

Eventually Strong Failure Detector with Unknown Membership

FABIOLA GREVE^{1,*}, PIERRE SENS², LUCIANA ARANTES² AND VÉRONIQUE SIMON²

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*

*Corresponding author: fabiola@dcc.ufba.br

The distributed computing scenario is rapidly evolving for integrating self-organizing and dynamic wireless networks. Unreliable failure detectors (FDs) are classical mechanisms that provide information about process failures and can help systems to cope with the high dynamics of these networks. A number of failure detection algorithms have 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 FD protocol that implements a new class of detectors, namely $\diamond S^M$, which adapts the properties of the $\diamond S$ class to a dynamic network with an unknown membership. It has the interesting feature of being time-free, so that it does not rely on timers to detect failures; moreover, it tolerates the mobility of nodes and message losses.

Keywords: unreliable failure detector; dynamic distributed systems; wireless mobile networks; asynchronous systems

Received 14 September 2011; revised 13 April 2012

Handling editor: Jane Hillston

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 that can cope with the high dynamics of these systems is a challenge. A failure detector (FD) is a fundamental service able to help in the development of fault-tolerant distributed systems. *Unreliable FDs* 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 by $\diamond S$. 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). Here $\diamond S$ is the weakest class allowing one 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 wireless mesh networks (WMNs) [3], wireless sensor networks (WSNs) [4]. These kinds of networks share the following properties: (1) a node does not necessarily know all the nodes of the network; (2) the message transmission delay between nodes is highly unpredictable; (3) the network is not fully connected, and thus a message sent by a node might be routed through a set of intermediate nodes until it reaches its destination; (4) a node can move around and thus change its 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 have 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 [5–7]. 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 the 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 that 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 an 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 the mobility of nodes. Few implementations of unreliable FDs focus on wireless mobile networks [11–13]. 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 makes two contributions: (i) the proposition of a model and the definition of the $\diamond S^M$ class of FDs; (ii) a new time-free FD algorithm that implements the class $\diamond S^M$ under a wireless mobile network.

To implement unreliable FDs in an asynchronous dynamic system of mobile nodes, some assumptions about the underlying system should be made. Owing 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 FDs suitable for mobile networks with unknown membership. The class of *eventually strong FDs with unknown membership* (namely, $\diamond S^M$) adapts the properties of the $\diamond S$ class to a dynamic system with an unknown membership.

The second contribution is the proposition of an FD algorithm that implements $\diamond S^M$. 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, and 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 the mobility of nodes. Correctness proofs are given that the algorithm can implement FDs of class $\diamond S^M$ when some properties are satisfied by the underlying system. We believe that our FD of class $\diamond S^M$ may be successfully adopted to implement coordination protocols in a dynamic set, such as the one proposed by Greve and Tixeuil [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 first in [16] as a brief announcement. Afterward, in [17], we have provided a first version of the algorithm to implement $\diamond S^M$, but for a slightly different model. Then, in [18], a discussion about an appropriate model to implement unreliable FDs in dynamic systems is presented.

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. Sections 4 and 5 present a time-free FD of the $\diamond S^M$ 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 that 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, and 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 Π of n nodes, namely, $\Pi = \{p_1, \dots, p_n\}$. 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 in dealing with natural disasters.

The membership is unknown. Processes are not aware about Π 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 know 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 Π , 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 possibly 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. Owing to arbitrary joins, leaves and failures, the network is represented by a communication graph with a dynamic topology, and 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) $\rho : E \times \mathcal{T} \rightarrow \{0, 1\}$ 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) $\zeta : E \times \mathcal{T} \rightarrow \mathbb{N}$ 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, and thus, we consider that ζ exists but cannot be estimated and (5) $\psi : V \times \mathcal{T} \rightarrow \{0, 1\}$ 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 $\psi(p_i, t) = 0$.

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 $\text{Deg}_i^t = |E_i^t|$.

Given a TVG \mathcal{G} , the graph $G = (V, E)$ is called the *underlying graph* of \mathcal{G} . G should be considered 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 $\mathcal{J} = \{(e_1, t_1), (e_2, t_2), \dots, (e_k, t_k)\}$, such that $\{e_1, e_2, \dots, e_k\}$ 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 medium access control (MAC) layer of the underlying wireless network provides a protocol that reliably delivers broadcast data, even in the presence of unpredictable behaviors, such as fading, collisions and interference; solutions in this sense have been proposed in [21–23].

Mobility model. 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 movement. Then, from the viewpoint of a neighbor, it is not possible to distinguish between a moving, a leave or a crash of p_m . During the change of neighborhood, 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. 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 an 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 that 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. Owing 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 having entered the system at 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, and thus, it is considered 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:*

$$\begin{aligned} \text{join}^t(p_i) &\Leftrightarrow \exists t \in \mathcal{T}, \forall s < t, \psi(p_i, s) \\ &= 0 \wedge \psi(p_i, t) = 1, \\ \text{stable}^t(p_i) &\Leftrightarrow \exists t \in \mathcal{T}, \forall t' \geq t, \psi(p_i, t') = 1, \\ \text{faulty}^t(p_i) &\Leftrightarrow \exists s, t \in \mathcal{T}, s < t, \psi(p_i, s) = 1 \wedge \forall t' \\ &\geq t, \psi(p_i, t') = 0, \\ \text{known}^t(p_i) &\Leftrightarrow \exists p_j, \exists t \in \mathcal{T}, \text{stable}^t(p_j) \wedge p_i \in \Pi_j(t). \end{aligned}$$

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, $F(t) = \{p_i : \text{faulty}^t(p_i)\}$. Similarly, $S(t)$ is the set of processes that are stable in the system by time t . That is, $S(t) = \{p_i : \text{stable}^t(p_i)\}$.

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

$$\begin{aligned} \text{STABLE} &\stackrel{\text{def}}{=} \bigcup_{t \in \mathcal{T}} S(t), \\ \text{FAULTY} &\stackrel{\text{def}}{=} \bigcup_{t \in \mathcal{T}} F(t), \\ \text{KNOWN} &\stackrel{\text{def}}{=} \{p_i : \exists t \in \mathcal{T}, p_i \in \text{STABLE} \cup \text{FAULTY} \\ &\wedge \text{known}^t(p_i)\}. \end{aligned}$$

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 Π , defining the TVG $\mathcal{G}_{KS} = (V_{KS}, E_{KS}, \mathcal{T}, \rho, \psi) \subseteq \mathcal{G}$.

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, and 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 [20, 25] and thus to ensure the global properties of the FD [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, 23, 27–29]. They define bounds on the maximum number of local failures in order to reliably deliver data. The 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. An FD of class $\diamond S^M$

Unreliable FDs provide information about the liveness of processes in the system [2]. Each process has access to a local FD that outputs a list of processes that it currently suspects of being faulty. The FD is *unreliable* in the sense that it may erroneously add to its list a process that is actually correct. But if the detector later believes that suspecting this process is a mistake, it then removes the process from its list.

FDs 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 an FD in a dynamic set. Then, we define the class of *Eventually Strong FDs* 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 either stable or faulty.

DEFINITION 2.3 (Eventually strong FD with unknown membership ($\diamond S^M$)). 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 $\diamond S^M$ class contains all the FDs that satisfy

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

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

3. TOWARD A TIME-FREE FD FOR THE $\diamond S^M$ CLASS

None of the FD 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 Mostefaoui *et al.* [8] and developed so far by Cao *et al.* [14] and Mostefaoui *et al.* [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 FD 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 α_i responses from different processes, the current QUERY–RESPONSE *terminates*. Without loss of generality, the response for p_i itself is among the α_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 $\text{faulty}^t(p_i)$ does not hold, then that query generates at least α_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;
- (ii) 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 α_i responses from its neighbors. Since the communication pattern followed by our FD is local, α_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 at which the QUERY is issued.

THEOREM 3.1. *To ensure liveness of the time-free QUERY–RESPONSE communication mechanism, a threshold α_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 α_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 α_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 α_i in order to define r_i and progress. \square

PROPERTY 1 (Stable termination property (*SatP*)).

Let p_i be a node that 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 in the membership of the system.

The local choice for α_i changes from previous works that 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 that 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 the SatP 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, $|N_i^t| > 2f_i$. \square

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 *SRP*. Thus, *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 *SRP*(p_i) also states that neighbors of p_i eventually stop moving outside p_i ’s neighborhood.

PROPERTY 2 (Stabilized responsiveness property (*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 *SRP* at time t if:

$$\begin{aligned} \text{SRP}^t(p_i) \Leftrightarrow \exists t \in \mathcal{T}, \text{stable}^t(p_i) \wedge \\ p_i \in X_j^{t'}, \forall p_j \in N_i^{t'}, \forall t' \geq t \vee \text{faulty}^{t'}(p_j) \wedge \end{aligned}$$

$$\begin{aligned} p_j \in N_i^{t'} \Rightarrow p_j \in N_i^{t''}, \forall p_j, \forall t'' > t' \\ \vee \text{faulty}^{t''}(p_j). \end{aligned}$$

This property denotes the ability of a stable known node p_i to reply, among the first α_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 from being 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 1.2 after the protocol’s explanation.

4. AN FD ALGORITHM FOR THE $\diamond S^M$ CLASS

4.1. Algorithm description

Algorithm 1 describes our protocol for implementing an 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:

- (i) *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.
- (ii) *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.
- (iii) *mist_i*: denotes the set of nodes that 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 $\langle id, ct \rangle$.
- (iv) *X_i*: denotes the set of nodes from which p_i has received responses to its last QUERY message.
- (v) *susp_from_i*: a vector containing in index j the *susp_i* set sent from p_j in response to the last QUERY from p_i .
- (vi) *mist_from_i*: a vector containing in index j the *mist_i* set sent from p_j in response to the last QUERY from p_i .
- (vii) *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 .

Algorithm 1 Time-free implementation of a $\diamond S^M$ FD.

```

1  init:
2
3  |  $susp_i \leftarrow \emptyset; mist_i \leftarrow \emptyset; knownTo_i \leftarrow \emptyset$ 
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  $p_j \in X_i$  Do
11 | | Call upon Update_State ( $p_j, susp\_from_i[j], mist\_from_i[j]$ )
12 | | For all  $p_j \in knownTo_i \setminus X_i \mid \langle p_j, - \rangle \notin susp_i$  Do
13 | | | If  $\langle p_j, ct \rangle \in mist_i$  Then
14 | | | | Add( $susp_i, \langle p_j, ct + 1 \rangle$ )
15 | | | |  $mist_i = mist_i \setminus \{\langle p_j, - \rangle\}$ 
16 | | | Else
17 | | | | Add( $susp_i, \langle p_j, 0 \rangle$ )
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  $knownTo_i \leftarrow knownTo_i \cup \{p_j\}$ 
27 For all  $\langle p_x, ct_x \rangle \in susp_j$  Do
28 If  $\langle p_x, - \rangle \notin susp_i \cup mist_i$  or  $(\langle p_x, ct \rangle \in susp_i \cup mist_i$  and  $ct < ct_x)$  Then
29 | If  $p_x = p_i$  Then
30 | | Add( $mist_i, \langle p_i, ct_x + 1 \rangle$ )
31 | Else
32 | | Add( $susp_i, \langle p_x, ct_x \rangle$ )
33 | |  $mist_i = mist_i \setminus \{\langle p_x, - \rangle\}$ 
34 | For all  $\langle p_x, ct_x \rangle \in mist_j$  Do
35 | If  $\langle p_x, - \rangle \notin susp_i \cup mist_i$  or  $(\langle p_x, ct \rangle \in susp_i \cup mist_i$  and  $ct < ct_x)$  Then
36 | | Add( $mist_i, \langle p_x, ct_x \rangle$ )
37 | |  $susp_i = susp_i \setminus \{\langle p_x, - \rangle\}$ 
38 | | If  $(p_x \neq p_j)$  Then
39 | | |  $knownTo_i \leftarrow knownTo_i \setminus \{p_x\}$ 

```

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, Algorithm 2). On the invocation of QR-DELIVER (line 9), p_i waits for at least α_i responses, which includes p_i 's own response (lines 6–11, Algorithm 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

Algorithm 2 QUERY–RESPONSE implementation.

```

1  Upon a call to QR-QUERY ( $susp_i, mist_i$ ) do
2
3  |  $X_i \leftarrow susp\_from_i \leftarrow mist\_from_i \leftarrow \emptyset$ 
4  | broadcast QUERY ( $susp_i, mist_i$ ) Enddo
5
6  When receive RESPONSE ( $susp_j, mist_j$ ) from  $p_j$  do
7
8  |  $X_i = X_i \cup p_j$ 
9  |  $susp\_from_i[j] = susp_j$ 
10 |  $mist\_from_i[j] = mist_j$ 
11 | When ( $|X_i| \geq \alpha_i$ ) trigger QR-DELIVER ( $X_i, susp\_from_i, mist\_from_i$ )
12

```

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 that 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 item of information about the 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.

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 item of 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). Note, however, that this condition is not sufficient to detect the mobility because p_x can be a neighbor of p_i but 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 first. If that is the case, p_x has been unduly removed from *knownTo_i*. Fortunately, since the 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 the RESPONSE messages. This condition is attained when, for example, the most widely used Random Waypoint Mobility Model [31] is considered.

Note that the FD algorithm does not demand f_i to be known, but α_i , which is a threshold on the number of stable nodes in the neighborhood of p_i . But, how to define α_i ? Theoretically, $\alpha_i = |N_i^t| - f_i + 1$ and to ensure $\text{Sat}\mathcal{P}$ (Theorem 3.2), $\alpha_i > f_i + 1$. But, in practice, α_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), α_i may represent all current nodes in the neighborhood of p_i . It is worth remarking that the value of α_i relates not only to the application density, but also to 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.

WMNs, WSNs and infra-structured mobile networks [14, 32] are good examples of platforms that 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 fixed node. A WSN may be composed of stationary nodes and can be organized in clusters, so that the 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 (MHs) and mobile support stations (MSSs). An MH is connected to an 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 WMNs, CHs for WSNs or MSSs for infra-structured networks) eventually form a strongly connected component of stable nodes over 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 WMNs, CMs for WSNs 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 $\diamond S^M$, as for instance, the time to execute a consensus algorithm.

Let us clarify that our FD protocol is coined to work in any environment: either synchronous or asynchronous. The fact that the network behaves synchronously contributes even more to the satisfaction of the stability properties of the model. Indeed, a ‘correct’ node which behaves synchronously will satisfy the stability properties. If the network exhibits a hierarchy (e.g. WSNs, clustering), the better candidates to satisfy the responsiveness are going to be the CHs. But, this is not mandatory; our FD protocol is flexible enough to consider any other node (either a CH or not) for satisfying the responsiveness property.

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 to 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 $\diamond S^M$ 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 $p_j \in \text{susp}_i$, then $p_j \notin \text{mist}_i$ and similarly if $p_j \in \text{mist}_i$, then $p_j \notin \text{susp}_i$. 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 the 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 the predicate of line 12, which is never satisfied, and the 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 1, Algorithm 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, Algorithm 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, Algorithm 2). When $|X_i| \geq \alpha_i$, the contents of all response messages received are delivered to p_i via these three sets (line 11, Algorithm 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 α_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 that the FD properties hold for a network composed only of *pausing* nodes (Theorem 5.1). Afterward, we prove that they hold for the generic case (Theorem 5.2). Lemmas 5.3 and 5.4 are used to prove Theorem 5.1 and Lemmas 5.7 and 5.8 are used to prove Theorem 5.2.

5.1. Proof for the specific case of pausing nodes

LEMMA 5.1. *Let $p_i, p_j \in \text{KNOWN} \cap \text{STABLE}$. Consider that, at time t , p_i owns the last status $\langle p_x, ct \rangle$ about $p_x \in \text{KNOWN}$ 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 $\forall p_j \in \text{KNOWN} \cap \text{STABLE}$ 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 $\mathcal{J} : p_i = p_0, p_1, \dots, p_{k-1}, p_k = p_j$ of stable known nodes between p_i and p_j . Let us prove 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 $p_{k-1} \in \text{KNOWN} \cap \text{STABLE}$, 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 *SatP* (Property 1), the QUERY terminates and is received by at least $p_k \in \text{KNOWN} \cap \text{STABLE}$ 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 $\forall p_j \in \text{KNOWN} \cap \text{STABLE}$ and the lemma follows. \square

LEMMA 5.2. *Infinitely often, $\forall p_i \in \text{KNOWN}$, $\exists p_j \in \text{KNOWN} \cap \text{STABLE}$, $p_i \in \text{knownTo}_j$.*

Proof. From Observation 5 and *SatP* (Property 1), there is at least one $p_j \in \text{KNOWN} \cap \text{STABLE}$ in the neighborhood of p_i that receives the last QUERY–RESPONSE message sent from p_i ; thus, $p_i \in \text{knownTo}_j$ (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 first 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. Afterward, 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, $p_i \in \text{knownTo}_j$.

Situation (2). Assume that, due to an asynchronism, the last mistake raised by p_i arrives first 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 at p_j at line 20. This holds because, by assumption, $p_j \in \text{KNOWN} \cap \text{STABLE}$ is the neighbor that 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, $p_i \in \text{knownTo}_j$. This concludes the proof. \square

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

Proof. Let us consider that $\text{faulty}^t(p_f)$ holds. \square

Remark 1. Since $p_f \in \text{KNOWN}$, p_f has sent at least one QUERY message before it crashed at t and there is at least a process $p_i \in \text{KNOWN} \cap \text{STABLE}$ which has received this last QUERY (lines 20–26). Additionally, from Lemma 5.2, $p_f \in \text{knownTo}_i$. 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 $p_f \in \text{knownTo}_i$. 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 $\langle p_f, ct \rangle \in \text{mist}_i$, 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 $p_f \notin \text{susp}_j$ (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, $p_f \in \text{susp}_i$ is always true $\forall p_i \in \text{KNOWN} \cap \text{STABLE}$.

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

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 $p_i \in \text{knownTo}_k$ and $p_i \notin X_k$ (line 12). Note that $p_i \in \text{knownTo}_k$ 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 $\text{SRP}^t(p_i)$, 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, $p_i \notin \text{susp}_i$. 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 ($\langle p_i, ct \rangle \in \text{susp}_j$), generated before t ; (2) a mistake ($\langle p_i, ct \rangle \in \text{mist}_j$) 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 $p_j \in \text{KNOWN} \cap \text{STABLE}$ with $\langle p_i, ct \rangle \in \text{susp}_j$ (lines 9, 20) and call upon $\text{Update_State}()$, which will cause p_i to generate a new mistake with a greater tag ($\langle p_i, ct + 1 \rangle \in \text{mist}_i$) (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 $\forall p_j \in \text{KNOWN} \cap \text{STABLE}$. Then, eventually,

lines 36–37 are executed by p_j and $p_i \notin \text{susp}_j$. From Remark 1, no new information about p_i is generated and $p_i \notin \text{susp}_j$ is always true. Case (3) follows directly from Remark 1 and the lemma follows. \square

THEOREM 5.1. *Algorithm 1 implements the $\diamond S^M$ FD, 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. \square

5.2. 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 $\forall p_i \in \text{KNOWN}$ 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 $p_m \in \text{KNOWN} \cap \text{STABLE}$ is a moving node that 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 $p_i \in \text{KNOWN} \cap \text{STABLE}$ 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 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 $\text{Update_State}()$ (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 $p_m \in \text{KNOWN} \cap \text{STABLE}$ is a moving node that 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 SatP (Property 1), the QUERY terminates and there will be at least one node $p_i \in \text{KNOWN} \cap \text{STABLE}$ in the neighborhood of p_m , that receives the QUERY with the last status about p_x . Thus, by executing task T2, p_i calls upon $\text{Update_State}()$ (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 $p_j \in \text{KNOWN} \cap \text{STABLE}$ will include $\langle p_x, ct \rangle$ in its susp_j set (respectively, mist_j set) and the lemma follows. \square

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

Proof. From Observation 5 and SatP (Property 1), for every QUERY--RESPONSE, issued by p_i at task T1, there will be at least

one $p_j \in \text{KNOWN} \cap \text{STABLE}$ in the neighborhood of p_i that 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, $p_i \in \text{knownTo}_j$ and the lemma holds. \square

LEMMA 5.7. *Let $p_f \in \text{KNOWN} \cap \text{FAULTY}$. Eventually, $p_f \in \text{susp}_i, \forall p_i \in \text{KNOWN} \cap \text{STABLE}$ mobile nodes.*

Proof. Let us consider that $\text{faulty}'(p_f)$. 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 $p_i \in \text{KNOWN} \cap \text{STABLE}$ (either moving or pausing) such that $p_f \in \text{knownTo}_i$.

Case (1): Assume that $p_m \in \text{KNOWN} \cap \text{STABLE}$ is a moving node that does not yet have the last status about process p_f after t ; and, assume that, by Lemma 5.3, every pausing node $p_j \in \text{KNOWN} \cap \text{STABLE}$ 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 $p_m \in \text{KNOWN} \cap \text{STABLE}$ is a moving node that 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, $\langle p_f, ct \rangle \in \text{mist}_m$. 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, $\langle p_f, - \rangle \in \text{susp}_m$. 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 $p_i \in \text{KNOWN} \cap \text{STABLE}$.

Finally, for every case, from the same arguments of Lemma 5.3 (Remark 2) and considering Lemma 5.5, $p_f \in \text{susp}_i$ is always true $\forall p_i \in \text{KNOWN} \cap \text{STABLE}$. \square

LEMMA 5.8. *Let $p_i \in \text{KNOWN} \cap \text{STABLE}$. If $\text{SRP}'(p_i)$ holds for p_i at time t , then eventually $p_i \notin \text{susp}_j, \forall p_j \in \text{KNOWN} \cap \text{STABLE}$ mobile nodes.*

Proof. The lemma follows directly from Lemma 5.4 for all pausing nodes. Owing to the 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 $p_m \in \text{KNOWN} \cap \text{STABLE}$ 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 ($\langle p_i, ct \rangle \in \text{susp}_m$). 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, $p_i \in \text{knownTo}_m$ 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 ($\langle p_i, ct + 1 \rangle \in \text{mist}_i$) that, due to Lemma 5.5, is propagated along the network. Finally, $\langle p_i, ct + 1 \rangle \in \text{mist}_m$ and $\langle p_i, - \rangle \notin \text{susp}_m$. 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, $\forall p_j \in \text{KNOWN} \cap \text{STABLE}$ (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 . Owing to Lemma 5.5, after time t' , p_m succeeds to update its state with the last information about p_i . Following Lemmas 5.4 and 5.5, eventually $p_i \notin \text{susp}_m$ is always true. Thus, the lemma follows for $\forall p_j \in \text{KNOWN} \cap \text{STABLE}$. \square

THEOREM 5.2. *Algorithm 1 implements the $\diamond^S M$ FD, 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. \square

6. PERFORMANCE EVALUATION

In this section, we study and evaluate the behavior of our asynchronous FD. The performance experiments were conducted on top of the OMNeT++ discrete event simulator [33]. We assume two two-dimensional regions: the first one is a square of $600 \times 600 \text{ m}^2$ and the second one is a rectangle of $200 \times 1800 \text{ m}^2$. They thus have both the same surface and 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 five neighbors ($|N_i| \geq 5$) and that at most two neighbors may crash ($f_i = 2$). Therefore, α_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 min. 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 70 s starting at 10 s (10, 80, 150 s, etc.). The one-hop network delay δ is computed using the bandwidth (2 Mb/s) and the size of messages since the delay of propagation within a range is negligible. Then δ is less than 1 ms.¹ In our experiments, we assume that the MAC layer provides a local reliable broadcast.

¹This value is negligible in comparison with the delay between two queries.

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 $\Delta = 1$ s between lines 8 and 9 of 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 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

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 100 to 380 m, 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 Fig. 1. The

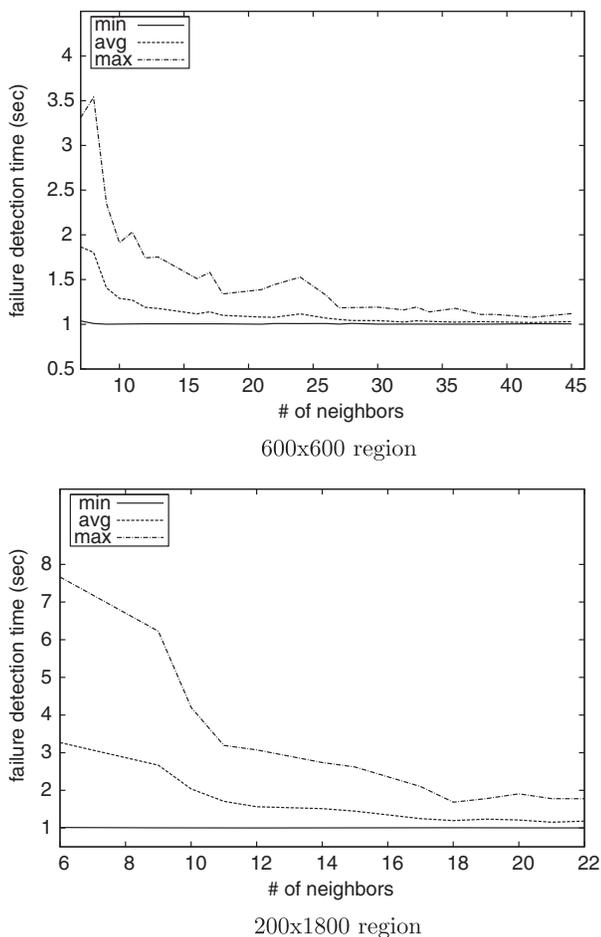


FIGURE 1. Failure detection time vs. number of neighbors.

maximum failure detection time characterizes, for each of the different number of neighbors, the time for all nodes to detect a failure (strong completeness).

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 1 and 10 nodes located at one boundary of the network move at a speed of 2 m/s. The range transmission r is set to 100 m for all p_i . When just one node moves, it starts moving at time 20 s while when 10 nodes move, the first one starts moving at 100 s and at every 5 s 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 α_j 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 (respectively, Fig. 3) shows the number of false suspicions between the moment that just one node (respectively, 10 nodes) starts moving at 20 s (respectively, 100 s and every 5 s) 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 has a change of neighborhood. These suspicions are quickly corrected by the moving node itself as soon as it receives the suspected information from its old neighbors, which usually remain close to it (in most cases they are at one hop).

Figure 4a and b, respectively, shows the distribution of mistake duration, i.e. how long on an 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 <0.01 s when the suspected node is close to the node that generates the suspicion and in 4s at the maximum, otherwise.

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

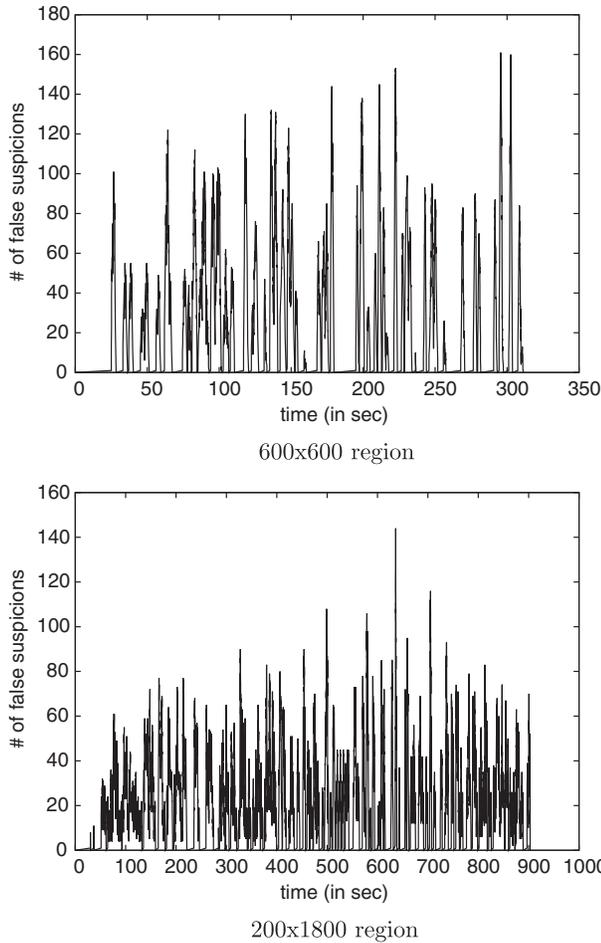


FIGURE 2. Total number of false suspicions when one node moves.

- (i) 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.
- (ii) Our FD is highly reactive in correcting false suspicions when a 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 that limit the propagation of wrong suspected information.

7. RELATED WORK

7.1. 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 FD based on a logical ring configuration of processes. Thus,

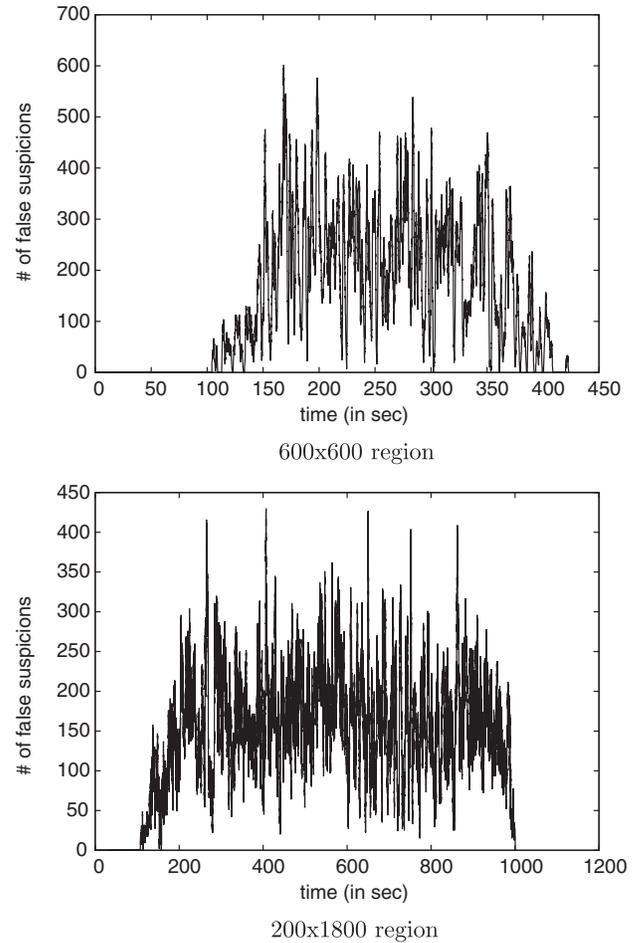


FIGURE 3. Total number of false suspicions when 10 nodes move.

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 of 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], the authors identify important problems on the design of scalable FDs for Grid architectures.

7.2. The heartbeat approach

Aguilera *et al.* [9] propose the heartbeat FD which does not assume a network of full connectivity and tolerates message losses. Tucci-Piergiovanni and Baldoni *et al.* [10] implement a heartbeat FD for the infinite arrival model and show how to use it to implement an FD of the Ω class. The Ω class ensures that eventually each correct process is going to trust

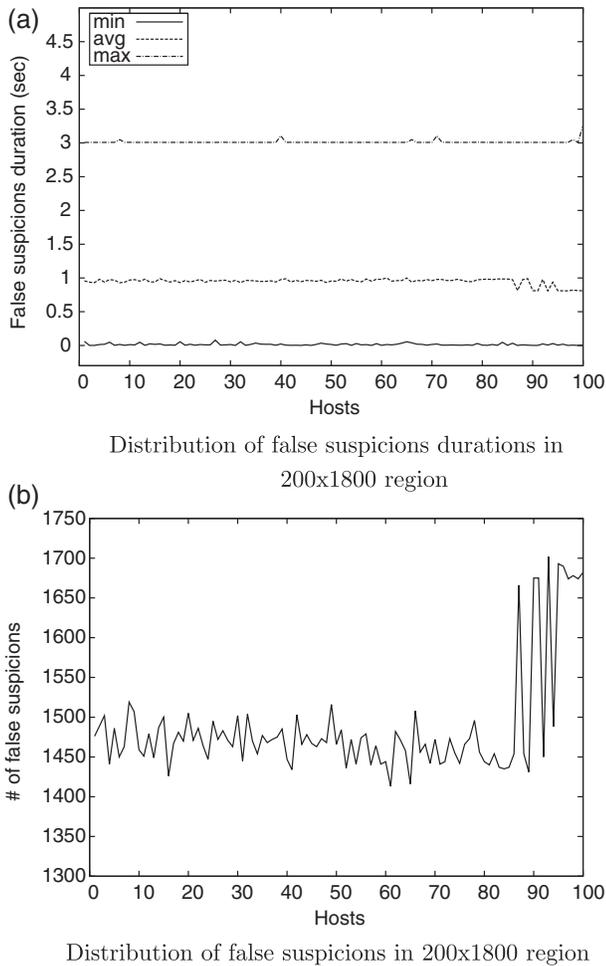


FIGURE 4. Distribution of mistakes when 10 nodes move.

the same correct process, considered as the *leader*. The solution assumes fair-lossy channels, but for a synchronous environment. Hutle [38] 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 tolerates the mobility of nodes.

Few implementations of unreliable FDs found in the literature focus on wireless mobile networks [11–13]. 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 Cao *et al.* [14] in order to implement a *leader* FD of the Ω class. Nonetheless, it does not tolerate node mobility.

7.3. Probabilistic approach for wireless *ad hoc* networks

Friedman and Tcherny [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 item of 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 an FD of the class $\diamond P$, which provides probabilistic guarantees for the accuracy and completeness properties; moreover, it does not consider mobility.

7.4. 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 restraint 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 the mobility of nodes across the network and is able to correct possible mistakes. He advocates the use of this local detection as an appropriate abstraction to deal with the lack of mobility and resources in WSNs. Unlike our solution that allows the implementation of a $\diamond S^M$ FD, this work implements an eventually perfect *local* FD of the class $\diamond P$, i.e. it provides strong completeness and eventual strong accuracy but with regard to a node's neighborhood.

7.5. Time-free approach for omega

Cao *et al.* [14] propose a time-free query-based implementation of a deterministic *leader* FD. It considers an infra-structured mobile network composed of MHs and MSSs (see Section 4.2). An MH is considered stable if, once it has 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 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

TABLE 1. Comparison of FDs.

Protocol	Node type	Number of nodes	Number of failures	Network connectivity	Failure model
Friedman and Tcherny [11]	Mobile	Known	Arbitrary	Connected graph	Crash
Tai <i>et al.</i> [12]	Static	Known	Arbitrary	Static connected graph of CHs	Message omission
Sridhar [13]	Mobile	Known	Arbitrary	Connected graph	Crash
Cao <i>et al.</i> [14]	Static and mobile	Known (static) Unknown (mobile)	f (fixed)	Complete graph of static nodes	Message omission
Our protocol	Mobile	Unknown	$N_i/2$	Connected graph	Crash Reliable channels Crash Message omission
Protocol	Detection strategy	Timer based	Membership property	Local detection	FD class
Friedman and Tcherny [11]	Heartbeat	Yes	Yes	No	
Tai <i>et al.</i> [12]	Heartbeat	Yes	Yes (CH)	Yes	Probabilistic $\diamond P$
Sridhar [13]	Generic	Yes	No (mobile nodes)	Yes	Local $\diamond P$
Cao <i>et al.</i> [14]	Query-response	No	Yes (static nodes)	No	Ω leader
Our protocol	Query-response	No	No (mobile nodes)	Yes	$\diamond S^M$

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.

7.6. 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 and Tixeuil [15], who present a solution for the fault-tolerant consensus in a dynamic system of unknown participants, with minimal synchrony assumptions (i.e. an FD of the class $\diamond S$).

Another very good application for our FD protocol is the consensus proposed in [39]. It makes use of an abstraction, called the *eventual clusterer oracle* (or $\diamond C$), able to detect failures of MHs and to elect CHs in a hierarchical net. The $\diamond C$ oracle has the same completeness and accuracy properties of the $\diamond S$ FD. Thus, an implementation of $\diamond C$ can make use of $\diamond S$. In the context of MANETs, our FD protocol of class $\diamond S^M$ will directly benefit $\diamond C$.

7.7. 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 and (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 FDs in mobile wireless networks, such as WMNs or WSNs and provides the specification of a new class of FDs for this context: the $\diamond S^M$ class (eventually strong FD with unknown membership). It presents an algorithm able to implement a time-free $\diamond S^M$ 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.

FUNDING

The work of F.G. is supported by grants from CAPES-Brazil and Paris City Hall, France.

REFERENCES

- [1] Conti, M. and Giordano, S. (2007) Multihop ad hoc networking: the theory. *IEEE Commun. Mag.*, **45**, 78–86.
- [2] Chandra, T. and Toueg, S. (1996) Unreliable failure detectors for reliable distributed systems. *J. ACM*, **43**, 225–267.
- [3] Akyildiz, I.F., Wang, X. and Wang, W. (2005) Wireless mesh networks: a survey. *Comput. Netw.*, **47**, 445–487.
- [4] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y. and Cayirci, E. (2002) Wireless sensor networks: a survey. *Comput. Netw.*, **38**, 393–422.
- [5] Larrea, M., Fernández, A. and Arévalo, S. (2000) Optimal Implementation of the Weakest Failure Detector for Solving Consensus. *Proc. 19th Annual ACM Symp. Principles of Distributed Computing*, Portland, Oregon, USA, pp. 334–334.
- [6] Sotoma, I. and Madeira, E. (2001) Adaptation—Algorithms to Adaptive Fault Monitoring and their Implementation on Corba. *Proc. IEEE 3rd Int. Symp. Distributed Objects and Applications*, New Orleans, LA, USA, pp. 219–228.
- [7] Devianov, B. and Toueg, S. (2000) Failure Detector Service for Dependable Computing. *Proc. 1st Int. Conf. Dependable Systems and Networks*, New York, NY, USA, pp. 14–15.
- [8] Mostefaoui, A., Mourgaya, E. and Raynal, M. (2003) Asynchronous Implementation of Failure Detectors. *Proc. Int. Conf. Dependable Systems and Networks*, San Francisco, CA, USA, pp. 351–360.
- [9] Aguilera, M.K., Chen, W. and Toueg, S. (1997) Heartbeat: A Timeout-Free Failure Detector for Quiescent Reliable Communication. *Proc. 11th Int. Workshop on Distributed Algorithms*, Saarbrücken, Germany, 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 Conf. (EDCC), 2010 European*, Valencia, Spain, pp. 127–134.
- [11] Friedman, R. and Tcherny, G. (2009) Evaluating failure detection in mobile ad-hoc networks. *Int. J. Pervasive Comput. Commun.*, **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. Dependable Systems and Networks*, pp. 805–814.
- [13] Sridhar, N. (2006) Decentralized Local Failure Detection in Dynamic Distributed Systems. *The 25th IEEE Symp. Reliable Distributed Systems*, Leeds, UK, 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. Dependable Computing*, Melbourne, Victoria, Australia, pp. 123–130.
- [15] Greve, F. and Tixeuil, S. (2007) Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks. *Proc. Int. Conf. Dependable Systems and Networks*, Edinburgh, UK, pp. 82–91.
- [16] Sens, P., Arantes, L., Bouillaguet, M., Simon, V. and Greve, F. (2008) An Unreliable Failure Detector for Unknown and Mobile Networks. *OPODIS*, Luxor, Egypt, pp. 555–559.
- [17] Greve, F., Sens, P., Arantes, L. and Simon, V. (2011) A Failure Detector for Wireless Networks with Unknown Membership. *Euro-Par Conf.*, Lecture Notes in Computer Science 6853, Bordeaux, France, 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*, Rome, Italy.
- [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., Arora, A. and Ten-Hwang, L. (2002) Reliable Mac Layer Multicast in IEEE 802.11 Wireless Networks. *Proc. Int. Conf. Parallel Processing*, Vancouver, BC, Canada, August, pp. 527–536.
- [22] Koo, C.-Y. (2004) Broadcast in Radio Networks Tolerating Byzantine Adversarial Behavior. *23th ACM Symp. Principles of Distributed Computing*, St. John’s, Newfoundland, Canada, 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*, Portland, Oregon.
- [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.*, Piran, Slovenia, 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. Reliable Distributed Systems*, Orlando, FL, USA, pp. 109–118.
- [27] Bhandari, V. and Vaidya, N.H. (2010) Reliable broadcast in radio networks with locally bounded failures. *IEEE Trans. Parallel Distrib. Syst.*, **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. Principles of Distributed Computing*, Las Vegas, NV, USA, 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. *Wirel. Commun. Mob. Comput.*, 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. 20th Annual ACM Symp. Principles of Distributed Computing*, Newport, Rhode Island, USA, 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. Int. Conf. Dependable Systems and Networks*, Washington, DC, USA, June, pp. 354–363. Bethesda, MD, USA.
- [37] Hayashibara, N., Cherif, A. and Katayama, T. (2002) Failure Detectors for Large-Scale Distributed Systems. *21st Symp. Reliable Distributed Systems (SRDS 2002)*, Osaka, Japan, pp. 404–409.
- [38] Hutle, M. (2004) An Efficient Failure Detector for Sparsely Connected Networks. *Proc. IASTED Int. Conf. Parallel and Distributed Computing and Networks*, Innsbruck, Austria, pp. 369–374.
- [39] Wu, W., Cao, J. and Raynal, M. (2009) Eventual clusterer: a modular approach to designing hierarchical consensus protocols in manets. *IEEE Trans. Parallel Distrib. Syst.*, **20**, 753–765.