A Time-Free Byzantine Failure Detector for Dynamic Networks

Fabíola Greve, Murilo Santos de Lima
Federal University of Bahia,
Computer Science Department, Salvador, Brazil
E-mail: [fabiola,murilolima]@dcc.ufba.br

Luciana Arantes, Pierre Sens
Université Pierre et Marie Curie,
CNRS - INRIA - REGAL, Paris, France
E-mail: [luciana.arantes, pierre.sens]@lip6.fr

***Abstract*—Modern distributed systems deployed over wireless ad-hoc networks are inherently dynamic and the issue of designing dependable services which can cope with the high dynamics of these systems is a challenge. Byzantine failure detectors provide an elegant abstraction for implementing Byzantine fault tolerance. However, very few works have been proposed for the new distributed system scenario. This paper presents a model and a protocol able to implement an unreliable Byzantine failure detector adequate for dynamic networks with unknown membership. The protocol has the interesting feature to be time-free, that is, it does not rely on timers to detect omission failures. To the best of our knowledge, the adoption of a time-free Byzantine failure detection is novel and this paper provides a first insight towards the understanding and implementation of such an approach in networks with unknown membership.**

***Keywords*-Byzantine fault-tolerance, failure detectors, Byzantine failures, dynamic distributed systems, wireless networks**

I. Introduction

Modern distributed systems, deployed over ad-hoc networks, such as wireless mesh networks (WMN), wireless sensor networks (WSN) are inherently dynamic. They are composed by a dynamic population of nodes, which randomly join and leave the network, at any moment of the execution, so that only a partial knowledge about the system's properties can be retained. Global assumptions, such as the knowledge about the whole membership, the maximum number of failures, complete or reliable communication, are no more realistic. Therefore, classical distributed protocols are no longer appropriate for this new context, since they make the assumption that the whole system is static and its composition is previously known.

Byzantine fault tolerance (BFT) [1] plays an important role on the development of dependable dynamic distributed systems. It deals with a number of security problems by tolerating the presence of corrupted processes, which may behave in an arbitrary manner, trying to hinder the system to work accordingly to its specification. The implementation of BFT techniques is a major challenge on dynamic distributed systems as many factors favor the action of malicious agents, e.g., the dynamic population, the wireless communication medium, the necessity to cooperate in order to achieve fundamental tasks (e.g., routing).

An *Unreliable failure detector*, namely FD, is a fundamental distributed service that provides an elegant approach to design secure, dependable and modular systems [2]. It can informally be seen as a per process oracle which gives hints (via a list of suspicions) on which processes in the system are faulty. The failure detector is unreliable in the sense that it may erroneously add to the 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 the list. Therefore, the FD may repeatedly add and remove the same process from its list of suspected processes.

Many important secure and dependable services (e.g., group membership, atomic commitment, consensus) and middleware (e.g., group communication, replicated and transaction servers) make direct use of FDs. FDs exempts the overlying service to deal with the failure treatment and synchrony requirements, so that it can just take care about its inherent task. The service is designed and proved correct based only on the formal properties provided by a FD class and it is exempted to deal with low-level aspects. The FD implementation and practical assumptions can be addressed independently. In this sense, the implementation can be better adapted to the particular characteristics of each environment; moreover, one FD implementation can serve many applications. An advantage of providing a FD for dynamic networks is that existing applications that already run on top of static networks using FDs could be more easily ported to the new context. Thus, seeking for conditions and protocols to implement Byzantine FDs in dynamic networks is of utmost interest.

But, differently from FDs for crash or "benign" failures, in which the FD implementation and practical assumptions can be addressed independently, FDs for Byzantine failures must rely on the detection on the application algorithm that uses it as an underlying oracle. This is because the detection is made according with the message contents and communication pattern followed by the specific algorithm. This inherent symbiosis between the FD and application algorithm is perhaps at the cause of the very little work done until now in the domain of Byzantine fault detection.

A. Related Work

Failure detectors have been proposed by Chandra and Toueg [2] as a way to circumvent the impossibility to solve consensus deterministically in an asynchronous distributed system prone to crash failures [3]. Roughly, consensus is a fundamental agreement problem which allows a set of processes to agree on a common proposal output value. It is at heart of many other dependable services, e.g., group membership, atomic broadcast, replicated servers, etc.

In the context of arbitrary failures, Malkhi *et al.* [4] extend the theory of Chandra and Toueg [2] and define a FD able to identify processes that prevent the progress of the algorithm using it. Doudou *et al.* [5] introduce the concept of *muteness failure detectors* in which the oracle detects when a process is mute, that is, when it ceases sending messages required by the algorithm. While these two works are restricted to a small subset of failures, the work of Kihlstrom *et al.* [6] extend the classical model of FDs for crash failures [2] to propose new classes able to consider more generic Byzantine failures and to solve consensus. Baldoni *et al.* [7] provide a framework to solve consensus which integrates muteness failure detectors (for mute crashes) and a byzantine behavior detector (for other Byzantine failures).

All these previous contributions about Byzantine failure detection [4], [5], [6], [7] consider a classical distributed system in which the communication graph is complete and the number and set of participants are globally known. An exception is [8], but it solves only a subset of the Byzantine failure detection problem to the specific application of routing.

Recently, Haeberlen *et al.* [9] presented the PeerReview system and propose a concrete solution to the Byzantine fault problem based on the use of accountability to detect and expose node faults. The solution is suitable for dynamic systems which span multiple administrative domains, e.g., P2P and overlay multicast systems, but it does not consider the case of systems in which the membership is unknown since the beginning of the execution. In a subsequent work [10], the same authors provide a formal study of the generic fault detection problem. They give a formal definition of commission (or security) and omission (or progress) faults [6] and identify some bounds on the costs of solving a weak definition of failure detection problem in asynchronous systems with authenticated channels.

All the Byzantine FDs proposed so far adopt the *timer-based* model to detect progress failures. This is a common design principle which supposes that eventually some bound on the transmission delays will permanently hold. However, these bounds are not known and they hold only after some unknown time [2]. An alternative approach suggested by [11] is *time-free* and considers that the system satisfies a message exchange pattern on the execution of a communication primitive. It does not rely on timers to detect crash failures and assumes that the responses from some stable known process to a query launched by other processes permanently arrive among the first ones. This idea has been exploited by [12] to develop a FD for dynamic networks, but for the crash failure model. 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). Both approaches (timer-based and time-free) are orthogonal and cannot be compared.

In dynamic networks, since the communication delays may frequently vary due to failures, arrivals and departures of nodes, the statement of the transmission bounds required by the timer-based detection becomes a big challenge. In this sense, the time-free model appears as a suitable alternative for being used in a dynamic set [13], [14].

B. Contributions

This paper advocates the use of the time-free approach to provide Byzantine failure detection [14]. It proposes a model, a specification, and an algorithm to implement an unreliable Byzantine FD adequate for dynamic networks with unknown membership and partial communication connectivity. To the best of our knowledge, the adoption of a time-free Byzantine detection in networks with unknown membership is novel and this paper provides a first insight towards the understanding and implementation of such an approach.

The rest of the paper provides the model (Section II), time-free additional assumptions (Section III), the algorithm (Section IV), its correctness proofs (Section V) and the conclusion (Section VI).

II. Model for Byzantine Failure Detection in Dynamic Networks

A. System Model

We are particularly interested in systems deployed over wireless ad-hoc networks, such as WSNs and WMNs. The system is a set of nodes communicating by broadcasting messages via a packet radio network. *It is asynchronous*: there are no assumptions on the relative speed of processes or on message transfer delays. There is no global clock known to the processes, but to simplify the presentation, we take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers.

Finite arrival model [15]. The network is a dynamic system composed of infinitely many processes; but each run consists of a finite set Π of $n > 3$ nodes, namely, $\Pi = \{p_1, \ldots, p_n\}$. This model properly expresses dynamic networks where nodes join and leave the system as they wish. It is suitable for long-lived or unmanaged applications, as for example, sensor networks deployed to support crises management or help on dealing with natural disasters.

The membership is unknown. Processes are neither aware about Π nor n, because, moreover, these values can vary from run to run [15]. There is one process per node; each process knows its own identity, but it does not necessarily knows the identities of the others. Nonetheless, they can make use of the broadcast facility of the wireless medium to know one another. Thus, we consider that a process knows a subset of Π, composed of nodes with whom it previously communicated.

Process failure model. Processes are subject to *Byzantine failures* [1], i.e., they can deviate arbitrarily from the algorithm they are specified to execute and work in collusion to corrupt the system behavior. A process p_i that does not follow its algorithm specification is said to be *Byzantine*; otherwise, it is *correct*, in this case, the predicate *correct*(p_i) is true. In particular, a Byzantine process may send messages not previously defined by its algorithm or may omit to send messages it is supposed to. In this sense, a process that crashes can be regarded as Byzantine. Notice that the sets of correct and Byzantine processes form a partition of Π. Every process is uniquely identified and a Byzantine process cannot obtain more than one identifier. Thus, it is impossible to launch a *sybil attack* against the system [16].

Communication graph is dynamic. Due to arbitrary joins, leaves, and failures, the network is represented by a communication graph with a dynamic topology, thus the relations between nodes take place over a time span $\mathcal{T} \subseteq \mathbb{N}$. Following [17], 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) $\mathcal{T} \subseteq \mathbb{N}$ is a time span, (4) $\rho : E \times \mathcal{T} \to \{0, 1\}$ is an *edge presence* function, indicating whether a given edge $e \in E$ is available at a given time $t \in \mathcal{T}$, such that $\rho(e, t) = 1$ iff e is present at t, otherwise $\rho(e, t) = 0$, (5) $\zeta : E \times \mathcal{T} \to \mathbb{N}$ is a *latency* function, indicating the time taken to cross a given edge e if starting at a given time t; since the system is asynchronous, there is no bound for this time, thus, we consider that ζ exists but cannot be estimated, (6) $\psi : V \times \mathcal{T} \to \{0, 1\}$ is a *node presence* function, indicating whether a given process $p_i \in V$ is up at a given time $t \in \mathcal{T}$, such that $\psi(p_i, t) = 1$ iff node p_i is up at t, otherwise $\psi(p, t) = 0$.

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

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

Authenticated channels. We assume a public key cryptosystem in which every process p_i holds a private key $\mathcal{K}_i$ with which it can sign its messages (e.g., RSA [18]) and every process in the system can obtain the public key of every other node in order to authenticate the sender of any signed message, even if another process has relayed that message. Byzantine processes cannot subvert the cryptographic primitives. Without authenticated channels, it is not possible to tolerate process misbehavior in an asynchronous system since a single faulty process can play the roles of all other processes to some (victim) process.

Communication is fair-lossy. A message m sent by a correct process p_i an infinite number of times is received by every correct process p_j in its neighborhood an infinite number of times, or p_j is Byzantine. 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. In addition, there is no message duplication, modification or creation; this means that a Byzantine node is not allowable to interfere on message transmissions by correct processes, and even if it sends multiple versions of a message, the message will be perceived by the others as only one message with the same contents [19], [20]. The fair-lossy assumption seems to be inadequate for the dynamic environment; above all, wireless channels are inherently unreliable and can in addition suffer from a number of attacks, e.g., a malicious node can raise a collision attack in messages sent by honest nodes, preventing reception. Fortunately, some works about ensuring reliability under these conditions have recently appeared. For example, [20] presents an approach to implement a reliable local broadcast primitive with probabilistic guarantees in a wireless network with lossy channels and Byzantine adversaries.

B. Byzantine Failures

Two requirements must be satisfied in a system prone to Byzantine failures: (i) correct processes must have a consistent view of the messages sent by every process; (ii) correct processes must be able to verify if a message is consistent with the requirements of the algorithm in execution. Thus, Byzantine failure detection is defined as a function of some algorithm $\mathcal{A}$. The first requirement may be addressed by two distinct techniques: information redundancy or unforgeable digital signatures; the second can be met by adding certificates to the messages, so that its content may be validated [21].

Two superclasses of Byzantine failures can be distinguished [6]: *detectable*, when the external behavior of a process provides evidence of the failure and *non-detectable*, otherwise. Non-detectable failures are grouped in unobservable, when other processes cannot perceive the occurrence of a failure (e.g., when a faulty process informs a parameter different from the supplied by the user) and undiagnosable, when it is not possible to identify the perpetrator of failure (e.g., the processes receive an unsigned message).

This work deals with detectable failures. They are classified in *omission* (or *progress*) failures and *commission* (or *security*) failures [6]. Omission failures hampers the termination of the computation, since a faulty process does not send the messages required by the specification. Commission failures violate invariant properties to which processes must obey, and can be defined as the noncompliance of one of the following restrictions: (i) a message sent by a process to other processes must have the same contents (no mutant message); (ii) the messages sent must conform the algorithm $\mathcal{A}$ under execution.

C. Stability Assumptions

One important aspect on the design of FDs for dynamic networks concerns the time period and conditions in which processes are connected to the system. During unstable periods, certain situations, as for example, connections for very short periods or numerous joins or leaves along the execution (characterizing a churn) could block the application and prevent any useful computation. Thus, to implement any global computation, the system should present some stability conditions that when satisfied for long enough time will be sufficient to satisfy the requirements of the application and terminate.

In order to implement FDs with an unknown membership, processes should interact with some other process that never departs from the system to be known. If there is some process such that the rest of processes have no knowledge whatsoever of its identity, there is no algorithm that implements a FD with weak completeness [22]. *Completeness* characterizes the FD capability of suspecting every faulty process permanently. In this sense, the characterization of the *actual membership* of the system, that is, the set of processes which might be considered for the computation is essential.

We consider then that a process p_i *joins* the network at some point $t \in \mathcal{T}$ in time. Subsequently, p_i must somehow communicate with the others in order to be known. In a wireless network, this can be done by simply broadcasting its identity to the neighbors. Due to this initial communication, every process p_j is able to gather an initial partial knowledge $\Pi_j \subseteq \Pi$ about the system's membership which increases over the time along p_j's execution. Let $\Pi_j(t)$ be the partial knowledge of p_j by time t. A process is *known* if, after having joined the system, it has been identified by some stable process. A *stable* process is thus a correct process that, after had entered the system for some point in time, never fails (by deviating from the algorithm specification, crashing or leaving the network); otherwise, it is *faulty*. When p_i *leaves* the network at time $t' > t$, it can re-enter the system with a new identity, thus, it is considered as a new process. Notice that sybil's attacks are not authorized. Processes may join and leave the system as they wish, but the number of re-entries is bounded, due to the finite arrival assumption.

Definition 1: Process Status. A process p_i may assume the following status in the system.

$join^t(p_i) \Leftrightarrow \exists t, \forall s < t,\ \psi(p_i, s) = 0 \wedge \psi(p_i, t) = 1$

$stable^t(p_i) \Leftrightarrow correct(p_i) \wedge \exists t, \forall t' \geq t,\ \psi(p_i, t') = 1$

$crash^t(p_i) \Leftrightarrow \exists s, t,\ s < t, \psi(p_i, s) = 1 \wedge \forall t' \geq t, \psi(p_i, t') = 0$

$faulty^t(p_i) \Leftrightarrow \exists t, crash^t(p_i) \vee p_i$ *deviates from* $\mathcal{A}$ at t

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

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

Definition 2: Membership. The membership of the system is the KNOWN set.

$\textsc{Stable} \overset{def}{=} \bigcup_{t \in \mathcal{T}} S(t)$

$\textsc{Faulty} \overset{def}{=} \bigcup_{t \in \mathcal{T}} F(t)$

$\textsc{Known} \overset{def}{=} \{p_i : \exists t \in \mathcal{T}, p_i \in \textsc{Stable} \cup \textsc{Faulty} \wedge known^t(p_i)\}$

D. Connectivity Assumptions

Let $V_{KS} = \textsc{Known} \cap \textsc{Stable}$ 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) [17] is important for our study. It means that the TVG is connected over time.

Assumption 1: Network with Byzantine Coverage. The system, represented by the TVG $\mathcal{G}_{KS}$, has *Byzantine coverage* if and only if:

(1) $\exists t \in \mathcal{T}, \forall t' \geq t, \forall p_i, p_j \in V_{KS}, p_i \rightsquigarrow p_j$ (p_i *reaches* p_j). That is, after t, there is a *journey* $\mathcal{J}$, $\forall p_i, p_j \in \textsc{Known} \cap \textsc{Stable}$. For a communication purpose, we assume that each edge e_i of $\mathcal{J}$ remains available until a message is delivered, thus $\rho(e_i, t) = 1, \forall t \in [t_i, t_i + \zeta(e_i, t_i)]$.

(2) the minimum degree of a node p_i in G_{KS} is $Deg_i^r > 2f_i, \forall r \in \mathcal{T}$.

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 [23], [17] and thus to ensure the global properties of the failure detector [2], [22], [13], [19], [24], [25]. In practice, whenever a higher number of faults occur, the network may be disconnected for a while and then the progress may be compromised. But, the most important is to ensure the safety of the protocol during these bad periods. As soon as the network starts presenting stability, the connectivity assumption will be satisfied and the protocol progresses.

Recent works about Byzantine radio communication advocate a "local" fault model, instead of a "global" fault model, as an adequate strategy to deal with the unreliability of wireless channels in spite of Byzantine failures [19], [26], [24], [20], [25]. They define bounds on the maximum number of local failures in order to reliably deliver data. Precisely, [24], [25] show that it is possible to achieve reliable broadcast if less than $1/4$ of nodes in any neighborhood are Byzantine and impossible otherwise. 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. Thus, we consider that f_i is the maximum number of faulty processes in p_i's neighborhood.

Connectivity Assumption (2) establishes a bound to tolerate Byzantine node faults. It is a guarantee that information from/to process p_i is going to be sent/received to/from a minimum of stable nodes in its neighborhood. Precisely, at least $f_i + 1$ stable nodes can communicate with p_i, ensuring that initially $p_i \in \Pi_j$ of at least $f_i + 1$ stable processes. Afterwards, if p_i is faulty, eventually at least $f_i + 1$ stable processes will suspect p_i and may spread the suspicion to the remaining of the system, so that the *completeness* property of the FD can be satisfied (see next Section). As we will show in the correctness proofs (Section V), Connectivity assumption (2), i.e., $Deg_i^r > 2f_i$ is sufficient and necessary for the failure detector protocol to tolerate Byzantine node faults. However, additionally, as it has been proved by [24], [25], to implement fair-lossy channels and tolerate Byzantine failures on channels, the condition $Deg_i^r > 4f_i, \forall r \in \mathcal{T}$ is necessary as well.

E. Byzantine Unreliable Failure Detector Specification

Kihlstrom *et al.* [6] define Byzantine FD classes which differ from those described by Chandra and Toueg [2], since the latter deals only with crash failures. Let $\mathcal{A}$ be an algorithm that uses the failure detector as an underlying module. The class $\Diamond\mathcal{S}(Byz, \mathcal{A})$ is an adaptation of the $\Diamond\mathcal{S}$ class to Byzantine failures. It is the focus of our work. Nonetheless, its properties should be adapted to a dynamic network. With this aim, we define the class of *Eventually Strong Byzantine Failure Detectors with Unknown Membership*, namely $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$. It keeps the same properties of $\Diamond\mathcal{S}(Byz, \mathcal{A})$, except that they are now valid to stable known processes. Informally, these properties are:

• *Strong Byzantine completeness* (for $\mathcal{A}$): eventually, every stable known process suspects permanently every process that has detectably deviated from $\mathcal{A}$;

• *Eventual weak accuracy*: eventually, one stable known process is never suspected by any stable known process.

Definition 3: $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ Eventually Strong Byzantine FD with Unknown Membership. Let $t \in \mathcal{T}$. Let p_i, p_j be nodes. Let $susp_j$ be the list of nodes that p_j currently suspects of being faulty. The $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ class contains all the failure detectors that satisfy:

Strong Byzantine completeness (for $\mathcal{A}$) $\overset{def}{=}$ $\{\exists t \in \mathcal{T},\ \forall t' \geq t,\ \forall p_i \in \text{KNOWN} \cap \text{FAULTY} \Rightarrow p_i \in susp_j,\ \forall p_j \in \text{KNOWN} \cap \text{STABLE}\}$;

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

III. Towards a Time-Free $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ Byzantine Failure Detector

A. Local Message Exchange Pattern

Most of the protocols for crash failure detection are based on the exchange of heartbeat messages by the failure detector. Nevertheless, in a Byzantine environment such a mechanism is no longer enough. A Byzantine process may correctly answer the FD messages, yet without guaranteeing progress and safety to the algorithm under execution. Therefore, the failure detection should be based on the pattern of the messages sent during the execution of the algorithm $\mathcal{A}$ which uses the FD as a building block.

We advocate the use of the time-free approach to raise suspicions and propose a FD protocol whose detection does not use timers but is based on the exchange of messages required by algorithm $\mathcal{A}$. Thus, when algorithm $\mathcal{A}$ requires the processes to exchange a message m, every process p_i waits until the reception of m from at least α_i distinct senders; for the remaining processes of its partial knowledge ($\in \Pi_i$), it raises an omission failure suspicion. In this case, p_i will send a SUSPICION message to processes in its neighborhood, carrying out its local view about suspicions. The detection follows a local message exchange pattern, i.e., between the nodes in the neighborhood [12]; thus, α_i corresponds to the minimum amount of stable known nodes in the neighborhood of p_i, i.e., $\alpha_i = Deg_i^t - f_i$. Knowing that $|Deg_i^t| > 2f_i$, $\alpha_i \geq f_i + 1$ (from the Byzantine coverage). The actual value of α_i depends on the type of dynamic network considered (WSN, WMN) as well as the topology of the network during execution.

It is worth noticing that, since the detection adopts a time-free approach in which suspicions are based on message exchange and not on the expiration of a timeout, the communication pattern followed by processes in algorithm $\mathcal{A}$

should be distributed. That is, whenever $\mathcal{A}$ requires a process p_i to broadcast a message m in some step, then this same message m must be broadcast by all the other processes in $p_i's$ neighborhood. In practice, if the communication pattern followed by algorithm $\mathcal{A}$ is not distributed, one can simulate this pattern by requiring processes to relay the messages received to the other processes. Thus, when process p_j receives a message m for the first time, before proceeding with the computation, it must broadcast m to all processes in its neighborhood. Without this behavior, since the underlying system is asynchronous, one could not distinguish an omission failure of p_i from a delay on the delivering of the message m from p_i [2]. The distributed pattern will finally ensure the reception of at least α_i messages required by the time-free failure detector protocol.

B. Behavioral System Property

With a time-free approach [11], [13], in order to satisfy *eventual weak accuracy* property of the $\Diamond\mathcal{S}^M(Byz,\mathcal{A})$ FD class, there must exist a stable known process p_i whose messages from some point on are always among the first messages received by its neighbors, at every request of $\mathcal{A}$. Thus, eventually p_i will no longer be suspected by any stable known process. The *Byzantine responsiveness* property characterizes this desired behavior.

Property 1: $Byz\mathcal{RP}$ (Byzantine Responsiveness Property). Let $X_j(t')$ be the set of processes from which p_j received the message required by algorithm $\mathcal{A}$ at its last step in execution until $t' \in \mathcal{T}$. Process p_i satisfies $Byz\mathcal{RP}$ at time t if:

$$Byz\mathcal{RP}^t(p_i) \Leftrightarrow \exists t \in \mathcal{T}, stable^t(p_i) \ \wedge$$
$$p_i \in X_j^{t'}, \forall p_j \in N_i^{t'}, \forall t' \geq t \vee faulty^{t'}(p_j)$$

Thus, the following behavioral assumption should be satisfied in the network in order to implement $\Diamond\mathcal{S}^M(Byz,\mathcal{A})$: $\exists p_i \in$ KNOWN $\cap$ STABLE : $Byz\mathcal{RP}^t(p_i)$ eventually holds. As a matter of comparison, in the timer-based model, the $Byz\mathcal{RP}^t(p_i)$ 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. That assumption coincides to the classical one used to implement $\Diamond S$ FDs in traditional networks [4], [6], [7].

C. Practical Issues

WSNs and WMNs are a good examples of networks which would satisfy the assumptions of our model, specially the $Byz\mathcal{RP}$ property and network assumptions. In a WMN, the client nodes move around a fixed set of nodes (the backbone of the network) and each 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. For all these platforms, special nodes (the fixed nodes for WMN, CHs for WSN) eventually form a strongly connected component of stable nodes; additionally, some of these nodes can be regarded as fast, so that they will always answer messages faster than the other nodes, considered as slow nodes. Thus, one of these fast nodes may satisfy the $Byz\mathcal{RP}$ property. The stability conditions and the $Byz\mathcal{RP}$ may seem strong, but in practice they should just hold during the time the application needs the completeness and accuracy properties of FDs of class $\Diamond\mathcal{S}^M(Byz,\mathcal{A})$, as for instance, the time to execute a consensus algorithm [7].

IV. A TIME-FREE BYZANTINE FAILURE DETECTOR OF THE CLASS $\Diamond\mathcal{S}^M(Byz,\mathcal{A})$

This section describes our protocol for implementing a FD of class $\Diamond\mathcal{S}^M(Byz,\mathcal{A})$ for a network of unknown membership that satisfies the model and assumptions stated in Sections II and III.

Security Failure Detection: In order to enable the *commission* (or *security*) failure detection, a message format must be established for algorithm $\mathcal{A}$. Every message m must also include a certificate that enables other processes to verify its consistency with algorithm $\mathcal{A}$. A message m *is valid* if (i) m is *properly signed*, that is, it has been sent by a process i with a valid key $\mathcal{K}_i$ and (ii) m is *properly formed* with regard with the algorithm specification, that is, it is syntactically correct and has the expected format and type for the specific algorithm. If a stable process detects the non validity of a received message, either for not obeying to the format or for a non valid authentication, it will permanently suspect the sender and will forward a SUSPICION message to the remaining processes, so that the suspicion is propagated.

Notice that it is also necessary to detect *mutant messages*. This anomaly happens when a process sends two or more different versions of the same message. One way to deal with this problem it to require stable processes to forward every received message. Additionally, processes should maintain a history of messages received by every process. This approach can be used in the case of point-to-point communication [6]. In our model, processes communicate through local broadcast under fair-lossy channels. Based on the recent advances to implement reliable wireless channels, we can assume that a message broadcast will be received with equal content by every stable known process [19], [26], [24], [20], [25], so that the protocol do not have to deal with mutant messages.

Suspicion Generation: Every SUSPICION message of an omission fault raised over p_j is related to a message m required by $\mathcal{A}$. That is, p_j is suspected of not sending the messages of $\mathcal{A}$ it should. Thus, messages must have unique identifiers. Let $m.id$ be the identifier of message m. Suspicions are propagated on the network and a stable process will adopt a SUSPICION not generated by itself if and only if it receives a SUSPICION message *properly signed* from at least $f_i + 1$ different senders. This requirement denies a Byzantine process to impose suspicions on stable processes. Since the network has a Byzantine coverage (Assumption 1), at least $(f_i + 1)$ neighbors of p_i are stable and shall spread a suspicion of its failure to their respective neighbors. Since, moreover, there is a time at which a journey is formed by each pair of stable processes in the system, eventually a stable process in the network receives at least $f_i + 1$ occurrences of this suspicion and may adopt it. This ensures the satisfaction of the *strong Byzantine completeness* property of the $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ FD.

Mistake Generation: Let p_i be a process that has been suspected of not sending a message m from $\mathcal{A}$. If eventually a stable process p_j receives m *properly signed* from p_i, p_j will declare a *mistake* on the suspicion and will spread m to the remaining nodes, so that they can do the same. In a network with Byzantine coverage, there will be at least one journey formed only by stable known processes between p_j and every stable known process. Then, every other stable process will receive m and will be able to remove the related suspicion. This behavior allows a Byzantine process to provoke a suspicion and revoke it continuously, masking part of the omission failures and degrading the failure detector performance. Nevertheless, it is not possible to distinguish that situation from the slowness of a process or an instability on a channel.

VARIABLES:
- $output_i$: stores the failure detector output, i.e., the set of processes identities that p_i suspects of having failed;
- $known_i$: stores the set of processes that have communicated with p_i. It is updated at the reception of SUSPICION messages or messages required by $\mathcal{A}$;
- $extern\_susp_i$: matrix that stores external suspicions (generated by other nodes). The matrix is indexed by a process identifier q and a message identifier idm. Every entry stores the set of processes from which p_i has received suspicions about q and message(idm);
- $intern\_susp_i$: array of internal suspicions indexed by process ids. An internal suspicion is generated by not receiving a message required by $\mathcal{A}$ or by the presence of at least $f_i + 1$ external suspicions on a pair process-message;
- $mistake_i$: array that stores, for every process p_j, the set of mistakes related to p_j. A mistake is stored as a message required by $\mathcal{A}$ about which a suspicion has been raised;
- $byzantine_i$: set of tuples in the form $\langle$process, message$\rangle$ that prove Byzantine behavior on the related process. The notation $\langle p, - \rangle$ means "any tuple related to process p";
- rec_from_i: set of processes from which p_i received the message required by $\mathcal{A}$.

PRIMITIVES:
- $m.id$ – returns the identifier of message m;
- message(idm) - returns the message related to identifier idm;
- broadcast m – broadcasts a message m to the neighbors of p_i;
- keys(v) – returns the index set of a dynamic array v;
- ids(s) – returns the set of ids of the messages in set s.

AUXILIARY PROCEDURES:
- *AddInternalSusp*(q, m) (lines 26-28, A1): adds an internal suspicion on process q and message m;
- *ValidateReceived*(q, m) (lines 30-45, A1): verifies if the message m received from q is valid (well-formed), removing any suspicions related to the pair (q, m) in the affirmative case, otherwise generating a security failure suspicion. Also, updates the set of nodes known by p_i ($known_i$) and call the *UpdateSuspicions*() function for the message;
- *UpdateSuspicions*(q, m) (lines 13–35, A2): if m is of the type SUSPICION, updates the internal state of p_i with the information in m.
- *AddByzantine*(q, m) (lines 1-3, A2): adds q permanently to the list of Byzantine processes (and, consequently, to the FD output), along with the message m as a proof of the Byzantine failure;
- *AddMistake*(q, m) (lines 5-11, A2): adds a mistake on a previous suspicion about process q and message m, removing any corresponding internal or external suspicions. If q has no other suspicions and has not presented Byzantine behavior, removes q from the failure detector output;
- *AddExternalSusp*(q, idm, p_s) (lines 37-41, A2): adds an external suspicion from p_s about process q and message identified by idm. Also, if there are at least $f_i + 1$ external suspicions about q and message(idm), generates a corresponding internal suspicion, if not already present.

A. Algorithm Description

Algorithms A1 and A2 implement a $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ FD. Every process p_i executes three parallel tasks. The variables, primitives, tasks, and procedures are described below.

T1. Generating new SUSPICION messages (lines 5-14, A1). When algorithm $\mathcal{A}$ requires the processes to exchange a message m (line 6), every node p_i waits until the reception of m from at least α_i neighbors, whose identifiers are stored in rec_from_i (lines 7-8). For the remaining processes known by p_i, it adds an internal omission failure suspicion (lines 9-11). Then every message has its format and certificates verified through *ValidateReceived()* (lines 12-14, 30-45, A1). Incorrect messages lead to security failure suspicions (lines 34-35, A1) and update the detector output; correct messages generate mistakes on possible omission failure suspicions (lines 37-38, A1).

Algorithm 1 Byzantine Failure Detector (A1)

```
 1: init:
 2: $output_i \leftarrow known_i \leftarrow \emptyset$; $extern\_susp_i \leftarrow [\,][\,]$
 3: $intern\_susp_i \leftarrow mistake_i \leftarrow [\,]$; $byzantine_i \leftarrow \emptyset$
 4:
 5: Task T1: /* generating new suspicions */
 6: when $p_i$ requires a message $m$ from $\mathcal{A}$ do
 7: wait until receive a $m$ properly signed for the first time from
    at least $\alpha_i$ distinct processes
 8: $rec\_from_i \leftarrow \{p_j \mid p_i$ received a message from $p_j$ at line
    7$\}$
 9: for all $p_j \in (known_i \setminus rec\_from_i)$ do
10:     AddInternalSusp($p_j$, $m$)
11: end for
12: for all $m_j$ received at line 7 from $p_j$ do
13:     ValidateReceived($p_j$, $m_j$)
14: end for
15:
16: Task T2: /* receiving SUSPICION or message from $\mathcal{A}$ coming
    from slow process */
17: upon receipt of $m$ properly signed from $p_j$ do
18: ValidateReceived($p_j$, $m$)
19:
20: Task T3: /* broadcasting suspicion state */
21: loop
22:     broadcast ⟨SUSPICION, $byzantine_i$, $mistake_i$,
        $intern\_susp_i$, $extern\_susp_i$⟩
23: end loop
24:
25: /* AUXILIARY PROCEDURES */
26: procedure AddInternalSusp($q$, $m$):
27: $intern\_susp_i[q] \leftarrow intern\_susp_i[q] \cup \{m.\mathrm{id}\}$
28: $output_i \leftarrow output_i \cup \{q\}$
29:
30: procedure ValidateReceived($q$, $m$):
31: if $m$ was sent directly by $q$ then
32:     $known_i \leftarrow known_i \cup \{q\}$
33: end if
34: if $m$ is not properly formed then
35:     AddByzantine($q$, $m$)
36: else
37:     if $m.id \in intern\_susp_i[q]$ or $m$ was not sent directly by
        $q$ then
38:         AddMistake($q$, $m$)
39:     end if
40:     if $m$ is from $\mathcal{A}$ and $m$ is received for the first time then
41:         broadcast $m$ /* relaying $m$ to the other processes*/
42:     else
43:         UpdateSuspicions($q$, $m$)
44:     end if
45: end if
```

T2. Receiving SUSPICION messages and $\mathcal{A}$ messages from slow processes (lines 16-18, A1). When a message m is received from a remote process p_j, its format and certificates are verified through *ValidateReceived()*. There are two possibilities: (1) m is a message required by $\mathcal{A}$ that is received lately by p_i, probably after a suspicion over p_j has been generated in task T1. In this case, m is treated similarly to task T1 (lines 30-38, A1). (2) m is a SUSPICION message. In this case, the internal state of p_i is going to be updated (line 43, A1) as follows:

Algorithm 2 Byzantine Failure Detector (A2)

```
 1: procedure AddByzantine($q$, $m$):
 2: $output_i \leftarrow output_i \cup \{q\}$;
 3: $byzantine_i \leftarrow byzantine_i \cup \{\langle q, m\rangle\}$
 4:
 5: procedure AddMistake($q$, $m$):
 6: $mistake_i[q] \leftarrow mistake_i[q] \cup \{m\}$
 7: $extern\_susp_i[q][m.\mathrm{id}] \leftarrow \emptyset$
 8: $intern\_susp_i[q] \leftarrow intern\_susp_i[q] \setminus \{m.\mathrm{id}\}$
 9: if $intern\_susp_i[q] = \emptyset$ and $\nexists\langle q, -\rangle \in byzantine_i$ then
10:     $output_i \leftarrow output_i \setminus \{q\}$
11: end if
12:
13: procedure UpdateSuspicions($q$, $m$):
14: if $m = $ ⟨SUSPICION, $byzantine_q$, $mistake_q$, $intern\_susp_q$,
    $extern\_susp_q$⟩ then
15:     for all $p_x \in$ keys($extern\_susp_q$) do
16:         for all $idm_x \in$ keys($extern\_susp_q[p_x]$) properly
            signed | $idm_x \notin$ ids($mistake_i[p_x]$) do
17:             for all $p_y \in extern\_susp_q[p_x][idm_x]$ do
18:                 AddExternalSusp($p_x$, $idm_x$, $p_y$)
19:             end for
20:         end for
21:     end for
22:     for all $p_x \in$ keys($intern\_susp_q$) do
23:         for all $idm_x \in intern\_susp_q[p_x]$ properly signed |
            $idm_x \notin$ ids($mistake_i[p_x]$) do
24:             AddExternalSusp($p_x$, $idm_x$, $q$)
25:         end for
26:     end for
27:     for all $p_x \in$ keys($mistake_q$) do
28:         for all $m_x \in mistake_q[p_x]$ properly signed do
29:             ValidateReceived($p_x$, $m_x$)
30:         end for
31:     end for
32:     for all $\langle p_x, m_x\rangle \in byzantine_q$ | $m_x$ is properly signed
        do
33:         ValidateReceived($p_x$, $m_x$)
34:     end for
35: end if
36:
37: procedure AddExternalSusp($q$, $idm$, $p_s$):
38: $extern\_susp_i[q][idm] \leftarrow extern\_susp_i[q][idm] \cup \{p_s\}$
39: if $|extern\_susp_i[q][idm]| \geq f_i + 1$ then
40:     AddInternalSusp($q$, message($idm$))
41: end if
```

Updating internal state (lines 13–35, A2). Upon the receipt of a SUSPICION message from a neighbor q (line 14, A2), a process p_i updates its internal state with new information. Internal and external suspicions from q are added to the external suspicion set of p_i (lines 15-26, A2), possibly generating new internal suspicions (lines 37-41, A2). Note that a security failure suspicion will be raised on q if the SUSPICION message m is malformed. Mistake information and security failure proofs are treated similarly to messages received directly from the sender through *ValidateReceived()*.

T3. Broadcasting suspicions and mistakes (lines 20-23, A1). This task periodically sends SUSPICION messages to p_i's neighbors carrying out its view on internal and external suspicions, mistakes and security failure proofs. The neighbors of p_i will receive that message in task T2.

V. Correctness Proof

To implement a failure detector of class $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$, the algorithm in Section IV should satisfy the *Byzantine strong completeness* and the *eventual weak accuracy* properties. In the following, a sketch of the proofs of the algorithm is given.

A. Byzantine Strong Completeness

Lemma 1: If a process p_i never send message m, then a process $p_j \in$ Known $\cap$ Stable will never execute *AddMistake*(p_i, m).

Proof: Assume, by contradiction, that some stable known process p_j executes *AddMistake*(p_i, m). Notice that *AddMistake()* is only invoked into *ValidateReceived()* (line 38, A1). The procedure *ValidateReceived()* is for its turn invoked in 3 cases: (1) on the reception of messages required for $\mathcal{A}$ on task T1 (line 13, A1) and task T2 (line 18, A1); (2) on the reception of SUSPICION messages on task T2 (line 18, A1); (3) on the update of the internal state with information from the neighbors (execution of *UpdateSuspicions()*, lines 29 and 33, A2). In all cases, the authentication of message m is properly verified (lines 7 and 17, A1; lines 28 and 32, A2). From this fact and since channels are authenticated, a faulty process p_f cannot send m in the place of p_i. The occurrence of case (2), specifically, could not lead to a call to *AddMistake()*, since there is no suspicion related to messages SUSPICION. Finally, we conclude that in all cases there is a contradiction, since for m to be received, p_i should had sent it at some point in time. Thus, the lemma follows.

Lemma 2: Let p_i be a process that fails by *omission*, $p_i \in$ Known $\cap$ Faulty.Then, eventually, every $p_j \in$ Known $\cap$ Stable will permanently include p_i in $output_j$.

Proof: Let t be the time at which p_i fails by *omission*, $faulty^t(p_i)$ is *true*. Let $t_0 < t$ be the time at which $known^{t_0}(p_i)$ is *true*. Let $t_0 \leq s \leq t$ be the time at which process p_i does not send the message m required by $\mathcal{A}$. From the Byzantine coverage Assumption 1, $|E_i^s| \geq 2f_i + 1$.

Case 1: $p_j \in E_i^t$. If this happens, from the execution of lines 21-23, A1, then p_j has received a message of type SUSPICION from p_i before time t. Thus, $p_i \in known_j$, according to the execution of lines 17-18, 31-33, A1. Whenever the execution of $\mathcal{A}$ requires m, p_j will wait until the reception of m from α_j distinct processes (lines 6-7, A1). This predicate will be satisfied at some point in time, since at most f_j process are faulty and $|E_j| > 2f_j$ (Assumption 1). Since p_i did not send m, p_i will not be included in rec_from_j (line 8, A1). According to the execution of lines 9-11 and 27-28, A1, $m.id$ will be included in $intern\_susp_j[p_i]$ and p_i will be included in $output_j$. Since p_i fails by omission, it will never send m afterwards. Thus, from Lemma 1 and lines 8-11, A2, $m.id$ will never be removed from $intern\_susp_j[p_i]$ and from p_i de $output_j$.

Case 2: If $p_j \notin E_i^t$. Since the network has Byzantine coverage (Assumption 1), then there is at least a journey J, between p_j and each stable known process $p_k \in E_i^t$ composed only by stable known processes. If there is more than one journey, than take the one with minimum distance. Let us prove, by induction on the length of J, that, eventually, p_k is added to $extern\_susp_j[p_i][m.\text{id}]$.
(1) If $|J| = 1$, then p_j is a neighbor of p_k. In this case, at some point in time, p_k send a SUSPICION message M (line 22, A1) with the certified information that $m.id$ $\in intern\_susp_k[p_i]$; since channels are fair-lossy, at some point, p_j receives M (line 17, A1). Since p_k is stable known, M is duly certified and formed; from Lemma 1, $m \notin mistake_j[p_i]$; Thus, from lines 18, 40 A1 and lines 22-26, A2, p_k is added to $extern\_susp_j[p_i][m.\text{id}]$ in line 38, A2 and the affirmation holds.
(2) If $|J| > 1$, we can assume by induction that the affirmation is true for the journey $P - p_j$ between p_k and a stable known process p_l, such that p_l and p_j are neighbors. For induction hypothesis, eventually, p_k is added to $extern\_susp_l[p_i][m.\text{id}]$ and, afterwards, p_l sends a SUSPICION message M (line 22, A1) with this information certified by p_k. Since channels are fair-lossy, eventually p_j receives M (in line 17, A1). Since p_l is stable known, M is duly certified and formed; from Lemma 1, $m \notin$ $mistake_j[p_i]$; thus, from lines 18, 40, A1 and lines 15-21, A2, p_k is added to $extern\_susp_j[p_i][m.\text{id}]$ in line 38, A2 and the affirmation holds.

From the above conditions, from $|E_i^t| \geq 2f_i + 1$ and knowing that there is at most f_i faulty processes, it follows that, at some point in time, p_j executes line 40, A2 and, from lines 27-28, A1, it adds $m.id$ to $intern\_susp_j[p_i]$ and p_i to $output_j$. Again, from Lemma 1, it follows that p_i will never be removed from $output_j$.

Lemma 3: Let p_i be a process that fails by *commission*, $p_i \in$ Known $\cap$ Faulty.Then, eventually, $\forall p_j \in$ Known $\cap$ Stable will permanently include p_i in $output_j$.

Proof: Let t be the time at which p_i fails by *commission*, $faulty^t(p_i)$ is *true*, that is p_i sends a message m not in accordance with $\mathcal{A}$. In this case, m is not well formed. By communication assumption, mutant messages are not possible; moreover, m is a certified message; otherwise, an undiagnosable faulty had been produced. Let $t_0 < t$ be the time at which $known^{t_0}(p_i)$ is *true*; this means that p_k received a message m from p_i and executed line 32, A1, $p_i \in known_k$, $p_k \in$ Known $\cap$ Stable. Since the network has Byzantine coverage (Assumption 1), then there is a journey J between p_k and each stable known process $p_j \in$ Known $\cap$ Stable composed by stable known processes. If there is more than one journey, than take the one with minimum distance. Let us prove, by induction on the length of J, that, eventually, p_j adds $\langle p_i, m \rangle$ to

$byzantine_j$ and p_i to $output_j$.
(1) If $|J| = 0$, then $p_j \in E_i^t$. Since channels are fair-lossy and m is certified, p_j receives m at some moment in lines 7 or 17, A1. In both cases, *ValidateReceived()* (lines 13 and 18, A1) is invoked. This procedure will attest the non-validity of m at line 34, A1. For its turn, *AddByzantine()* (lines 1-3, A2) adds p_i to $output_j$ and $\langle p_i, m \rangle$ to $byzantine_j$, and the affirmation holds.
(2) If $|J| > 0$, by induction, the affirmation is true for the journey $J \setminus \{p_j\}$ between p_k and a stable known process p_l, such that p_l and p_j are neighbors. In this case, $\langle p_i, m \rangle$ is in $byzantine_l$ and, at some point in time, p_l sends a SUSPICION message M with this information (line 22, A1); since channels are fair-lossy, eventually, p_j receives M at line 17, A1. Since p_l is stable known, M is duly certified and formed; thus, as m is certified from lines 18, 40 A1 and lines 32-34, A2, p_j invokes *ValidateReceived()* and attest the non-validity of m; thus, p_i is added to $output_j$ and $\langle p_i, m \rangle$ to $byzantine_j$ and the affirmation holds. From the above conditions and since p_j only removes p_i from $output_j$ if there is no pair $\langle p_i, - \rangle$ in $byzantine_j$ (lines 9-11, A2), p_i is definitely added to $output_j$ and the lemma follows.

B. Eventual Weak Accuracy

Lemma 4: Let $p_i, p_j \in$ KNOWN $\cap$ STABLE, then p_j never invokes *AddByzantine*$(p_i, -)$

Proof: The only invocation of *AddByzantine()* is in line 35, A1 into *ValidateReceived()*. From line 34, A1, knowing that p_j is stable known, this calling only occurs if p_i has sent a message which was not in good format; but this is impossible, since p_i is stable known as well. If a faulty process sends such a message in the place of p_i, then process p_j will discard it. This happens because channels are authenticated and p_j validates the authentication of every message it receives (lines 7 and 17, A1 and lines 28 and 32, A2), and the lemma follows.

Lemma 5: Let $p_i \in$ KNOWN $\cap$ STABLE and let m be a message required by $\mathcal{A}$. Assume that the property $Byz\mathcal{RP}$ $^t(p_i)$ holds for p_i at time t. Eventually, no process $p_j \in$ KNOWN $\cap$ STABLE will invoke *AddInternalSusp* (p_i, m).

Proof: The procedure *AddInternalSusp*() is called in 2 cases: CASE (1): in the task T1, during the reception of messages from $\mathcal{A}$ (line 10, A1); CASE (2): in the procedure *AddExternalSusp*() (line 40, A2), when the process receives more than f_i external suspicions regarding p_i.

CASE 1: Since $Byz\mathcal{RP}$ $^t(p_i)$ holds at time t, $p_j \in E_i^{t'}$ receives m from p_i, $\forall t' \geq t$, if $p_j \in$ KNOWN $\cap$ STABLE. Then, p_j adds p_i to its rec_from_j set in line 8, A1. Thus, p_j does not invoke *AddInternalSusp*(p_i, m) in line 10, A1. A process $p_k \notin E_i^{t'}$, $\forall t' \geq t$ (out of p_i's neighborhood) cannot receive messages directly from p_i, thus, p_k will never add p_i to $known_j$ (lines 31-33, A1); hence, it will never invoke *AddInternalSusp* (p_i, m) in line 10, A1 from $t' \geq t$. Both situations confirm Case 1.

CASE 2: Notice that *AddExternalSusp()* is only invoked in lines 18 and 24, A2 of *UpdateSuspicions*(). Moreover, $extern\_susp_j$ is only updated by $p_j \in$ KNOWN $\cap$ STABLE on the execution of *AddExternalSusp*() (line 37,A2). This means that every external suspicion regarding a stable known process was firstly generated as an internal suspicion (lines 39 and 40, A2). From the same argument of CASE 1, the stable known process p_j never adds $m.id$ to $intern\_susp_j[p_i]$ from $t' \geq t$ on the execution of task T1 (line 27, A1). If a Byzantine process $p_f \in$ KNOWN $\cap$ FAULTY adds p_j to $extern\_susp_f[p_i][m.\text{id}]$, then a stable known process p_k will not adopt this suspicion since the message will not pass the authentication test realized in line 16, A2. The Byzantine process p_f can, otherwise, add $m.id$ to $intern\_susp_f[p_i]$ and certify this information. Nonetheless, there are at most f_i faulty processes in the system and the predicate in line 39, A2 is never satisfied. Thus, the stable known process p_j will not invoke *AddInternalSusp* (p_i, m) in line 40, A2, and the lemma follows.

Lemma 6: Let $p_i, p_j \in$ KNOWN $\cap$ STABLE. If there is a message m, such that $m.id \in intern\_susp_j[p_i]$, then, eventually, p_j will invoke *AddMistake* (p_i, m).

Proof: Two cases are possible.
CASE 1: Process $p_j \in E_i^t$. Since p_i is stable known and channels are fair-lossy and authenticated, eventually p_j receives m from p_i (duly certified and formed) (line 17, A1). From the hypothesis of lemma, $m.id \in intern\_susp_j[p_i]$, thus, from lines 18, 34, 37 A1, since p_i is stable known, p_j will call AddMistake (p_i, m) in line 38, A1.

CASE 2: Process $p_j \notin E_i^t$. By a similar argument used in Lemma 5, $p_i \notin known_j$. Thus, some other stable known process $p_k \in$ KNOWN $\cap$ STABLE has raised the suspicion over p_i regarding the reception of $m.id$; that is, there is a $p_k \in E_i^t$ such that $m.id \in intern\_susp_k[p_i]$. Since the network has Byzantine coverage (Assumption 1), then there is a journey $J : p_0 = p_k, p_1, \ldots, p_l, p_j$ between p_k and p_j composed by stable known nodes. If there is more than one journey, than take the one with minimum distance. Let us prove, by induction on the length of J, that, eventually, each $p_l \in J$ invokes *AddMistake* (p_i, m), and thus $m.id \in mistake_l[p_i]$.
(1) If $|J| = 0$, J has only $p_k = p_j$. For the same argument of Case (1), the affirmation holds.
(2) If $|J| > 0$, by induction hypothesis, the affirmation is true for the journey $J \setminus \{p_j\}$ between p_k and p_l, such that, eventually, $m \in mistake_l[p_i]$. When this happens, p_l broadcast a SUSPICION message M with m duly certified in $mistake_l[p_i]$. Since channels are fair-lossy, at some point in the future, p_j receives M in line 17, A1. Since p_l is stable known, M is duly certified and formed. Thus, for the execution of line 18, A1 and lines 14 and 27-31, A2, p_j calls *ValidateReceived*(p_i, m). Since p_i is stable known, m is duly certified and formed. Since m was passed by p_l (m *was not sent directly* by p_i), predicate of line 37,A1 is

satisfied and p_j calls *AddMistake* $(p_i, m.id)$ in line 38, A1 and the affirmation holds. The lemma thus follows.

Lemma 7: Let $p_i \in$ KNOWN $\cap$ STABLE. Assume that $Byz\mathcal{RP}\ ^t(p_i)$ holds for p_i at time t. Eventually, $\forall p_j \in$ KNOWN $\cap$ STABLE is such that $p_i \notin output_j$.

Proof: From Lemma 4, from some $t' \geq t$, eventually $p_i \notin output_j$. From Lemma 5, p_j does not add p_i to $output_j$ in a call to AddInternalSusp (p_i, m). For every message m' required by $\mathcal{A}$ before t, it is possible that $m' \in intern\_susp_j[p_i]$. But, from Lemma 6, at some point in the future, p_j calls AddMistake (p_i, m'); thus, for line 8, A2, eventually $intern\_susp_j[p_i] = \emptyset$. From Lemma 4, there is no pair $\langle p_i, - \rangle$ in $byzantine_j$; thus, p_i is removed from $output_j$ in line 10, A2 and the lemma holds.

Theorem 1: Algorithms 1 and 2 implement a Byzantine FD of class $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$, assuming a network of KNOWN nodes that satisfies Assumptions of Sections II and III .

Proof: The theorem follows from Lemmata 2, 3 and 7.

VI. CONCLUSION

This paper presented a Byzantine failure detector of class $\Diamond\mathcal{S}^M(Byz, \mathcal{A})$ with two innovative features that favor the scalability and adaptability: (i) it is suitable for dynamic distributed systems in which the membership is unknown and (ii) it does not rely on timers to detect omission failures. As a future work, we plan to (i) extend the protocol to tolerate node mobility and (ii) implement the protocol for performance evaluation.

ACKNOWLEDGMENT

Fabíola Greve would like to thank CAPES/Brasil and FAPESB/Bahia/Brasil for the financial support of her research.

REFERENCES

[1] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[2] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[4] D. Malkhi and M. Reiter, "Unreliable intrusion detection in distributed computations," in *Proc. 10th Computer Security Foundations Workshop*. Rockport, MA: IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 116–124.

[5] A. Doudou and A. Schiper, "Muteness detectors for consensus with byzantine processes," in *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1998, p. 315.

[6] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Byzantine Fault Detectors for Solving Consensus," *The Computer Journal*, vol. 46, no. 1, pp. 16–35, 2003.

[7] R. Baldoni, J.-M. Hèlary, M. Raynal, and L. Tangui, "Consensus in byzantine asynchronous systems," *Journal of Discrete Algorithms*, vol. 1, no. 2, pp. 185 – 210, 2003.

[8] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *1st ACM Work. on Wireless Security*. New York, NY, USA: ACM, 2002, pp. 21–30.

[9] A. Haeberlen, P. Kuznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Oct 2007.

[10] A. Haeberlen and P. Kuznetsov, "The Fault Detection Problem," in *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS'09)*, Dec. 2009.

[11] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous Implementation of Failure Detectors," in *Int. Conf. on Dependable Systems and Networks*. Los Alamitos, CA, USA: IEEE Computer Society, 2003, p. 351.

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

[13] A. Mostefaoui, M. Raynal, C. Travers, S. Patterson, D. Agrawal, and A. El Abbadi, "From Static Distributed Systems to Dynamic Systems," in *24th Symp. on Reliable Distributed Systems*, 2005, pp. 109–118.

[14] M. Lima, F. Greve, L. Arantes, and P. Sens, "The time-free approach to byzantine failure detection in dynamic networks," in *WRAITS - 5th Workshop on Recent Advances in Intrusion-Tolerant Systems, with DSN - Int. Conf. on Dependable Systems and Networks*, 2011.

[15] M. K. Aguilera, "A Pleasant Stroll through the Land of Infinitely Many Creatures," *ACM SIGACT News*, vol. 35, no. 2, pp. 36–59, June 2004.

[16] J. R. Douceur, "The sybil attack," in *Revised Papers from the First Int. Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[17] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," University of Ottawa, Tech. Rep., 2011.

[18] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, 1978.

[19] C.-Y. Koo, "Broadcast in radio networks tolerating byzantine adversarial behavior," in *23rd ACM Symp. on Principles of distributed computing*. New York, NY, USA: ACM, 2004, pp. 275–282.

[20] V. Bhandari and N. H. Vaidya, "Reliable local broadcast in a wireless network prone to byzantine failures," in *DIALM-POMC*, 2007.

[21] B. Schneier, *Applied Cryptography (2nd ed.)*. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[22] E. Jiménez, S. Arévalo, and A. Fernández, "Implementing unreliable failure detectors with unknown membership," *Inf. Process. Lett.*, vol. 100, no. 2, pp. 60–63, 2006.

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

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

[25] ——, "Reliable broadcast in radio networks with locally bounded failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 801–811, 2010.

[26] A. Pelc and D. Peleg, "Broadcasting with locally bounded byzantine faults," *Inf. Process. Lett.*, vol. 93, no. 3, pp. 109–115, 2005.