of wireless channels in spite of failures [22, 28, 29, 23, 27]. They define bounds on the maximum number of local failures in order to reliably delivery data. Locality of failures can be interpreted as an uniform distribution of failures across the network and represents more accurately the reality of wireless channels. Following these recent works, the local fault model is the approach adopted in our work.

### 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 $\diamondsuit S$. 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 $\diamondsuit S^M$. This class keeps the same properties of $\diamondsuit S$, except that they are now valid to known processes, that are either stable or faulty.

DEFINITION 2.3. ***Eventually Strong FD with Unknown Membership*** *($\diamondsuit S^M$)*

*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 $\diamondsuit S^M$ class contains all the failure detectors that satisfy:*

*Strong completeness* $\stackrel{def}{=}$ $\{\exists t \in \mathcal{T},\ \forall t' \geq t,\ \forall p_i \in$ KNOWN $\cap$ FAULTY $\Rightarrow p_i \in susp_j,\ \forall p_j \in$ KNOWN $\cap$ STABLE$\}$;

*Eventual weak accuracy* $\stackrel{def}{=}$ $\{\exists t \in \mathcal{T},\ \forall t' \geq t,\ \exists p_i \in$ KNOWN $\cap$ STABLE $\Rightarrow p_i \notin susp_j,\ \forall p_j \in$ KNOWN $\cap$ STABLE$\}$.

## 3. TOWARDS A TIME-FREE FAILURE DETECTOR FOR THE $\diamondsuit S^M$ 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 considered in order to implement them. With this aim, two orthogonal approaches can be distinguished in the literature: the time-based and the time-free failure detection [30]. The time-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, 26] considers that the system satisfies a message exchange pattern on the execution of a *query-based* communication and is time-free. While the time-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 relative order that assumes that some nodes – not previously defined – have faster communications than the other ones). 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 [30].

### 3.1. Stable Query-Response Communication Mechanism

Our failure detector is time-free and based on a local QUERY-RESPONSE communication mechanism [26] 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($m'$) with a message $m'$ to $p_i$. Then, when $p_i$ has received at least $\alpha_i$ responses from different processes, the current QUERY-RESPONSE *terminates*. Without loss of generality, the response for $p_i$ itself is among the $\alpha_i$ responses.

Formally, the 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-Termination: Let $t$ be the time at which a process $p_i$ terminates to send a query. If $faulty^t(p_i)$ does not hold, then that query generates at least $\alpha_i$ RESPONSE($m'$) messages from a subset of $X_i$ processes, $|X_i| \geq \alpha_i$.

The QUERY–RESPONSE primitive has the following interface:
(i) QR-QUERY($m$): A QUERY is sent by a process $p_i$ to all the processes in its neighborhood;
(i) QR-DELIVER($X_i$, $M$): The set $M$ of RESPONSE messages sent from processes in $X_i$ in response to a QUERY is delivered to $p_i$;

An implementation of a couple of 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 $\alpha_i$ responses from its neighbors. Since the communication pattern followed by our FD is local, $\alpha_i$ is defined locally as a function of the expected number of stable known neighbors with whom $p_i$ may communicate at the time $t$ in which the QUERY is issued.

THEOREM 3.1. *To ensure liveness of the time-free* QUERY-RESPONSE *communication mechanism, a threshold $\alpha_i$ on the number of stable nodes in the neighborhood of $p_i$ must be established.*

*Proof.* Let us assume that $p_i$ issues a QUERY-RESPONSE, but it does not know $\alpha_i$. Assume that $f_i$ is the maximum number of faulty processes in $p_i$'s neighborhood and that $f_i$ neighbors crash at time $t$ during the QUERY. Since the set of responses received

by $p_i$ includes its own response, $\alpha_i = |N_i^t| - f_i + 1$. According to the QUERY-RESPONSE primitive pattern, $p_i$ waits for a number of $r_i > 0$ responses. However, on the occurrence of $f_i$ failures, only $\alpha_i$ responses are sent to $p_i$. Thus, if $r_i > \alpha_i$, $p_i$ will wait forever, violating then the termination property. Therefore, $p_i$ must know $\alpha_i$ in order to define $r_i$ and progress. □

PROPERTY 1. **Stable Termination Property ($\mathcal{SAT}\mathcal{P}$).** Let $p_i$ be a node which issues a QUERY. Thus, $\exists p_j \in \text{KNOWN} \cap \text{STABLE}, p_j \neq p_i$, which receives that QUERY and issues a RESPONSE to $p_i$.

For the failure detection problem, the *stable termination* is necessary for the reliable dissemination of the information to the whole network and consequent satisfaction of the accuracy and completeness properties. It is a guarantee that information from/to $p_i$ is going to be sent/received to/from at least a stable $p_j$ in its neighborhood. 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.

The local choice for $\alpha_i$ changes from previous works which consider a global value either proportional to the total number of correct processes [8] or the total number of stable processes [26] or the total number of faults [14] in the system. 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 dynamics and unreliability of wireless channels in spite of failures [22].

THEOREM 3.2. *To ensure $\mathcal{SAT}\mathcal{P}$ property, $|N_i^t| > 2f_i$, $\forall p_i$, $\forall t$.*

*Proof.* $\alpha_i = |N_i^t| - f_i + 1$ which ensures the liveness of query-response rounds. To ensure that at least one stable known node $p_j$ ($p_j \neq p_i$) receives the QUERY and sends a RESPONSE to $p_i$, responses must be issued by more than $f_i + 1$ nodes. Therefore, $\alpha_i > f_i + 1$ and, consequently, $|N_i^t| > 2f_i$. □

### 3.2. Behavioral Property

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 $p_j$, which is in the neighborhood of a stable known node $p_i$, the set of responses received by $p_j$ to its QUERY always includes a response from $p_i$, that is, the response of $p_i$ is always a winning response [30]. Moreover, as nodes may move, the $\mathcal{SRP}(p_i)$ also states that neighbors of $p_i$ eventually stop moving outside $p_i$'s neighborhood.

PROPERTY 2. **Stabilized Responsiveness Property ($\mathcal{SRP}$).** Let $X_j^{t'}$ be the set of processes from which $p_j$ has received responses to its QUERY sent at $t'$. Process $p_i$ satisfies $\mathcal{SRP}$ at time $t$ if:
$\mathcal{SRP}^t(p_i) \Leftrightarrow \exists t \in \mathcal{T}, stable^t(p_i) \wedge$

$$p_i \in X_j^{t'}, \forall p_j \in N_i^{t'}, \forall t' \geq t \vee faulty^{t'}(p_j) \wedge$$
$$p_j \in N_i^{t'} \Rightarrow p_j \in N_i^{t''}, \forall p_j, \forall t'' > t' \vee faulty^{t''}(p_j)$$

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

A discussion about how the properties in this section could be satisfied in practice is presented 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. An implementation of the QUERY-RESPONSE primitive is given in Algorithm 2.

**Notations.** We use the following notations:

- $knownTo_i$: 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.
- $susp_i$: 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 $\langle id, ct \rangle$.
- $X_i$: denotes the set of nodes from which $p_i$ has received responses to its last QUERY message.
- $susp\_from_i$: a vector containing in index $j$ the $susp_j$ set sent from $p_j$ in response to the last QUERY from $p_i$.
- $mist\_from_i$: a vector containing in index $j$ the $mist_j$ set sent from $p_j$ in response to the last QUERY from $p_i$.
- $Add(set, \langle id, ct \rangle)$: 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.** Algorithm 1 is composed of two tasks, $T1$ and $T2$, and the procedure $Update\_State()$, called

by these tasks in order to analyze the contents of a receiving message (QUERY or RESPONSE). Note that both messages contain information about suspected nodes and mistakes kept by the sending node $p_i$.

*Task T1: Generating suspicions.* This task is made up of an infinite loop. At each round, a QR-QUERY is invoked (line 8) and a QUERY($susp_i$, $mist_i$) message is sent to all nodes of $p_i$'s neighborhood (line 4, Alg. 2). On the invocation of QR-DELIVER (line 9), $p_i$ waits for at least $\alpha_i$ responses, which includes $p_i$'s own response (lines 6–11, Alg. 2). For each RESPONSE($susp_j, mist_j$) received from $p_j$, $p_i$ stores $p_j$ in $X_i$, $susp_j$ in $susp\_from_i[j]$ and $mist_j$ in $mist\_from_i[j]$. Then, based on the RESPONSE messages sent by the processes that answered to its query, $p_i$ updates its local information about suspicions and mistakes by calling, for each of these messages from $p_j$, the procedure *Update_State* (line 11), whose code is detailed below. Furthermore, considering its current information about partial knowledge of the system's membership, $p_i$ also includes new suspicions (lines 12-17): it starts suspecting each node $p_j$, not previously suspected ($p_j \notin susp_i$), which it knows ($p_j \in knownTo_i$), 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 15) and the suspicion information is then included in $susp_i$ with a tag which is greater than the previous mistake tag (line 14). 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 17).

*Task T2: Propagating suspicions and mistakes.* This task allows node $p_i$ to handle the reception of a QUERY message. Similarly to a RESPONSE message, $p_i$ calls the procedure *Update_State* in order to update its local information about suspicions and mistakes with that sent by the querying node $p_j$ (line 22). At the end of the task (line 23), $p_i$ sends to $p_j$ a RESPONSE message.

*Procedure Update_State.* It is responsible for analyzing the information about suspicions and mistakes sent by a process. 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 respectively handle the information received about suspected nodes (lines 27–33) and about mistaken nodes (lines 34–39). 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, a new mistake is detected if the receiving node $p_i$ is included in the suspected set of the QUERY message (line 29) with a greater tag. It is worth pointing out that procedure *Update_State* is also responsible for updating $p_i$'s partial knowledge of the system's membership (line 26).

*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-RESPONSE 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-RESPONSE messages with a greater tag (lines 29-30). 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-RESPONSE 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 38–39 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 38). 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 since only the process can generate a new mistake about itself (line 29). Thus, $p_x$ is removed from the local set $knownTo_i$ (line 39). 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-RESPONSE 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 $knownTo_i$. Fortunately, since local broadcast is fair-lossy, the QUERY-RESPONSE from $p_x$ is going to eventually arrive at $p_i$, if $p_i$ is stable, and, as soon as the message arrives, $p_i$ will once again add $p_x$ to its $know_i$ set (lines 20–26).

### 4.2. Practical Issues

The *stable termination* of the QUERY-RESPONSE primitive 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

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

```
1  init:
2
3      susp_i ← ∅; mist_i ← ∅; knownTo_i ← ∅
4
5
6  Task T1:
7  Repeat forever
8      QR-QUERY(susp_i, mist_i)
9      Upon QR-DELIVER(X_i, susp_from_i, mist_from_i)
10     For all p_j ∈ X_i do
11         Call upon Update_State (p_j, susp_from_i[j], mist_from_i[j])
12     For all p_j ∈ knownTo_i \ X_i | ⟨p_j, −⟩ ∉ susp_i do
13         If ⟨p_j, ct⟩ ∈ mist_i
14             Add(susp_i, ⟨p_j, ct + 1⟩)
15             mist_i = mist_i \ {⟨p_j, −⟩}
16         Else
17             Add(susp_i, ⟨p_j, 0⟩)
18 End repeat
19
20 Task T2: Upon reception of QUERY (susp_j, mist_j) from p_j
21
22     Call upon Update_State (p_j, susp_j, mist_j)
23     send RESPONSE (susp_i, mist_i) to p_j
24
25 Procedure Update_State (p_j, susp_j, mist_j):
26 knownTo_i ← knownTo_i ∪ {p_j}
27 For all ⟨p_x, ct_x⟩ ∈ susp_j do
28 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
29     If p_x = p_i
30         Add(mist_i, ⟨p_i, ct_x + 1⟩)
31     Else
32         Add(susp_i, ⟨p_x, ct_x⟩)
33         mist_i = mist_i \ {⟨p_x, −⟩}
34 For all ⟨p_x, ct_x⟩ ∈ mist_j do
35 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
36     Add(mist_i, ⟨p_x, ct_x⟩)
37     susp_i = susp_i \ {⟨p_x, −⟩}
38     If (p_x ≠ p_j)
39         knownTo_i ← knownTo_i \ {p_x}
```

the RESPONSE messages. This condition is attained when for example, the most widely used Random Waypoint Mobility Model [31] is considered.

Notice that the FD algorithm does not demand $f_i$ to be known, but $\alpha_i$, which is a threshold on the number of stable nodes in the neighborhood of $p_i$. But, how to define $\alpha_i$ ? Theoretically, $\alpha_i = |N_i^t| - f_i + 1$ and to ensure $\mathcal{S}at\mathcal{P}$ (Theorem 3.2), $\alpha_i > f_i + 1$. But, in practice, $\alpha_i$ may be defined to be the current number of nodes in the neighborhood of $p_i$ at time $t$. Since 1-hop neighbors are easily established (through MAC-level hello packets), $\alpha_i$ may represent all current nodes in the neighborhood of $p_i$. It is worth remarking that the value of $\alpha_i$ relates not only with the application density, 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, 32] are good examples of platforms which 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

**Algorithm 2** QUERY-RESPONSE Implementation

```
1  Upon a call to QR-QUERY (susp_i, mist_i) do
2
3      X_i ← susp_from_i ← mist_from_i ← ∅
4      broadcast QUERY (susp_i, mist_i) Enddo
5
6  When receive RESPONSE(susp_j, mist_j) from p_j do
7
8      X_i = X_i ∪ p_j
9      susp_from_i[j] = susp_j
10     mist_from_i[j] = mist_j
11     When (|X_i| ≥ α_i) trigger QR-DELIVER(X_i, susp_from_i, mist_from_i)
12
```

may be 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 between 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 that propose to implement a leader oracle [14] or to solve consensus in this type of network [32].

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 over the time; 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 (mobile nodes 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.

In practice, the FD protocol is going to be used by other protocols in the dynamic network on the fly. That is, it will be queried during specific periods, the time to help the overlying protocol (e.g., consensus) to detect the failures and converge. Thus, in practice, the membership that is considered is the one that could satisfy the requirements (of the model suggested) during that period. For each new execution of the overlying protocol, the membership should change (to take into account the system dynamics).

In consequence, in spite of the fact that the proposed model does not authorize existing "unknown processes" to be in the membership, it does allow the membership to be updated according with the system dynamics on each FD invocation. Evidently, to ensure the global properties (strong completeness and eventual weak accuracy), at some point in time, until the end of the FD execution (the time for the overlying protocol to converge), a set of stable processes should be formed.

## 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, assumptions and properties stated in Sections 2 and 3. Let us first make the following remarks.

**Observation 1.** The *last status* about a process $p_x$ concerning failures is represented by the tuple $\langle p_x, ct\rangle$ which has the greatest counter $ct$ in the network at some point $t$ in time. It is stored in a $susp_i$ or $mist_i$ set of some process $p_i$.

**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 14–15 and 32–33) and vice-versa (lines 36–37). The only exceptions occur in lines 17 and 30, but in both cases, due to predicates of lines 13 and 29, respectively, the observation is ensured.

**Observation 3.** Only $p_i$ can generate a new mistake about itself (lines 29–30). Moreover, the counter associated with the mistake is strictly increasing (this comes from predicate of lines 28 and 30). Finally, $p_i$ never removes its own mistake $\langle p_i, -\rangle$ from $mist_i$ (lines 12, 15, 33) since by definition it always belongs to the $X_i$ set.

**Observation 4.** Process $p_i \notin susp_i$ is always true. This follows from predicate of line 12, which is never satisfied, and predicate of line 29, which is always satisfied for $\langle p_i, -\rangle$; thus $p_i$ is never included in $susp_i$ in lines 14 and 32.

**Observation 5.** Algorithm 2 implements a QUERY–RESPONSE primitive in conjunction with *Task T2* of Algorithm 1. To implement QR-QUERY(), a QUERY($susp_i$, $mist_i$) is broadcast in line 4 - Alg. 2. When a process receives the QUERY in line 20, it sends a RESPONSE in line 23. When a RESPONSE($susp_j, mist_j$) is received from $p_j$ (line 6 - Alg. 2), then $p_j$ is stored in the $X_i$ set, and the message parameters are stored respectively in $susp\_from_i[j]$ and $mist\_from_i[j]$ (lines 8-10 - Alg. 2). When $|X_i| \geq \alpha_i$, the contents of all response messages received are delivered to $p_i$ via these 3 sets (line 11 - Alg. 2), which thus implements QR-DELIVER(). The QR-Termination can be achieved over fair-lossy local channels by the repeated broadcast of the query by the sender $p_i$ until it has received at least $\alpha_i$ responses from its neighbors. The other properties QR-Validity and QR-Uniformity are easily satisfied.

As stated in the Model (Section 2), a mobile node is either *moving* (to reach a neighborhood, possibly different from 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 (Theorem 5.1). Afterwards, we prove they hold for the generic case (Theorem 5.2). Lemmata 5.3 and 5.4 are used to prove Theorem 5.1 and Lemmata 5.7 and 5.8 are used to prove Theorem 5.2.

*Proof for the Specific Case of Pausing Nodes.*

LEMMA 5.1. *Let $p_i, p_j \in$ KNOWN$\cap$STABLE. Consider that, at time $t$, $p_i$ owns the last status $\langle p_x, ct\rangle$ about $p_x \in$ KNOWN in its $susp_i$ set (respectively, $mist_i$ set). If no new information $\langle p_x, ct'\rangle$, $ct' > ct$, is generated after $t$, then eventually $\forall p_j \in$ KNOWN $\cap$ STABLE will include $\langle p_x, ct\rangle$ in $susp_j$ (respectively, $mist_j$).*

*Proof.* By Assumption 1, $\exists t' \geq t$, there is a journey $\mathcal{J} : p_i = p_0, p_1, \ldots, p_{k-1}, p_k = p_j$ of stable known nodes between $p_i$ and $p_j$. 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 $p_{k-1} \in$ KNOWN $\cap$ STABLE, it will execute a QUERY-RESPONSE (lines 8-9) in order to broadcast to its neighbors, after time $t$, a message $m$ which contains $\langle p_x, ct\rangle$ in the $susp_{k-1}$ (respectively, $mist_{k-1}$) set. From Observation 5 and $\mathcal{SAT}\mathcal{P}$ (Property 1), the QUERY terminates and is received by at least $p_k \in$ KNOWN$\cap$STABLE in the neighborhood of $p_{k-1}$ (line 20). Thus, $p_k$ will execute lines 27-33 (respectively, lines 34-39). Since, by assumption, $ct$ is the greatest counter associated with $p_x$ in the network, $p_k$ executes line 32 (respectively, line 36) and adds $\langle p_x, ct\rangle$ to its own $susp_k$ set (respectively, $mist_k$ set). Thus, the claim is valid for $\forall p_j \in$ KNOWN $\cap$ STABLE and the lemma follows. □

LEMMA 5.2. *Infinitely often, $\forall p_i \in$ KNOWN, $\exists p_j \in$ KNOWN $\cap$ STABLE, $p_i \in knownTo_j$.*

*Proof.* From Observation 5 and $\mathcal{SAT}\mathcal{P}$ (Property 1), there is at least one $p_j \in$ KNOWN $\cap$ STABLE in the neighborhood of $p_i$ which receives the last QUERY-RESPONSE message sent from $p_i$; thus, $p_i \in knownTo_j$ (lines 20 and 26). However, $p_i$ can be removed from $knownTo_j$ in line 39 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 39 is the only point in which $p_i$ can be removed from $knownTo_j$. Thus, regarding this removal, two situations are possible:

Situation (1). Assume the last mistake raised by $p_i$ arrives firstly at $p_j$ in a QUERY (line 20) (or RESPONSE (line 9)) sent by $p_i$, that is $\langle p_i, ct\rangle \in mist_i$, $ct$ is the greatest counter associated to $p_i$. In this case, on the execution of *Update_State*(), the predicate of line 35 is satisfied, and lines 36–37 are executed, but not lines 38–39. Afterwards, if a QUERY (or RESPONSE) 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 35 will no more be satisfied and lines 38–39 are not executed. Thus $p_j$ will not remove $p_i$ from the $knownTo_j$ set, $p_i \in knownTo_j$.

Situation (2). Assume that, due to an asynchronism, the last mistake raised by $p_i$ arrives firstly at $p_j$ in a QUERY (line 20) (or RESPONSE (line 10)) sent by $p_k \neq p_i$, that is $\langle p_i, ct\rangle \in mist_k$, $ct$ is the greatest counter associated to $p_i$. Since this QUERY (or RESPONSE) from $p_k$ arrives at $p_j$ before the own message from $p_i$, on the execution of *Update_State*(), the predicate of line 35 is satisfied and lines 38–39 are executed. Thus $p_j$ removes $p_i$ from $knownTo_j$. Nonetheless, later on, the original QUERY message sent by $p_i$ in which $p_i \in mist_i$ arrives to $p_j$ at line 20. This holds because, by assumption, $p_j \in$ KNOWN$\cap$STABLE is the neighbor which receives the last QUERY message from $p_i$. In this case, process $p_j$ will execute line 26 including $p_i$ in $knownTo_j$. Moreover, since this mistake has already been taken into account and its counter is not greater than the existing one, the predicate of line 35 will no more be satisfied and lines 38–39 are no more executed. Thus, $p_j$ will keep $p_i$ in its $knownTo_j$ set, $p_i \in knownTo_j$. This concludes the proof. □

LEMMA 5.3. *Let $p_f \in$ KNOWN$\cap$FAULTY. Eventually, $p_f \in susp_i$, $\forall p_i \in$ KNOWN $\cap$ STABLE.*

*Proof.* Let us consider that $faulty^t(p_f)$ holds.

Remark 1. Since $p_f \in$ KNOWN, $p_f$ has sent at least one QUERY message before it crashed at $t$ and there is

at least a process $p_i \in$ KNOWN $\cap$ STABLE which has received this last QUERY (lines 20–26). Additionally, from Lemma 5.2, $p_f \in knownTo_i$. After the crash of $p_f$ at $t$, $p_i$ will never receive a RESPONSE message from $p_f$ in line 9, thus $p_f \notin X_i$ and $p_f \in knownTo_i$. In this case, in the next QUERY-RESPONSE round, if $p_f$ was not already suspected by $p_i$ (line 12), 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 13–15 and adds $\langle p_f, ct+1 \rangle$ in $susp_i$. Otherwise, $p_i$ executes line 17 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 12).

Remark 2. Thus, in the network, after $t$, the last status about $p_f$ will eventually be a suspicion. In this case, following Lemma 5.1 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 $\forall p_i \in$ KNOWN $\cap$ STABLE. □

LEMMA 5.4. *Let* $p_i \in$ KNOWN$\cap$STABLE. *If* $\mathcal{SRP}^t(p_i)$ *holds for* $p_i$ *at time* $t$, *then eventually* $p_i \notin susp_j$, $\forall p_j \in$ KNOWN $\cap$ STABLE.

*Proof.* Remark 1. A *new suspicion* over process $p_i$ is only raised if its RESPONSE is not received by some process $p_k$ in its neighborhood in response to a QUERY, that is, if $p_i \in knownTo_k$ and $p_i \notin X_k$ (line 12). Notice that $p_i \in knownTo_k$ if $p_k$ has received a QUERY (or RESPONSE) from $p_i$ in the past (lines 20, 10), that is $p_k \in N_i^s$, for some time $s \leq t$. According to $\mathcal{SRP}^t(p_i)$, after $t$, (i) $p_i \in X_k$ is always true, $\forall p_k \in N_i^{t'}, t' \geq t$ and, (ii) $\forall p_k,\ p_k \in N_i^{t'} \Rightarrow p_k \in N_i^{t''}, t'' > t'$. Thus, after $t$, $\forall p_k \in N_i^{t'}, \forall t' \geq t$ never adds $p_i$ in $susp_k$ raising a *new* suspicion. From Observation 4, $p_i \notin susp_i$. 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 $t$, the last status about $p_i$, represented by $\langle p_i, ct \rangle$, can be (1) an *old* suspicion ($\langle p_i, ct \rangle \in susp_j$), generated before $t$; (2) a mistake ($\langle p_i, ct \rangle \in mist_j$) or (3) none of the previous cases (if $p_i$ has never been suspected). In Case (1), following the propagation Lemma 5.1, $p_i$ will eventually receive a QUERY or a RESPONSE message from $p_j \in$ KNOWN$\cap$STABLE with $\langle p_i, ct \rangle \in susp_j$ (lines 9, 20) and call upon *Update_State*(), what will cause $p_i$ to generate a new mistake with a greater tag ($\langle p_i, ct+1 \rangle \in mist_i$) (line 30). The last status about $p_i$ is now a mistake and we fall in Case (2). In Case (2), following Lemma 5.1 and *Observation 2*, the mistake will be propagated to $\forall p_j \in$ KNOWN $\cap$ STABLE. Then, eventually, lines 36-37 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 5.1. *Algorithm 1 implements* $\Diamond S^M$ *failure detector, assuming a network of* KNOWN pausing *nodes.*

*Proof.* The strong completeness follows directly from Lemma 5.3. The eventual weak accuracy follows directly from Lemma 5.4 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 5.5. *Lemma 5.1 holds* $\forall p_i \in$ KNOWN *mobile nodes.*

*Proof.* The lemma follows directly from Lemma 5.1 for all pausing nodes. To take into account moving nodes, we should consider two cases.

Case (1): Assume that $p_m \in$ KNOWN $\cap$ STABLE is a moving node which has not yet the last status $\langle p_x, ct \rangle$ about process $p_x$. Let us assume that, due to Lemma 5.1, every pausing node $p_i \in$ KNOWN $\cap$ STABLE has added the last status $\langle p_x, ct \rangle$ in its $susp_i$ (respectively, $mist_i$) set before or at time $t''$. As soon as $p_m$ reaches the new neighborhood at $t''' \geq t''$, it will execute task T1 (to implement a QUERY-RESPONSE). From Observation 5 and $\mathcal{SatP}$ (Property 1), the QUERY terminates and $p_m$ receives a RESPONSE message from a pausing node $p_i$, carrying out $\langle p_x, ct \rangle$, the last status about $p_x$ (line 9). On the execution of *Update_State*() (line 11), since, by assumption, $ct$ is the greatest counter associated with $p_x$, $p_m$ executes line 32 (respectively, line 36) and adds $\langle p_x, ct \rangle$ to its own $susp_m$ set (respectively, $mist_m$ set).

Case (2): Assume that $p_m \in$ KNOWN $\cap$ STABLE is a moving node which has the last status $\langle p_x, ct \rangle$ about process $p_x$. As soon as $p_m$ reaches the new neighborhood at time $t'$, it will execute task T1 (to implement a QUERY-RESPONSE). From Observation 5 and $\mathcal{SatP}$ (Property 1), the QUERY terminates and there will be at least one node $p_i \in$ KNOWN$\cap$STABLE in the neighborhood of $p_m$, which receives the QUERY with the last status about $p_x$. Thus, by executing task T2, $p_i$ calls upon *Update_State*() (line 22) and adds $\langle p_x, ct \rangle$ in its $susp_i$ (line 32) (respectively, $mist_i$ set, line 36). Following the propagation Lemma 5.1 and knowing that Case (1) holds, eventually every $p_j \in$ KNOWN$\cap$STABLE will include $\langle p_x, ct \rangle$ in its $susp_j$ set (respectively, $mist_j$ set) and the lemma follows. □

LEMMA 5.6. *Lemma 5.2 holds* $\forall p_i \in$ KNOWN *mobile nodes.*

*Proof.* From Observation 5 and $\mathcal{SatP}$ (Property 1), for every QUERY-RESPONSE, issued by $p_i$ at task T1, there will be at least one $p_j \in$ KNOWN $\cap$ STABLE 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 5.2, $p_i \in$

$knownTo_j$ and the lemma holds. □

LEMMA 5.7. *Let* $p_f \in$ KNOWN∩FAULTY. *Eventually,* $p_f \in susp_i$, $\forall p_i \in$ KNOWN ∩ STABLE *mobile nodes.*

*Proof.* Let us consider that $faulty^t(p_f)$. The lemma follows directly from Lemma 5.3 for all pausing nodes. To take into account moving nodes, two cases are possible. Let us first observe that, from Lemma 5.6 and knowing that the last QUERY-RESPONSE issued by $p_f$ (either moving or pausing) before $t$ has been received, there is $p_i \in$ KNOWN ∩ STABLE (either moving or pausing), such that $p_f \in knownTo_i$.

Case (1): Assume that $p_m \in$ KNOWN ∩ STABLE is a moving node which has not yet the last status about process $p_f$ after $t$; and, assume that by Lemma 5.3, every pausing node $p_j \in$ KNOWN ∩ STABLE has added $p_f$ in its $susp_j$ set after $t$. In this case, due to Lemma 5.5 (Case 1), $p_m$ will add $p_f$ in its $susp_m$.

Case (2): Assume that $p_m \in$ KNOWN ∩ STABLE is a 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 5.3 (Remark 1), $p_i$ will add $\langle p_f, ct+1\rangle$ in $susp_m$ (i.e., $susp_i$) and remove $p_f$ from $mist_m$ (i.e., $mist_i$). 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 5.3 (Remark 1). When $p_m$ reaches its new neighborhood after $t$, due to the propagation Lemma 5.5 (Case 2), this information about the suspicion of $p_f$ will be propagated to every $p_i \in$ KNOWN ∩ STABLE.

Finally, for every Case, from the same arguments of Lemma 5.3 (Remark 2) and considering Lemma 5.5, $p_f \in susp_i$ is always true $\forall p_i \in$ KNOWN ∩ STABLE. □

LEMMA 5.8. *Let* $p_i \in$ KNOWN∩STABLE. *If* $\mathcal{SRP}^t(p_i)$ *holds for* $p_i$ *at time* $t$, *then eventually* $p_i \notin susp_j$, $\forall p_j \in$ KNOWN ∩ STABLE *mobile nodes.*

*Proof.* The lemma follows directly from Lemma 5.4 for all pausing nodes. Due to same arguments of Lemma 5.4 (Remark 1), after $t$, $\forall p_k \in N_i^{t'}, \forall t' \geq t$ never adds $p_i$ in $susp_k$, raising a *new* suspicion. Assume that $p_m \in$ KNOWN ∩ STABLE is a moving node. Let us consider that $p_m$ reaches its new neighborhood at time $t' \geq t$. By hypothesis, if $p_i \in N_m^{t'}$ then $p_i \in N_m^{t''}, p_i \in X_m^{t''}, t'' \geq t'$ is always true, thus $p_m$ never adds $p_i$ in $susp_m$. If $p_i \notin N_m^{t'}$, two cases are possible:

Case (1): $p_m$ has the last status $\langle p_i, ct\rangle$ about $p_i$. The following situations are possible. *Situation (1)*: $p_m$ suspects $p_i$ ($\langle p_i, ct\rangle \in susp_m$). This can be an *old* suspicion, generated before $t$ or a *new* one, generated after $t$ due to $p_m$'s move. In this last case, $p_i \in knownTo_m$ and $p_i \notin X_m$, because $p_m$ will no longer receive RESPONSE messages from $p_i$, since $p_m$ moves. Thus, $p_m$ will suspect $p_i$ (executing lines 12–17). According to Lemma 5.4 (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 ($\langle p_i, ct+1\rangle \in mist_i$) that, due to Lemma 5.5, is propagated along the network. Finally, $\langle p_i, ct+1\rangle \in mist_m$ and $\langle p_i, -\rangle \notin susp_m$. Now, the last status about $p_i$ is not a suspicion and we fall in Situation (2). *Situation (2)*: $p_m$ does not suspect $p_i$. Following Lemma 5.4 (Remark 2, Cases 2 and 3) and Lemma 5.5, $\forall p_j \in$ KNOWN∩STABLE (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 $p_i$. Due to Lemma 5.5, after time $t'$, $p_m$ succeed to update its state with the last information about $p_i$. Following Lemma 5.4 and 5.5, eventually $p_i \notin susp_m$ is always true. Thus, the lemma follows for $\forall p_j \in$ KNOWN ∩ STABLE. □

THEOREM 5.2. *Algorithm 1 implements* $\Diamond S^M$ *failure detector, assuming a network of* KNOWN mobile *nodes.*

*Proof.* The strong completeness property follows directly from Lemma 5.7. The eventual weak accuracy property follows directly from Lemma 5.8 and the theorem follows. □

## 6. 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 $200m$x$1800m$. They thus have both the same surface, number of nodes but not the same network diameter. Such a difference aims at studying the propagation of failure suspicions and mistakes over the network.

We consider that every $p_i$ has at least 5 neighbors ($|N_i| > 5$) and that at most 2 neighbors may crash ($f_i = 2$). Therefore, $\alpha_i$ is the same for all $p_i$. The total number of nodes $N$ is fixed to 100 and it is uniformly distributed over the region. Each simulation lasts 30 minutes. In the square configuration the minimum number of neighbors $|N_i|$ is equal to 7, the maximum is 16 and the average is 10. For the rectangle configuration, the minimum, maximum, and average number of neighbors are 9, 6, and 16 respectively.

We have considered that 10% of the nodes can fail, i.e., 10 faults have been uniformly injected at every 70s starting at 10s (10s, 80s, 150s, etc.). The one-hop network delay $\delta$ is computed using the bandwidth (2Mb/s) and size of messages since the delay of propagation within a range is negligible. Then $\delta$ is less than 1ms[4]. In our experiments, we assume that MAC layer provide a local reliable broadcast.

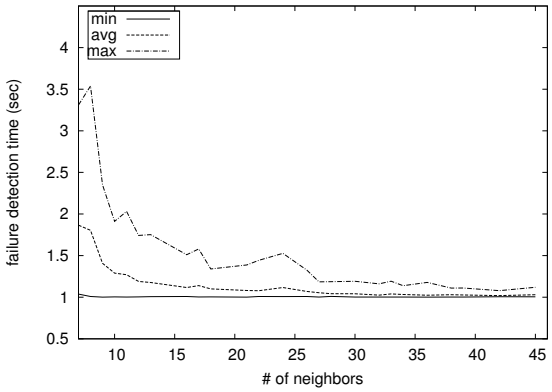
Concerning the implementation of our FD, it is not feasible that a node continuously broadcasts a QUERY

[4]This value is negligible in comparison to delay between two queries.

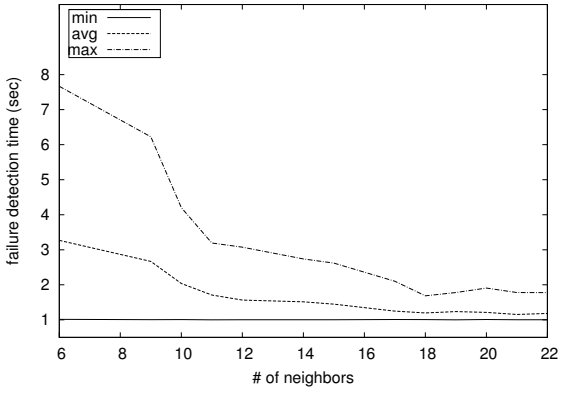
message since the network would be overloaded with messages. To overcome this problem, we have included a delay of $\Delta = 1s$ between lines 8 and 9 of the Algorithm 1. However, by adding this waiting period, process $p_i$ may receive more than $\alpha_i$ replies. Therefore, the extra replies will also be included in the $X_i$ set of this process (line 10), reducing then the number of false suspicions. It is worth remarking that this improvement does not change the protocol correctness.

### 6.1. Failure Detection

In order to evaluate the completeness property of our FD, we have measured the impact of the number of neighbors on the failure detection time. To this end, the transmission range $r$ varied from $100m$ to $380m$ which results in the variation of the number of neighbors. For each number of neighbors, we have measured the average, maximum, and minimum failure detection time considering the 90 correct nodes, as shown in Figure 1. The maximum failure detection time characterizes, for each of the different number of neighbors, the time for all nodes to detect a failure (strong completeness).




(a) 600x600 region



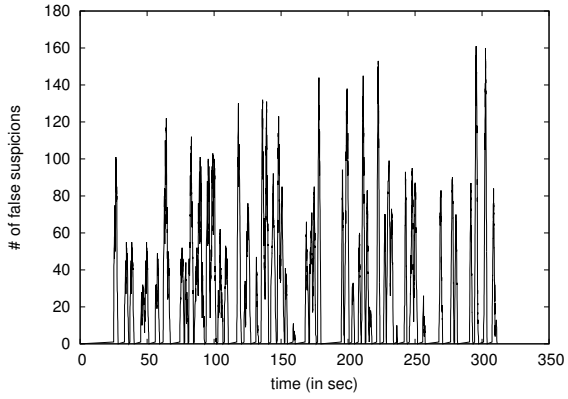

(b) 200x1800 region

**FIGURE 1.** Failure detection time vs. number of neighbors

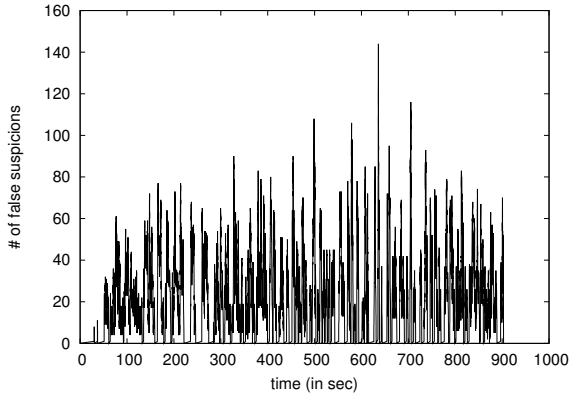
We observe that there is no false suspicion. The failure detection time decreases with the number of neighbors. This happens because failure detection information is included in QUERY messages which spreads faster over the network when the density increases.

### 6.2. Impact of mobility

We have evaluated the accuracy property when both one and ten nodes located at one boundary of the network move at a speed of 2m/s. The range transmission $r$ is set to 100m for all $p_i$. 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 and that at least $\alpha_j$ nodes will reply to the query of $p_j$ after $p_m$ moves.




(a) 600x600 region



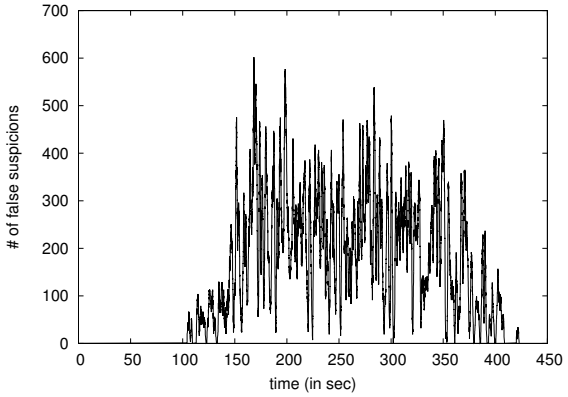

(b) 200x1800 region

**FIGURE 2.** Total number of false suspicions when one node moves

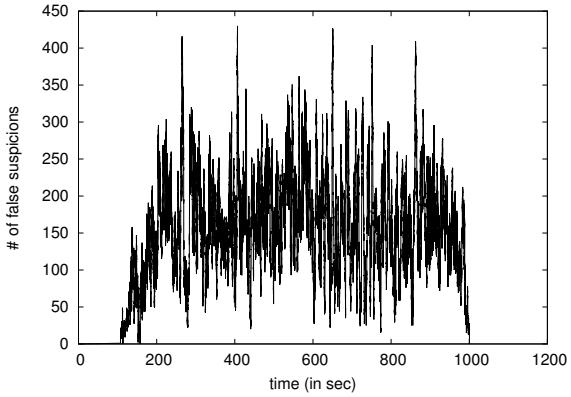
For each experiment, the total number of false suspicions has been measured. Figure 2 (respectively, 3) shows the number of false suspicions between the moment that just one node (respectively, 10 nodes) stars moving at 20s (respectively, 100s and every 5s) for both the square and rectangle region configurations.

We observe in both Figures that false suspicions are rather punctual: the number of false suspicions

increases very fast but decreases very fast too. This behavior can be explained because false suspicions are generated around the moving node when it changes of neighborhood. These suspicions are quickly corrected by the moving node itself as soon as it receives the query from its old neighbors which usually remain close to it (in most cases they are at one hop).



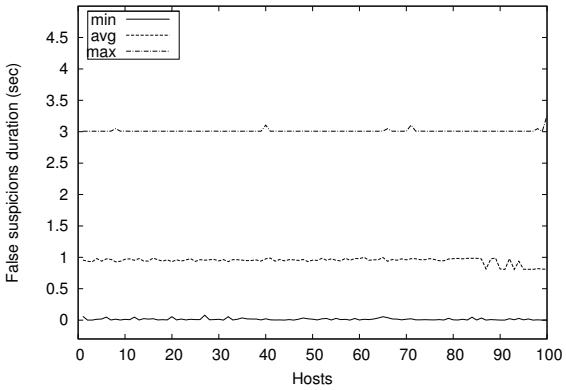

(a) 600x600 region



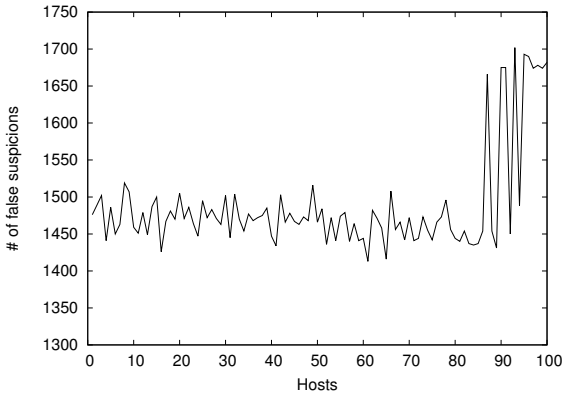

(b) 200x1800 region

**FIGURE 3.** Total number of false suspicions when ten nodes move

Figures 4.(a) and 4.(b) respectively show the distribution of mistake duration, i.e., how long in average a node is erroneously suspected, 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. Thus, we observe that the 10 moving nodes have a number of suspicions 15% higher than the other nodes. 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) Distribution of false suspicions durations in 200x1800 region




(b) Distribution of false suspicions in 200x1800 region

**FIGURE 4.** Distribution of mistakes when ten nodes move

**Synthesis.** From the results of the above experiments, we can outline two key properties of our failure detector:

- the average failure detection time is short thanks to the local query-response approach: as soon as a node fails, its neighbors start to suspect it in the next round. Then, in networks with high density and short diameter, the detection time converges to the inter-query delay.
- our failure detector is highly reactive in correcting false suspicions when node moves: mobility implies a relatively high number of false suspicions around the moving nodes, but the latter detect them very fast. Then, these nodes immediately generate mistake messages which limit the propagation of wrong suspected information.

## 7. RELATED WORK

**Scalable Approaches.** 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 [34, 35]. Practically, the randomization makes the definition of timeout values difficult. In [36], a scalable hierarchical failure adapted for Grid configurations is proposed. However, the global configuration of the network is initially known by all nodes. Whereas in [37], authors identify important problems on the design of scalable failure detectors for Grid architectures.

**The Heartbeat Approach.** Aguilera *et al.* [9] propose the heartbeat FD which does not assume a network of fully connectivity and tolerates message losses. Tucci *et al.* [10] implement a heartbeat FD for the infinite arrival model and show how to use it to implement a FD of the $\Omega$ class. The $\Omega$ class ensures that eventually each correct process is going to trust the same correct process, considered as the *leader*. The solution assumes fair-lossy channels, but for a synchronous environment. Hutle [38] proposes a $\diamondsuit P$ 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 time-based. As far 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 $\Omega$ class. Nonetheless, it does not tolerate node mobility.

**Probabilistic Approach for wireless ad-hoc networks.** 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. Upon 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. Contrarily to our approach, this work considers a cluster-based communication architecture and implements a FD of the class $\diamondsuit P$ which provides probabilistic guarantees for the accuracy and completeness properties; moreover, it does not consider mobility.

**Local Failure Detection Approach.** 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. He advocates the use of this local detection as an appropriate abstraction to deal with mobility and resources lack in wireless sensor networks. Unlike our solution that allows the implementation of a $\diamondsuit S^M$ FD, this work implements an eventually perfect *local* failure detector of the class $\diamondsuit P$, i.e., it provides strong completeness and eventual strong accuracy but with regard to a node's neighborhood.

**Time-Free Approach for Omega.** 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). An 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. Contrarily to our approach, this work considers a hybrid network of mobile and static nodes; moreover, it implements an $\Omega$ 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. We consider that this protocol is not very well adapted to ad-hoc networks, since it makes strong assumptions on the connectivity and global knowledge of MSSs. It considers that all MSSs form a complete graph and moreover that the maximum number of failures is known.

**FD Application.** We believe that our $\diamondsuit S^M$ FD will be of great interest to implement consensus algorithms, such as the one proposed by Greve *et al.* [15], 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 $\diamondsuit S$).

**Synthesis.** Table 1 shows a panorama of the FDs for mobile and wireless networks 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) considered failure

model, (6) strategy followed to detect failures, (7) the use of timers to detect failures, (8) the satisfaction of the membership property by the network, (9) the use of local communication for detection, (10) the provided FD class. The work presented herein exhibits the most generic features. It implements a time-free query-based deterministic FD suitable for any dynamic wireless network topology.

## 8. CONCLUSION

This paper has suggested a model able to implement unreliable failure detectors in mobile wireless networks, such as WMNs or WSNs and provides the specification of a new class of failure detectors for this context: the $\diamondsuit S^M$ class (eventually strong FD with unknown membership). It presents an algorithm able to implement a time-free $\diamondsuit S^M$ FD which is proved to be correct when the underlying network satisfies some assumptions regarding stability, connectivity, and the pattern of messages exchanged by the nodes.

## REFERENCES


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

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

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

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

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

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

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

[8] Mostefaoui, A., Mourgaya, E., and Raynal, M. (2003) Asynchronous implementation of failure detectors. *Proc. of Int. Conf. on Dependable Systems and Networks*, pp. 351–360.

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

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

[11] Friedman, R. and Tcharny, G. (2009) Evaluating failure detection in mobile ad-hoc networks. *Int. J. Pervasive Computing and Communications*, **5**, 476–496.

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

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

[14] Cao, J., Raynal, M., Travers, C., and Wu, W. (2007) The eventual leadership in dynamic mobile networking environments. *3th Pacific Rim Int. Symp. on Dependable Computing*, pp. 123–130.

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

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

[17] Greve, F., Sens, P., Arantes, L., and V.Simon (2011) A failure detector for wireless networks with unknown membership. *Euro-Par Conference, LNCS 6853*, pp. 27–38.

[18] Greve, F., Arantes, L., and Sens, P. (2011) What model and what conditions to implement unreliable failure detectors in dynamic networks ? *TADDS - 3rd Work. on Theoretical Aspects of Dynamic Distributed Systems with DISC*.

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

[20] Casteigts, A., Flocchini, P., Quattrociocchi, W., and Santoro, N. (2011) Time-varying graphs and dynamic networks. Technical report. University of Ottawa.

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

[22] Koo, C.-Y. (2004) Broadcast in radio networks tolerating byzantine adversarial behavior. *23th ACM Symp. on Principles of distributed computing*, pp. 275–282.

[23] Bhandari, V. and Vaidya, N. H. (2007) Reliable local broadcast in a wireless network prone to byzantine failures. *4th ACM Int. Work. on Foundations of Mobile Computing*.

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

[25] Casteigts, A., Chaumette, S., and Ferreira, A. (2010) Characterizing topological assumptions of distributed algorithms in dynamic networks. *Structural Information and Communication Complexity Conf.*, pp. 126–140.

[26] Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and Abbadi, A. (2005) From static distributed systems to dynamic systems. *24th IEEE Symp. on Reliable Distributed Systems*, pp. 109–118.

**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<br>Message omission |
| Tai *et al.* [12] | Static | Known | Arbitrary | Static connected graph of cluster heads | Crash<br>Message omission |
| Sridhar [13] | Mobile | Known | Arbitrary | Connected graph | Crash |
| Cao *et al.* [14] | Static and Mobile | Known (static)<br>Unknown (mobile) | $f$ (fixed) | Complete graph of static nodes | Crash (only mobile)<br>Reliable channels |
| **Our protocol** | **Mobile** | **Unknown** | $N_i/2$ | **Connected graph** | **Crash**<br>**Message omission** |

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


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

[28] Pelc, A. and Peleg, D. (2005) Broadcasting with locally bounded byzantine faults. *Inf. Process. Lett.*, **93**, 109–115.

[29] Bhandari, V. and Vaidya, N. H. (2005) On reliable broadcast in a radio network. *24th ACM Symp. on Principles of distributed computing*, pp. 138–147. ACM.

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

[31] Camp, T., Boleng, J., and Davies, V. (2002) 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**, 483–502.

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

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

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

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

[36] Bertier, M., Marin, O., and Sens, P. (2002) Implementation and performance evaluation of an adaptable failure detector. *Proc. of the Int. Conf. on Dependable Systems and Networks*, Washington, DC, USA, June, pp. 354–363.

[37] Hayashibara, N., Cherif, A., and Katayama, T. (2002) Failure detectors for large-scale distributed systems. *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pp. 404–409.

[38] Hutle, M. (2004) An efficient failure detector for sparsely connected networks. *Proc. of the IASTED Int. Conf. on Parallel and Distributed Computing and Networks*, pp. 369–374.