A Failure Detector for Wireless Networks 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 - Brazil
[2] LIP6, University of Paris 6, CNRS,
INRIA, 4 - Place Jussieu, 75005, Paris, France

Abstract. The distributed computing scenario is rapidly evolving for integrating self-organizing and dynamic wireless networks. Unreliable failure detectors are classical mechanisms which provide information about process failures and can help systems to cope with the high dynamism of these networks. A number of failure detection algorithms has been proposed so far; nonetheless, most of them assume a global knowledge about the membership as well as a fully communication connectivity; additionally, they are timer-based, requiring that eventually some bound on the message transmission will 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 MANETs (mobile ad-hoc networks) [1]. Nonetheless, the issue of designing reliable services which can cope with the high dynamism of these systems is a challenge. Failure detector is a fundamental service, able to help in the development of fault-tolerant distributed systems. *Unreliable failure detectors*, namely FD, can informally be seen as a per process oracle, which periodically provides a list of processes suspected of having crashed [2]. In this paper, we are interested in the class of eventually strong FDs, denoted $\diamondsuit S$. 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) and

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

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6853, Part II, pp. 27–38, 2011.

consensus is as the heart of important middleware, e.g., group communication services, transactions and replication servers.

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

This paper proposes a FD algorithm that implements the class $\diamondsuit S^M$ of failure detectors. This class adapts the properties of the $\diamondsuit S$ class to a dynamic system with an unknown membership. It is suitable for wireless mobile networks and has the following innovative features that allow for scalability and adaptability: (i) it is conceived for a network whose membership is unknown and whose communication graph is not complete; (ii) it tolerates node mobility, beyond arbitrary joins and leaves; (iii) the failure detection uses local information (for the membership of the neighborhood), instead of traditional global information, such as n (the total number of nodes) and f (the maximum number of faults); (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. As far as we are aware of, this is the first time-free FD algorithm for networks with unknown membership that tolerates mobility of nodes.

1.1 Related Work

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 [3]. As they usually consider a fully connected set of known nodes, these implementations are not adequate for dynamic environments. Furthermore, they are usually timer-based, assuming that eventually some bound of the transmission will permanently hold. Such an assumption is not suitable for dynamic environments where communication delays between two nodes can vary due to mobility of nodes. In [4], Mostefaoui *et al.* have proposed an asynchronous implementation of FDs which is time-free. It is based on an exchange of messages which just uses the values of f and n. However, their computation model consists of a set of fully connected initially known nodes. Some works [5–7] focus on the heartbeat FD for sparsely connected networks with unknown membership. The heartbeat FD is a special class of FD which is time-free and is able to implement quiescent reliable communication. But, instead of lists of suspects, it outputs a vector of unbounded counters; if a process crashes, its counter eventually stops increasing. It is worth remarking that none of these works tolerate mobility of nodes.

Few implementations of unreliable FDs focus on wireless mobile networks [8–10]. The fundamental difference between these works and ours is the fact that all of them are timer-based. Friedman and Tcharny [8] propose a simple gossiping protocol which exploits the natural broadcast range of wireless networks to delimit the local member-

ship of a node in a mobile network. Contrarily to our approach, this work assumes a known number of nodes and provides probabilistic guarantees for the FD properties. Tai *et al.* [9] 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. Unlike our solution, this work considers a cluster-based communication architecture and provides probabilistic guarantees for the accuracy and completeness properties; moreover, it does not consider mobility. Sridhar [10] 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. While our approach 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 perfect failure detection, but with regard to a node's neighborhood. As soon as we are aware of, the only work to follow a time-free detection strategy has been proposed by [11] in order to implement a *leader* FD of the Ω class. This class ensures that, each process will be provided by a unique leader, elected among the set of correct processes, in spite of crashes. Differently from ours, this work is for a specific infra-structured network composed of mobile and static nodes. 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.*[12], who present a solution for the fault-tolerant consensus in a network of unknown participants with minimal synchrony assumptions.

The rest of the paper is organized as follows. Section 2 defines the model and specifies the $\diamondsuit S^M$ FD class. Section 3 identifies assumptions to implement those FDs. Section 4 presents a time-free FD of the $\diamondsuit S^M$ class. Section 5 concludes the paper. In an extended report [13], one can find complete correctness proofs, a thorough related work section and performance experiments showing that the proposed FD exhibits a good reactivity to detect failures and revoke false suspicions, even in presence of mobility.

2 Model and Problem Definition

The wireless mobile network is a dynamic system composed of infinitely many processes; but each run consists of a finite set Π of $n > 1$ mobile nodes, namely, $\Pi = \{p_1, \ldots, p_n\}$. Contrarily to a static network, the membership is unknown, thus processes are not aware about Π and n, because, moreover, these values can vary from run to run; this coincides with the *finite arrival model* [14]. This model is suitable for long-lived or unmanaged applications, as for example, sensor networks deployed to support crises management or help on dealing with natural disasters. There is one process per node; each process knows its own identity, but it does not necessarily knows the identities of the others. Nonetheless, nodes communicate by sending and receiving messages via a packet radio network and may make use of the broadcast facility of this communication medium to know one another. There are no assumptions on the relative speed of processes or on message transfer delays, thus the system is *asynchronous*; there is no global clock, but to simplify the presentation, we take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers. A process may fail by *crashing*, *i.e.*, by prematurely or by deliberately halting (switched off); a crashed process does not recover.

The network is represented by a communication graph $G = (V, E)$ in which $V = \Pi$ represents the set of mobile nodes and E represents the set of logical links. The topology of G is dynamic due to arbitrary joins, leaves, crashes and moves. A bidirectional link between nodes p_i and p_j means that p_i is within the wireless transmission range of p_j and vice-versa. If this assumption appears to be inappropriate for a mobile environment, one can use the strategy proposed in [15] for allowing a protocol originally designed for bidirectional links to work with unidirectional links. Let R_i be the transmission range of p_i, then all the nodes that are at distance at most R_i from p_i in the network are considered 1-hop *neighbors*, belonging to the same *neighborhood*. We denote N_i to be the set of 1-hop *neighbors* from p_i; thus, $(p_i, p_j) \in E$ *iff* $(p_i, p_j) \in N_i$. Local broadcast between 1-hop neighbors is *fair-lossy*. This means that messages may be lost, but, if p_i broadcasts m to processes in its neighborhood an infinite number of times, then every p_j in the neighborhood receives m from p_i an infinite number of times, or p_j is faulty. This condition is attained if the MAC layer of the underlying wireless network provides a protocol that reliably delivers broadcast data, even in presence of unpredictable behaviors, such as fading, collisions, and interference; solutions in this sense have been proposed in [16–18]. Nodes in Π may be mobile and they can keep continuously *moving* and *pausing* in the system. When a node p_m moves, its neighborhood may change. We consider a *passive mobility* model, i.e., the node that is moving does not know that it is moving. Hence, the mobile node p_m cannot notify its neighbors about its moving. Then, for the viewpoint of a neighbor, it is not possible to distinguish between a moving, a leave or a crash of p_m. During the neighborhood changing, p_m keeps its state, that is, the values of its variables.

2.1 Stability Assumptions

In order to implement unreliable failure detectors with an unknown membership, processes should interact with some others to be known. If there is some process in the system such that the rest of processes have no knowledge whatsoever of its identity, there is no algorithm that implements a failure detector with weak completeness, even if links are reliable and the system is synchronous [19]. In this sense, the characterization of the *actual membership* of the system, that is, the set of processes which might be considered for the computation is of utmost importance for our study. We consider then that after have joined the system for some point in time, a mobile process p_i must communicate somehow with the others in order to be known. Afterwards, if p_i leaves, it can re-enter the system with a new identity, thus, it is considered as a new process. Processes may join and leave the system as they wish, but the number of re-entries is bounded, due to the finite arrival assumption. One important aspect concerns the time period and conditions in which processes are connected to the system. During unstable periods, certain situations, as for example, connections for very short periods, the rapid movement of nodes, or numerous joins or leaves along the execution (characterizing a churn) could block the application and prevent any useful computation. Thus, the system should present some stability conditions that when satisfied for longtime enough will be sufficient for the computation to progress and terminate.

Definition 1. ***Membership*** *Let $t, t' \in \mathcal{T}$. Let $UP(t) \subset \Pi$ be the set of mobile processes that are in the system at time t, that is, after have joined the system before t, they neither*

leave it nor crash before t. Let p_i, p_j be mobile nodes. Let the $known_j$ set denotes the partial knowledge of p_j about the system's membership. The membership of the system is the KNOWN *set.*

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

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

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

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

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

This connectivity assumption states that, in spite of changes in the topology of G, from some point in time t, the set of known stables forms a *strongly connected component* in G. This condition is frequently present in the classical model of static networks and is indeed mandatory to ensure dissemination of messages to all stable processes and thus to ensure the global properties of the failure detector [2, 19–21].

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

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 stable and faulty.

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

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

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

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

None of the failure detector classes can be implemented in a purely asynchronous system [2]. Indeed, while completeness can be realized by using "I am alive" messages and timeouts, accuracy cannot be safely implemented for all system executions. Thus, some additional assumptions on the underlying system should be made in order to implement them. With this aim, two orthogonal approaches can be distinguished: the timer-based and the time-free failure detection [22]. The timer-based model is the traditional approach and supposes that channels in the system are eventually timely; this means that, for every execution, there are bounds on process speeds and on message transmission delays. However, these bounds are not known and they hold only after some unknown time [2]. An alternative approach suggested by [4] and developed so far by [11, 20] considers that the system satisfies a message exchange pattern on the execution of a *query-based* communication and is time-free. While the timer-based approach imposes a constraint on the physical time (to satisfy message transfer delays), the time-free approach imposes a constraint on the logical time (to satisfy a message delivery order). These approaches are orthogonal and cannot be compared, but, they can be combined at the link level in order to implement hybrid protocols with combined assumptions [22].

3.1 Stable Query-Response Communication Mechanism

Our failure detector is time-free and based on a local QUERY-RESPONSE communication mechanism [20] adapted to a network with unknown membership. At each *query-response* round, a node systematically broadcasts a QUERY message to the nodes in its neighborhood until it possibly crashes or leaves the system. The time between two consecutive queries is finite but arbitrary. Each couple of QUERY-RESPONSE messages are uniquely identified in the system. A process p_i launches the primitive by sending a QUERY(m) with a message m. When a process p_j delivers this query, it updates its local state and systematically answers by sending back a RESPONSE() to p_i. Then, when p_i has received at least α_i responses from different processes, including a stable one, the current QUERY-RESPONSE *terminates*. Without loss of generality, the response for p_i itself is among the α_i responses. An implementation of a QUERY-RESPONSE communication over fair-lossy local channels can be done by the repeated broadcast of the query by the sender p_i until it has received at least α_i responses from its neighbors. Formally, the stable QUERY–RESPONSE primitive has the following properties:

(i) QR-Validity: If a QUERY(m) is delivered by process p_j, it has been sent by p_i;
(ii) QR-Uniformity: A QUERY(m) is delivered at most once by a process;
(iii) QR-Stable-Termination: If a process p_i is not faulty (it does not crash nor leave the system) while it is issuing a query, that query generates at least α_i responses.

The value associated to α_i should correspond to the expected number of processes with whom p_i can communicate, in spite of moves and faults. Since communication is local, α_i is a local parameter and can be defined as the value of the neighborhood density of p_i (i.e., $|N_i|$) minus the maximum number of faulty processes in its neighborhood; let f_i be this number; that is, $\alpha_i = |N_i| - f_i$. This local choice for α_i changes from previous works which consider a global value either proportional to the number

of correct processes [4] or the number of stable processes [20] or the global number of faults [11]. Moreover, it follows recent works on fault tolerant communication in radio networks which propose a "local" fault model, instead of a "global" fault model, as an adequate strategy to deal with the dynamism and unreliability of wireless channels in spite of failures [17]. To reliably delivery data in spite of crashes, the maximum number of local failures should be $f_i < |N_i|/2$ [23]. From Assumption 1 about the network connectivity over time, at least one stable known node p_j will receive the QUERY and send a RESPONSE to p_i, since moreover channels are fair-lossy. Thus, the following property holds:

Property 1. **Stable Termination Property** ($\mathcal{S}at\mathcal{P}$). Let p_i be a node which issues a QUERY. Let X_i be the set of processes that issued a RESPONSE to that query. Thus, $\exists p_j \in X_i, p_j \in \text{KNOWN} \cap \text{STABLE}, p_j \neq p_i$.

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

3.2 Behavioral Properties

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

Property 2. **Mobility Property** ($\mathcal{M}obi\mathcal{P}$). Let $t', t \in \mathcal{T}, t' < t$. Let p_i be a node. Let t' be the time after which p_i has changed of neighborhood. Let SQ_i^t be the set of processes from which p_i has received a QUERY message after t' and before or at t. Process p_i satisfies $\mathcal{M}obi\mathcal{P}$ at time t if:

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

Instead of synchrony assumptions, to ensure the accuracy of the detection, the time-free model establishes conditions on the logical time the messages are delivered by processes. These are unified in the *stabilized responsiveness property*, namely $\mathcal{SRP}$. Thus, $\mathcal{SRP}(p_i)$ states that eventually, for any process p_j (which had received a response from $p_i$in the past), the set of responses received by p_j to its last QUERY always includes a response from p_i, that is, the response of p_i is always a winning response [22].

Property 3. **Stabilized Responsiveness Property** ($\mathcal{SRP}$). Let $t'', t', t \in \mathcal{T}$. Let p_i be a stable known node. Let $rec\_from_j^{t'}$ ($rec\_from_j^{t''}$) be the set of processes from which p_j has received responses to its last QUERY that terminated at or before t'(t''). Process p_i satisfies $\mathcal{SRP}$ at time t if:

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

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

In order to implement a $\diamondsuit S^M$ FD, the following behaviors should be satisfied:
1) $\forall p_i \in$ KNOWN : $\mathcal{MobiP}^t(p_i)$ holds after p_i moves and changes of neighborhood;
2) $\exists p_i \in$ KNOWN $\cap$ STABLE : $\mathcal{SRP}^t(p_i)$ eventually holds.

A discussion about how to satisfy in practice the properties and assumptions of the model is done in Section 4.2 after the protocol's explanation.

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

4.1 Algorithm Description

Algorithm 1 describes our protocol for implementing a FD of class $\diamondsuit S^M$ for a network of KNOWN mobile nodes that satisfies the model stated in Sections 2 and 3.

Notations. We use the following notations:

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

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

Task $T1$: Generating suspicions. This task is made up of an infinite loop. At each round, a QUERY($susp_i$, $mist_i$) message is sent to all nodes of p_i's neighborhood (line 5). Node p_i waits for at least α_i responses, which includes p_i's own response (line 6). Then, p_i detects new suspicions (lines 8-13). It starts suspecting each node p_j, not previously suspected ($p_j \notin susp_i$), which it knows ($p_j \in known_i$), but from which

it does not receive a RESPONSE to its last QUERY. If a previous mistake information related to this new suspected node exists in the mistake set $mist_i$, it is removed from it (line 11) and the suspicion information is then included in $susp_i$ with a tag which is greater than the previous mistake tag (line 10). If p_j is not in the $mist$ set (i.e., it is the first time p_j is suspected), p_i suspected information is tagged with 0 (line 13).

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

```
1  init:
2   susp_i ← ∅; mist_i ← ∅  ;  known_i ← ∅
3  Task T1:
4  Repeat forever
5  |  broadcast QUERY(susp_i, mist_i)
6  |  wait until RESPONSE received from ≥ α_i processes
7  |  rec_from_i ← all p_j, a RESPONSE is received in line 6
8  |  For all p_j ∈ known_i \ rec_from_i | ⟨p_j, −⟩ ∉ susp_i do
9  |  |  If  ⟨p_j, ct⟩ ∈ mist_i
10 |  |  |  Add(susp_i, ⟨p_j, ct + 1⟩)
11 |  |  |  mist_i = mist_i \ {⟨p_j, −⟩}
12 |  |  Else
13 |  |  |  Add(susp_i, ⟨p_j, 0⟩)
14 End repeat
15
16 Task T2:
17 Upon reception of QUERY (susp_j, mist_j) from p_j do
18 known_i ← known_i ∪ {p_j}
19 For  all ⟨p_x, ct_x⟩ ∈ susp_j do
20 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
21 |  If p_x = p_i
22 |  |  Add(mist_i, ⟨p_i, ct_x + 1⟩)
23 |  Else
24 |  |  Add(susp_i, ⟨p_x, ct_x⟩)
25 |  |  mist_i = mist_i \ {⟨p_x, −⟩}
26 For  all ⟨p_x, ct_x⟩ ∈ mist_j do
27 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
28 |  Add(mist_i, ⟨p_x, ct_x⟩)
29 |  susp_i = susp_i \ {⟨p_x, −⟩}
30 |  If (p_x ≠ p_j)
31 |  |  known_i ← known_i \ {p_x}
32 send RESPONSE to p_j
```

Task $T2$: Propagating suspicions and mistakes. This task allows a node to handle the reception of a QUERY message. A QUERY message contains the information about suspected nodes and mistakes kept by the sending node. However, based on the tag associated to each piece of information, the receiving node only takes into account the ones that are more recent than those it already knows or the ones that it does not know at all. The two loops of task $T2$ respectively handle the information received about suspected nodes (lines 19–25) and about mistaken nodes (lines 26–31). Thus, for each

node p_x included in the suspected (respectively, mistake) set of the QUERY message, p_i includes the node p_x in its $susp_i$ (respectively, $mist_i$) set only if the following condition is satisfied: p_i received a more recent information about p_x status (failed or mistaken) than the one it has in its $susp_i$ and $mist_i$ sets. Furthermore, in the first loop of task $T2$, a new mistake is detected if the receiving node p_i is included in the suspected set of the QUERY message (line 21) with a greater tag. At the end of the task (line 32), p_i sends to the querying node a RESPONSE message.

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

In order to avoid a "ping-pong" effect between information about suspicions and mistakes, lines 30–31 allow the updating of the $known$ sets of both the node p_m and of those nodes that belong to the original destination of p_m. Then, for each mistake $\langle p_x, ct_x \rangle$ received from p_j, such that p_i keeps an old information about p_x, p_i verifies whether p_x is the sending node p_j (line 30). If they are different, p_x should belong to a remote neighborhood, because otherwise, p_i would have received the mistake by p_x itself. Notice that only the node can generate a new mistake about itself (line 21). Thus, p_x is removed from the $known_i$ set (line 31). Notice, however, that this condition is not sufficient to detect the mobility, because p_x can be a neighbor of p_i and due to an asynchronous race, the QUERY sent by p_x with the mistake has not yet arrived at p_i. In fact, the propagated mistake sent by p_j has arrived at p_i firstly. If that is the case, p_x has been unduly removed from $known_i$. Fortunately, since local broadcast is fair-lossy, the QUERY from p_x is going to eventually arrive at p_i, if p_i is stable, and, as soon as the QUERY arrives, p_i will once again add p_x to $know_i$ (lines 17–18).

4.2 Practical Issues

The *stable termination* of the QUERY-RESPONSE primitive and the $\mathcal{M}obi\mathcal{P}$ property may be satisfied if the time of pause, between changes in direction and/or speed, is defined to be greater than the time to transmit the QUERY and receive the RESPONSE messages. This condition is attained when for example, the most widely used Random Waypoint Mobility Model [24] is considered. In practice, the value of α_i (the number of responses that a process p_i should wait in order to implement a QUERY-RESPONSE) relates not only with the application density and the expected number of local faults, but also with the type of network considered (either WMN, WSN, etc.) and the current topology of the network during execution. Thus, it can be defined on the fly, based on the current behavior of the network. Wireless Mesh Network (WMN), Wireless Sensor

Network (WSN), and infra-structured mobile networks [11, 25] are a good examples of platforms who would satisfy the assumptions of our model, specially the $\mathcal{SRP}$. In a WMN, the nodes move around a fixed set of nodes (the core of the network) and each mobile node eventually connects to a fix node. A WSN is composed of stationary nodes and can be organized in clusters, so that communication overhead can be reduced; one node in each cluster is designated the cluster head (CH) and the other nodes, cluster members (CMs). Communication inter-clusters is always routed through the respective CHs which act as gateway nodes and are responsible for maintaining the connectivity among neighboring CHs. An infra-structured mobile network is composed of mobile hosts (MH) and mobile support stations (MSS). A MH is connected to a MSS if it is located in its transmission range and two MHs can only communicate through MSSs, but, due to mobility, an MH can leave and enter the area covered by other MSSs. The system is composed of N MSSs but infinitely many MHs. However, in each run the protocol has only finitely many MHs. There are some works to implement a leader oracle [11] and to solve consensus in this type of network [25].

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

5 Conclusion

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

References

1. Conti, M., Giordano, S.: Multihop ad hoc networking: The theory. IEEE Communications Magazine 45(4), 78–86 (2007)
2. Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
3. Devianov, B., Toueg, S.: Failure detector service for dependable computing. In: Proc. of the 1st Int. Conf. on Dependable Systems and Networks, pp. 14–15 (2000)
4. Mostefaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: Proc. of Int. Conf. on Dependable Systems and Networks (2003)
5. Aguilera, M.K., Chen, W., Toueg, S.: Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: Proc. of the 11th International Workshop on Distributed Algorithms, pp. 126–140 (1997)

6. Hutle, M.: An efficient failure detector for sparsely connected networks. In: Proc. of the IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 369–374 (2004)
7. Tucci-Piergiovanni, S., Baldoni, R.: Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In: Dependable Computing Conference (EDCC), pp. 127–134 (2010)
8. Friedman, R., Tcharny, G.: Evaluating failure detection in mobile ad-hoc networks. Int. Journal of Wireless and Mobile Computing 1(8) (2005)
9. Tai, A., Tso, K., Sanders, W.: Cluster-based failure detection service for large-scale ad hoc wireless network applications. In: Int. Conf. on Dependable Systems and Networks, pp. 805–814 (2004)
10. Sridhar, N.: Decentralized local failure detection in dynamic distributed systems. In: The 25th IEEE Symp. on Reliable Distributed Systems, pp. 143–154 (2006)
11. Cao, J., Raynal, M., Travers, C., Wu, W.: The eventual leadership in dynamic mobile networking environments. In: 13th Pacific Rim Intern. Symp. on Dependable Computing, pp. 123–130 (2007)
12. Greve, F., Tixeuil, S.: Knowledge conectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In: Int. Conf. on Dependable Systems and Networks, pp. 82–91 (2007)
13. Sens, P., Arantes, L., Bouillaguet, M., Simon, V., Greve, F.: Asynchronous implementation of failure detectors with partial connectivity and unknown participants. Technical Report, RR6088, INRIA - France, `http://hal.inria.fr/inria-00122517/fr/`
14. Aguilera, M.K.: A pleasant stroll through the land of infinitely many creatures. SIGACT News 35(2), 36–59 (2004)
15. Ramasubramanian, V., Chandra, R., Mossé, D.: Providing a bidirectional abstraction for unidirectional adhoc networks. In: Proc. of the 21st IEEE International Conference on Computer Communications (2002)
16. Min-Te, S., Lifei, H., Arora, A.A., Ten-Hwang, L.: Reliable mac layer multicast in ieee 802.11 wireless networks. In: Proc. of the Intern, August 2002, pp. 527–536 (2002)
17. Koo, C.Y.: Broadcast in radio networks tolerating byzantine adversarial behavior. In: 23th Symp. on Principles of Distributed Computing, pp. 275–282 (2004)
18. Bhandari, V., Vaidya, N.H.: Reliable local broadcast in a wireless network prone to byzantine failures. In: The 4th Int. Work. on Foundations of Mobile Computing (2007)
19. Jiménez, E., Arévalo, S., Fernández, A.: Implementing unreliable failure detectors with unknown membership. Inf. Process. Lett. 100(2), 60–63 (2006)
20. Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., Abbadi, A.: From static distributed systems to dynamic systems. In: Proc. of the 24th IEEE Symposium on Reliable Distributed Systems, pp. 109–118 (2005)
21. Bhandari, V., Vaidya, N.H.: Reliable broadcast in radio networks with locally bounded failures. IEEE Trans. on Parallel and Distributed Systems 21, 801–811 (2010)
22. Mostefaoui, A., Raynal, M., Travers, C.: Time-free and timer-based assumptions can be combined to obtain eventual leadership. IEEE Trans. Parallel Distrib. Syst. 17(7), 656–666 (2006)
23. Bhandari, V., Vaidya, N.H.: On reliable broadcast in a radio network. In: 24th Symp. on Principles of Distributed Computing, pp. 138–147. ACM, New York (2005)
24. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications & Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications 2, 483–502 (2002)
25. Wu, W., Cao, J., Yang, J., Raynal, M.: Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. IEEE Trans. Comput. 56(8), 1055–1070 (2007)