Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants

Pierre Sens, Luciana Arantes, Mathieu Bouillaguet

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

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

Pierre Sens — Luciana Arantes — Mathieu Bouillaguet

N° 6088

Janvier 2007

Thème COM

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

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

Pierre Sens , Luciana Arantes , Mathieu Bouillaguet

Thème COM — Systèmes communicants
Projets Regal

Rapport de recherche n° 6088 — Janvier 2007 — 21 pages

Résumé : Cet article aborde le problème de la détection de fautes dans les réseaux dynamiques de type MANET. Les détecteurs de fautes non fiables fournissent des informations sur la processus défaillants. Ils permettent de résoudre le consensus dans les réseaux asynchrones. Cependant, la plupart des détecteurs considèrent un ensemble connu de processus interconnectés par un réseau complètement maillé. Des telles hypothèses ne sont pas réalistes dans les environnements dynamiques. Généralement, les implémentations des détecteurs reposent sur des temporisateurs dont les bornes sont particulièrement difficiles à déterminer dans des contextes dynamiques. Cet article présente une implémentation asynchrone de détecteurs de défaillances adaptée aux environnements dynamiques. Notre détecteur est une extension de l'algorithme de requête-réponse précedemment proposé par Mostefaoui, Mourgaya et Raynal [MMR03] qui ne repose sur aucun temporisateur. Nous supposons que les identités et le nombre des nœuds sont inconnus. Nous prouvons que notre algorithme permet d'implémenter des détecteurs de fautes de classe $\Diamond S$ et présentons des simulations afin de valider notre approche.

Mots-clés : Pas de motclef

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

Asynchronous Implementation of Failure Detectors with partial connectivity and unknown participants

Abstract: We consider the problem of failure detection in dynamic network such as MANET. Unreliable failure detectors (FD) are classical mechanisms providing information about process failures. They allow to solve consensus in asynchronous network. However, most of implementations consider a set of known processes fully connected by reliable links. Such an assumption is not applicable in dynamic environments. Furthermore, the majority of current failure detector implementation are timer-based ones while in dynamic networks there is not an upper bound for communication delays. This paper presents an asynchronous implementation of a failure detector for dynamic environments. Our implementation is an extension of the query-response algorithm previously proposed by Mostefaoui, Mourgaya and Raynal [MMR03] which does not rely on timers to detect failures. We assume that neither the identity nor the number of nodes are initially known. We also proof that our algorithm can implement failure detectors of class $\diamondsuit S$ when some behavioral properties are satisfied by the underlying system. Simulation results have validated our approach.

Key-words: No keywords

1 Introduction

Introduced by Chandra and Toueg [CT96], unreliable failure detectors (FD) are classical mechanisms which offer information about process failures. As they allow to solve deterministically the consensus problem in asynchronous networks, they are considered as a basic block for building reliable distributed applications on top of such networks. Many distributed applications use a failure detection service in order to solve the consensus problem [VCF00, DT00].

An unreliable failure detector can be informally considered as a per process oracle, which periodically provides a list of processes that it currently suspects of having crashed. It is unreliable in the sense it can make mistakes. Two properties characterize a failure detector : *completeness* and *accuracy*. Roughly, *completeness* sets requirements with respect to crashed processes, while *accuracy* restricts the number of false suspicions.

This paper focuses on unreliable failure detector for dynamic mobile Ad-hoc Networks (MANET). This kind of network presents the following properties : (1) a node does not necessarily known all the nodes of the networks. It can only send messages to its neighbors i.e., those nodes that are within its transmission range. The concept of range models for instance homogeneous radio communication in MANET networks ;(2) the network is not fully connected. Thus, a message sent by a node maybe is routed through a set of intermediate nodes until reaching the destination node ; (3) message transmission delay between nodes is highly unpredictable ; (4) a node can move around and might change its neighborhood. Since in such dynamic environment it is not feasible to maintain a fixed routing infrastructure, message *flooding* is an effective mechanism for communicating between nodes.

Most of current implementations of failure detectors are based on all-to-all communication where each process periodically sends a *heartbeat* message to all processes [LFA00, SM01, DT00]. At the same time, every process suspects all those processes from which it did not receive any *heartbeat* message for a given timeout. As they usually consider a set of known nodes that are fully connected by reliable links, these implementations are not suitable for dynamic environments. Furthermore, they usually need to rely on additional synchrony assumptions. In other words, even if initially they assume no bound for message transmission delay, they assume that some bound will permanently hold eventually. Such an assumption is not suitable for dynamic environments where communication delays between two nodes can vary due to nodes' mobility. In fact, failure detector for dynamic networks must be able to efficiently distinguish failure from mobility.

In [MMR03], Mostefaoui, Mourgaya, and Raynal have proposed a not timer-based implementation of failure detectors. It is an asynchronous implementation based on a *query-response* mechanism, which uses just the value of f, the maximum number of processes that can crash, and Π, the total number of known nodes. Nevertheless, the authors' computation model consists of a set of known processes Π fully connected by reliable links which is not applicable to dynamic environments.

This paper presents an extension of the above asynchronous FD implementation adapted for dynamic environments. Neither the identity of other nodes nor the number of nodes are initially known by a node. Channels are considered to be reliable. The network is not fully

connected presenting the failure resilient coverage property i.e., despite of failures there is always a path between two nodes.

The basic idea of our approach is the flooding of failure suspicion information over the network. Initially, each node just knows itself. Then, it periodically diffuses a *QUERY* message to its neighbors (we assume that a node can broadcast a message to its neighbors without knowing their identity). Upon receiving such message, the neighbor sends back to the former a *RESPONSE* message which includes its identity and both its current list of suspected processes and the possible mistakes that it previously made. Based on this simple mechanism, we propose an algorithm for building unreliable failure detectors for dynamic networks. Firstly, we have considered that nodes do not move around. We show then that if the underline distributed system satisfies some defined behavioral properties, such algorithm can satisfy the strong completeness and eventually weak accuracy of failure detectors from class $\diamondsuit S$. We finally proposes an extension of the above algorithm which supports nodes' mobility.

The rest of the paper is organized as follows. Section 2 presents Chandra-Toueg's failure detectors. Section 3 defines the computation model, introducing the failure resilient coverage property. Section 4 describes our asynchronous failure detector algorithm. Section 5 presents some behavioral properties that allow our algorithm to implement failure detectors $\diamondsuit S$. A sketch of proof of our algorithm is also presented in this section. An example of the execution of our algorithm is given in section 6. Section 7 describes how the algorithm can be extend to support nodes' mobility. Simulation performance results are shown in section 8 while some related work are briefly described in section 9. Finally, section 10 concludes the paper.

2 Chandra-Toueg's Failure detectors

Unreliable failure detectors introduced by Chandra and Toueg [CT96] provide information about the aliveness of other processes. Each process has access to a local failure detector which outputs a list of processes that it currently suspects of having crashed. A local 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. Therefore, a detector may repeatedly adds and remove the same process from its list of suspect processes. It can thus make an infinity number of mistakes.

Failure detectors are abstractly characterized by two properties : *completeness* and *accuracy*. *Completeness* characterizes the failure detector capability of suspecting every incorrect process permanently. *Accuracy* characterizes the failure detector capability of not suspecting correct processes. Two kinds of completeness and four kinds of accuracy are defined in [CT96], which once combined yield eight classes of failure detectors.

Our work is focused on the class of *Eventually Strong* $\diamondsuit S$ detectors. This class contains all the failure detectors that satisfies the following properties :

– **Strong completeness** : there is a time after which every process that crashes is permanently suspected by every correct process.

– **Eventual weak accuracy** : there is a time after which some correct processes are not suspected by any correct process.

3 Model

We consider a system consisting of a finite set Π of processes. Contrary to the usual model, processes in Π do not necessarily know each other. A process can fail by crashing. Processes communicates by sending messages. The network is not fully connected and we assume that there is at least on hop. A communication range is associated to each process. A process can broadcast a message in its range without knowing the processes of its range. The message is then delivered in a reliable way to all correct processes belonging to the range. The words node and process are interchangeable.

Initially, we assume that processes cannot move, ie. they cannot change of range. In section 7, we relax this assumption.

Definition 1. Range : $\forall i \in \Pi$, $range_i \subseteq \Pi$ *defines the set of nodes which can directly receive and send a message to* i.

We can state the following properties about ranges :

Property 1. *ranges are reflexive :* $\forall i \in \Pi$, $p_i \in range_i$

Property 2. *ranges are symmetric :* $\forall i, j \in \Pi$, $i \neq j$, $j \in range_i \Rightarrow i \in range_j$

Considering the set of ranges of the network, we that assume there is always a path between each pair of process in spite of failures. We call a network having this property a *f-covering* network.

Definition 2. f-covering network : *Let* f *be the maximum number of processes that can crash. A network* $\mathcal{R}$ *linking processes in* Π *is* f*_covering iff :*

$\forall i, j \in \Pi, \exists S_1, \ldots, S_n \subseteq \Pi : \forall k \in 1, \ldots, n; |S_k| > f$ *and :*
$\forall\ x \in S_1,\ x \in range_i$
$\forall\ x \in S_{k+1}, \forall y \in S_k,\ x \in range_y \quad \forall k \in 1, \ldots, n-1$
$\forall\ x \in S_n,\ j \in range_x$

We say that two nodes x, y are range adjacent if they cannot communicate directly, but can exchange messages through one intermediate node despite of failures. In other words, there are at least $f + 1$ nodes in the intersection of the two ranges.

Definition 3. Range adjacent nodes : $x, y \in \Pi$ *are range adjacent if* $x \notin range_y$ *and* $|range_x \cap range_y| > f$

The size of the smallest range set in the network is called the range density, and it is noted d.

Definition 4. Range density : $d \stackrel{def}{=} min(|range_i|), \forall i \in \Pi$

In a *f-covering* network we distinguish the set of nodes belonging to the range intersection of range adjacent nodes. These nodes have a particular role since they are used to route information between nodes. These sets belong to the $\mathcal{IS}$ set defined as follows :

Definition 5. Intersection set $\mathcal{IS}$:
$\mathcal{IS} \stackrel{def}{=} s \subseteq \Pi : \forall x \in s, \exists a, b \in \Pi$ *which are range adjacent nodes and* $x \in range_a \cap range_b$

Figure 1 illustrates a 1-covering network with a range density equal to 3 :
$range_1 = range_2 = \{1, 2, 3, 4\}$
$range_3 = range_4 = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$range_5 = range_6 = \{3, 4, 5, 6, 7, 8\}$
$range_7 = range_8 = \{3, 4, 5, 6, 7, 8, 9\}$
$range_9 = \{7, 8, 9\}$.
$\mathcal{IS} = \{\{3, 4\}, \{7, 8\}\}$.

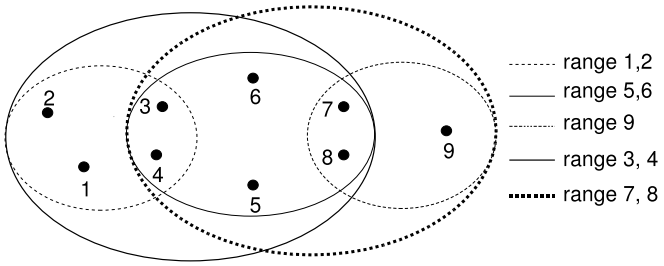


FIG. 1 – Example of f-covering network with f=1

4 Asynchronous algorithm

We assume a *f-covering* network whose range density d is greater the the maximum number of faults f (i.e., $d > f$). Figure 1 describes our protocol, which is based on a *query-response* mechanism.

We use the following notations :
- rec_from_i denotes the set of $(d - f)$ processes from which p_i has received responses to its last QUERY message.
- $suspected_i$ denotes the current set of processes suspected by p_i.
- $mistake_i$ denotes the current set of processes that were previously suspected by p_i but which are currently not suspected by p_i.
- rec_i^j denotes the last rec_from_j set received by p_i from p_j.
- mis_i^j denotes the last $mistake_j$ set received by p_i from p_j.
- sus_i^j denotes the last $suspected_j$ set received by p_i from p_j.
- K_i denotes the current knowledge of p_i. K_i is the set of processes from which p_i has received a message (QUERY or RESPONSE) since the beginning.
- R_i denotes the sets of the rec_from sets received by p_i from processes in rec_from_i.
- S_i denotes the sets of the $suspected$ sets received from processes in rec_from_i.
- M_i denotes the sets of the $mistake$ sets received from processes in rec_from_i.

For simplicity's sake, we are not going to show that K_i is updated whenever p_i receives a RESPONSE message.

Algorithm 1 – Asynchronous implementation of failure detectors

1: **init :**
2: $suspected_i \leftarrow \varnothing$
3: $mistake_i \leftarrow \varnothing$
4: $K_i \leftarrow \{p_i\}$

5: **Task** T1 :
6: **loop**
7: **foreach** p_j in $range_i$ **do** *send* $QUERY()$ to p_j **enddo**
8: **wait until** ($RESPONSE()$ received from $(d-f)$ distinct processes) ;
9: $R_i \leftarrow \cup_{x \in rec\_from_i} \quad rec_i^x$
10: $S_i \leftarrow \cup_{x \in rec\_from_i} \quad sus_i^x$
11: $M_i \leftarrow \cup_{x \in rec\_from_i} \quad mist_i^x$
12: $old\_suspected_i \leftarrow suspected_i$
13: $suspected_i \leftarrow (S_i \cup K_i) \setminus R_i \setminus M_i$
14: $mistake_i \leftarrow old\_suspected_i \setminus suspected_i$
15: **end loop**

16: **Task** T2 :
17: **upon reception of** $QUERY$ **from** p_j **do**
18: **if** it is the first time a $QUERY$ is received **then**
19: send $RESPONSE(rec\_from_i, suspected_i, mistake_i)$ **to** $range_i$
20: **else**
21: send $RESPONSE(rec\_from_i, suspected_i, mistake_i)$ **to** p_j
22: **end if**

The algorithm is composed of two tasks.

Task $T1$ is made up of an infinite loop. At each step of the loop, a QUERY message is reliably sent to all nodes of p_i's range. Then, p_i waits for $d-f$ responses (lines 7-8). Since each range contains at least d nodes, p_i will eventually receive $d-f$ responses. At each step, p_i accumulates the rec_from, $suspected$, and $mistakes$ sets received from the first $d-f$ nodes which respond to p_i's QUERY in its local variables R_i, S_i, and M_i respectively. R_i variable gathers the information given by the nodes that respond to p_i about currently alive nodes they are aware of whereas S_i and M_i gather information from these same nodes about current sets of suspected crashed nodes and mistakes respectively. Based on these sets, p_i updates its suspected set (line 13). The latter is the union of the set of suspected nodes received from remote nodes and the set of nodes known by p_i ($x \in S_i \cup K_i$), excluding those nodes that currently responded to some QUERY ($x \notin R_i$). Naturally, mistakes are removed from the suspected set. Finally, p_i updates its mistake set (line 14) : a mistake node m is a node which was previously suspected ($m \in old\_suspected$) and is currently not suspected ($m \notin suspected$).

Task $T2$ is simple : whenever a node receives a QUERY message from another one of its range, it sends back to the querying node a RESPONSE message (line 21). However, if it is the first time a QUERY is received, the node broadcasts a RESPONSE message to all the nodes of its range to be sure that at least $f+1$ processes will learn that it exists (line 19).

5 Implementation of a failure detector of class $\diamondsuit S$

Inspired by the behavioral properties introduced in [MMR03] and [Gaf98], this section starts by presenting two behavioral properties that, when satisfied by the underlying *f-covering* dynamic network, allow to implement failure detector of class $\diamondsuit S$. Then, we give a sketch of proof that our algorithm can implement a failure detector of class $\diamondsuit S$ when the above properties are satisfied.

5.1 Behavioral Properties

Responsiveness Property (RP) : this behavioral property denotes the ability of a node to reply to a query among the first $d-f$ nodes.

Let $t \in \mathcal{T}$ and f^t be the set of processes that have crashed at time t. Considering a time u, the **RP** property of process p_i is defined as follows :

Property 3.

$$\mathcal{RP}(p_i) \stackrel{def}{=} \exists u \in \mathcal{T} : \forall t > u, \forall j \in range_i,\ p_i \in rec\_from_j$$

Intuitively, $RP(p_i)$ property states that after a finite time u, p_i always reply to the queries it receives among the first $d-f$ nodes.

Propagation Property (PP) : This second property is needed to assure that suspected and mistake information included in RESPONSE messages reaches all the nodes of the network. To this end, it is necessary that at least one node verifies the **RP** property in each intersect set of $\mathcal{IS}$. The **PP** property is defined as follows :

Property 4.

$$\mathcal{PP} \stackrel{def}{=} \forall s \in \mathcal{IS}, \exists p_i \in s : \mathcal{RP}(p_i)$$

5.2 Sketch of Proof

We present in this section a sketch of proof of both the strong completeness and eventual weak accuracy properties that characterize failure detectors of class $\diamondsuit S$.

5.2.1 Strong completeness

We consider a process p_f that crashes at time t. To satisfy strong completeness, we must prove that if p_i is correct then eventually p_f remains permanently in $suspected_i$.

We assume that p_f interacted with other processes within its range at least one time before it crashes. Without this assumption, no process could be aware of the existence of p_f. Then, all correct processes within $range_f$ will include p_f in their K set variable. We note r_c this set.

As p_f crashed, there will be a time $t' > t$ after which all processes in r_c will not receive a RESPONSE message from p_f anymore (i.e., $p_f \notin R$ sets of processes within r_c). Consequently, they will include p_f in their suspected sets (line 13 of the algorithm).

According to the PP property, eventually at least one process in each set of $\mathcal{IS}$ included in r_c will propagate the information about the crash of p_f outside the range of r_c. Applying this same principle to the other ranges, this information will be propagated to all correct processes. These processes will add p_f in their suspected set variable. Then, since p_f will never be included in a rec_from set of any correct process, it will be never removed from the suspected set of such processes. □

5.2.2 Eventual weak accuracy

Considering a correct process p_i, we must prove that there is a time u after which p_i is not included in the *suspected* variable of any correct process.

We consider that p_i satisfies the RP property. If the network has at least one set in $\mathcal{IS}$ then according to PP property, such a process p_i exists. Then, after a time t, p_i is always included in the rec_from sets of all nodes in $range_{p_i}$.

If p_i was not suspected before t, it will not be suspected after t since all processes in $range_{p_i}$ include it in their R sets (line 9) and as consequence remove it from their *suspected* sets (line 13).

We consider now that p_i was suspected before t by some correct (possibly all) process. After t, p_i will be inserted in the R sets of all processes within its range. Then, these processes remove p_i from their *suspected* set. All processes which previously suspected p_i, will include

p_i in their mistake set M. The mistake sets will then be propagated in the same way the suspected sets are propagated (see previous section).

When a process p_j receives a message including p_i as a mistake, the latter is removed from $suspected_j$ (line 13). It may happen that before t no process in $range_i$ suspected p_i, which in fact was suspected only by processes belonging to "remote" ranges. This case is possible only if p_i was previously suspected by processes belonging to $range_i$ and the corresponding mistake messages are still transiting. These messages will be propagated through reliable path and eventually, all processes from the mentioned "remote" ranges will remove p_i from their suspected set. □

6 Example of failure detection

Figure 2 illustrates an execution of the algorithm of figure 1. We consider a *1-covering* network whose density is equal to 3 and whose nodes 2 and 5 present the *Responsiveness Property* ie., they always belong to the first $d-f$ processes responding to a QUERY message (a). In step (b), node 1 fails and does not respond to nodes 2 and 3. Thus, R_2 and R_3 sets do not include node 1 which will be added to S_2 and S_3 sets. At step (c), node 2 sends S_2 to nodes 4 and 5 in response to their QUERY messages. Upon receiving S_2, these nodes add node 1 to their S_4 and S_5 sets respectively. Finally in step (d), node 6 updates its suspected set S_6 when receiving S_5 in response to its query.

7 Extension for mobility management

We extend the initial algorithm of figure 1 to manage node mobility. When a node moves around it may change range.

We make the assumption that despite the move, the *f-covering* property of the network is maintained and that the range density of the network does not change. We also assume that PP property always holds.

For the explanation of the algorithm we consider that $suspected_m$ message is a RESPONSE message that includes m as a suspected node and $mistake_m$ is a RESPONSE message that corrects the suspicion that m has crashed.

When a node m moves to another range, it will start being suspected by correct nodes of its source's range since it cannot reply to $QUERY$ messages anymore. Therefore, a $suspected_m$ message will be reliably propagated to all nodes of the network. Eventually, nodes of the new range of m will receive such message. Upon receiving it, the latter will correct the mistake, propagating then a $mistake_m$ message.

For avoiding a "ping-pong" effect between *suspected* and *mistake* messages, we have modified the algorithm of figure 1 as follows : when a node x which suspected m receives a $mistake_m$ message from a remote range, if m belongs to x's knowledge set (ie., m was a neighbor of x, included then in K_x), m is removed from x'knowledge set K_x. To detect *mistake* messages issued from remote ranges, we have added to *mistake* message' headers

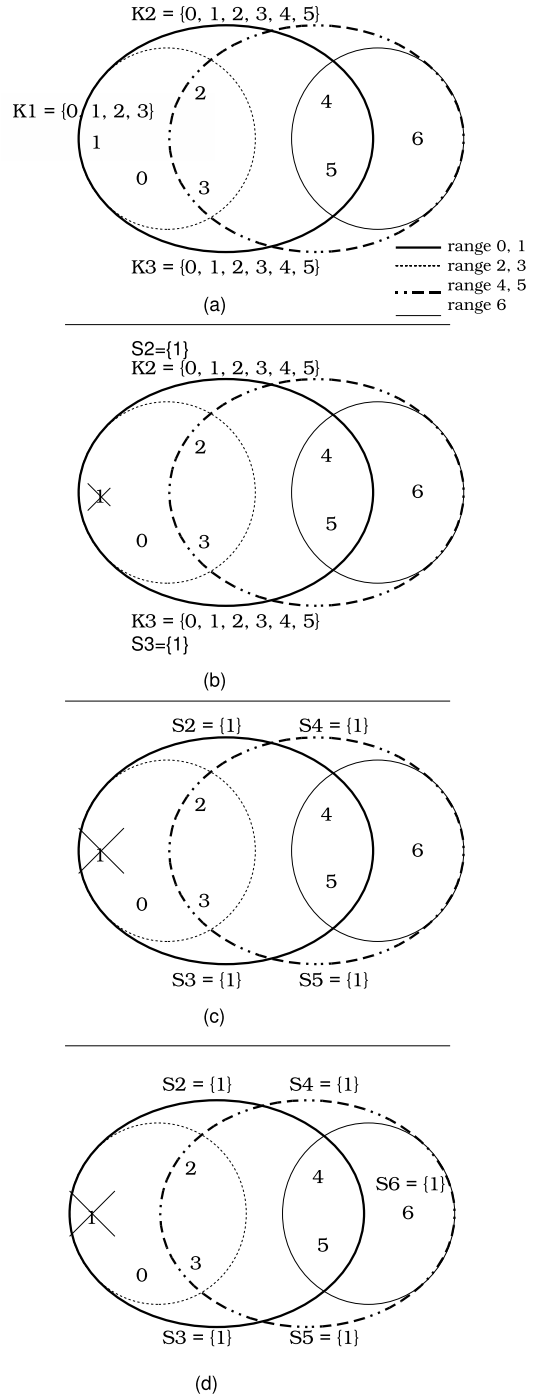


FIG. 2 – Example of failure detection

the identifier of the node that detects the mistake. Then, the receivers of *mistake* messages have just to check if the former belongs to its knowledge set to know if the *mistake* message was issued from a remote range or not.

Algorithm 2 adapts our previous algorithm for mobility. Only task T1 has been modified (lines 14-16).

Algorithm 2 – Asynchronous implementation of failure detectors with mobility

```
1:  init :
2:  suspected_i ← ∅
3:  mistake_i ← ∅

4:  Task T1 :
5:  loop
6:      foreach j in range_i do send QUERY() to p_j enddo
7:      wait until RESPONSE received from (d − f) distinct processes ;
8:      R_i ← ∪_{x∈rec_from}  rec_i^x
9:      S_i ← ∪_{x∈rec_from}  sus_i^x
10:     M_i ← ∪_{x∈rec_from}  mist_i^x
11:     old_suspected ← suspected_i
12:     suspected_i ← (S_i ∪ K_i) \ R_i \ M_i
13:     mistake_i ← old_suspected \ suspected_i
14:     foreach p ∈ M_i and p ∈ suspected_i do
15:     if mistake_sender(p) ∉ K_i and p ∈ K_i then
16:         K_i = K_i − {p}
17:     end if
18:     enddo
19: end loop
```

Figure 3 shows an example of the algorithm execution where node 2 moves to the node 7's range. Initially in (a), node 2 knows 1, 2, 3 and 4. When node 2 moves (b), it starts eventually suspecting nodes 1, 3 and 4 since the latter cannot response anymore to node 2's QUERY message. Similarly, nodes 1, 3 and 4 which had 2 in their range start suspecting 2. In (c), the suspected lists information is propagated until it reaches nodes which have received suspected message to their last queries. Then, mistake messages will be generated by those nodes and propagated over the network. We have considered that just after receiving S_2, nodes 5 and 6 receive a response from 3 and 4. Thus, they will remove 3 and 4 from their suspected sets and add the latter in M_5 and M_6. Similarly, when nodes 5 and 6 receive S_3 and S_4, they will add 2 in their mistake sets. On the other hand, when nodes 3 and 4 receive S_6 and S_5, they will remove 1 from their suspected sets.

Finally, node 1 (respectively node 2), receives the suspected set from nodes 3 or 4 (resp. 5 or 6). Since the suspected information is issued from a remote range, node 1 (resp. 2) removes 2 (resp. 1, 3, 4) from its (resp. their) knowledge set (resp. sets) (d).

8 Evaluation

In order to analyze the behavior of our algorithm, we develop a simulator for $f-$*covering* networks. A set of experiments were conducted for evaluating both completeness and accuracy properties. We use the following metrics for such evaluation [CSA02] :

- **failure detection time** : time interval between the instant the failure takes place and the instant the failure is permanently detected by all correct processes ;
- **propagation of "false" failure suspicion** : number of nodes that falsely suspected a node ;
- **mistake duration** : time it takes for a mistake to be permanently corrected by all nodes that falsely suspected it.

Our simulator uses a discrete time. At each time step, correct processes sends QUERY messages to process within their range. We assume a constant message transmission delay equals to one unit of time. In other words, every message sent by a process of a range r at time t is received by all r's correct processes at time $t + 1$.

We have considered two *f-covering* network configurations as shown in Figure 4. The first one models a linear *1-covering* network i.e, a *1-covering* configuration which presents the highest diameter for a given number of nodes. The average failure detection time of linear *f-covering* networks can be considered as an upper bound for all other *f-covering* network configurations with the same number of nodes since the latter present smaller diameters. The second configuration models a *1-covering* star network having 2 nodes which are included in all ranges.

8.1 Completeness

In order to evaluate completeness property of our failure detector, we have measured the impact of failure detection time on network size, network position of a failure, and message transmission delay.

Impact of network size : The above described two *1-covering* network configurations were considered. The number of nodes varies from 20 to 164. We have injected one failure in the first range. Figure 5 presents such simulation. We can observe a quite linear behavior of detection time for the *linear* network. Indeed, this time depends on the propagation delay of response messages which include information about suspected nodes. On the other hand, failure detection time is independent of the number of nodes for the *star* network configuration, as expected.

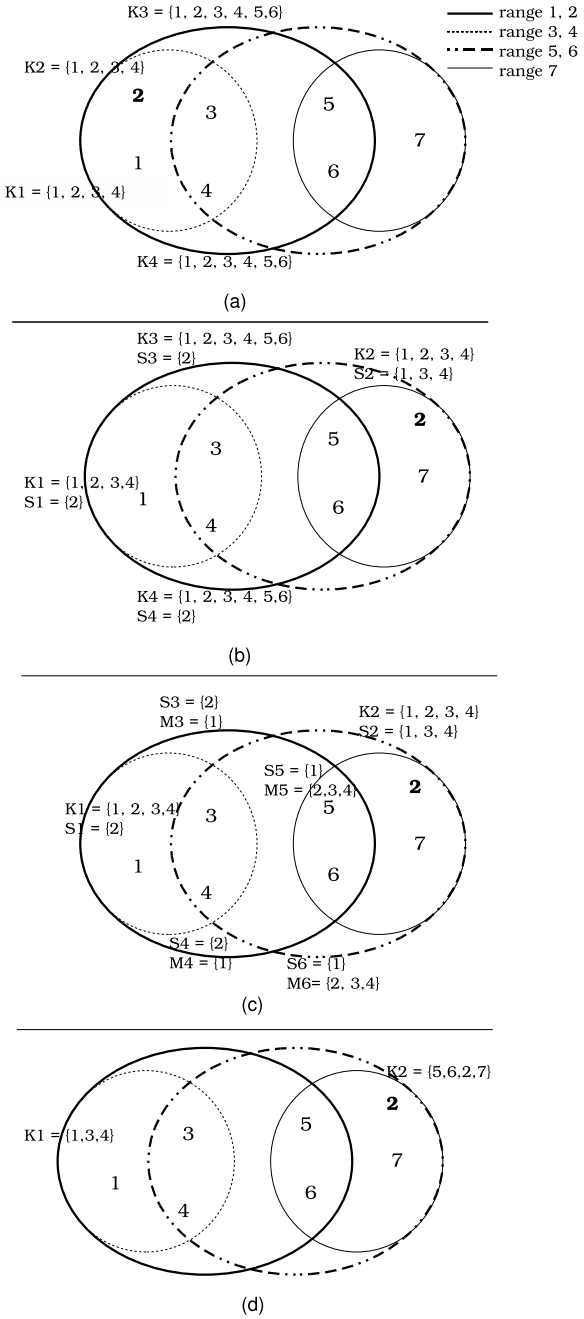
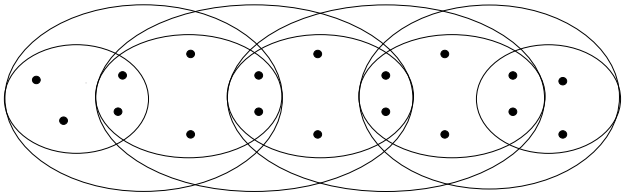
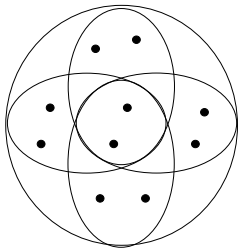


FIG. 3 – Example of node mobility



linear 1-covering network



star 1-covering network

FIG. 4 – Network 1-covering configurations

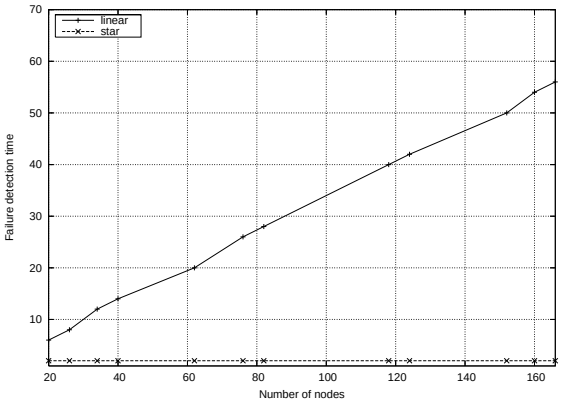


FIG. 5 – Detection time of one failure as a function of network size

Impact of the network position of a failure : We considered a *linear 1-covering* network composed of 34 nodes whose diameter is 5 hops i.e., any message sent by a node travels along at most 5 hops in order to reach its destination. We have injected one failure per experiment but in a different location of the network at each experiment. We have then observed the propagation of response messages which include the node failure's suspicion. It allowed us to express the failure detection time as a function of the position of the faulty node in the network. As shown in 6, failures in the middle of network are fast detected by all correct nodes while those belonging to ranges near the boundary of the network are slowly detected.

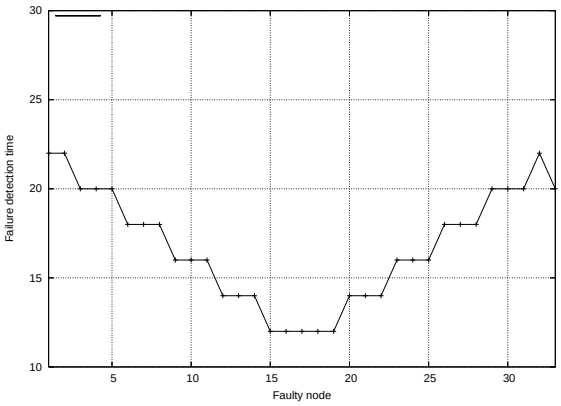


FIG. 6 – Detection time as a function of the position of the faulty node

Impact of message transmission delay : For such study, we have distinguished two kinds of nodes : *speed nodes* which spend just one unit of time to respond to a query request, and *slow nodes* which spend three units of time to respond to the same query. Figure 7 shows a *linear 1-covering* network configuration with 34 nodes where *slow nodes* are uniformly spread over it. For performance measure we have varied the ratio slow nodes/total nodes. We can observe that the slowdown of the failure detection time is not linear. In fact, such time is especially high when the majority of intersection nodes are slow.

8.2 Accuracy

For evaluating accuracy property, we have injected "false" failures on our simulator : one process at time t is frozen and it is waken up k units of time later (at time $t + k$). The considered configuration was the same *linear 1-covering* network with 34 nodes. *Node 17*, a node in the middle of the network, is frozen at time 1 and it is waken up at time 16. Notice that at time 16, *Node 17* is erroneously suspected by all correct nodes. In Figure 8, we can observe that the propagation of response messages which include *Node 17* as suspected takes place from time 1 to 13. As *Node 17* is waken up at time 16, nodes of its range detect their

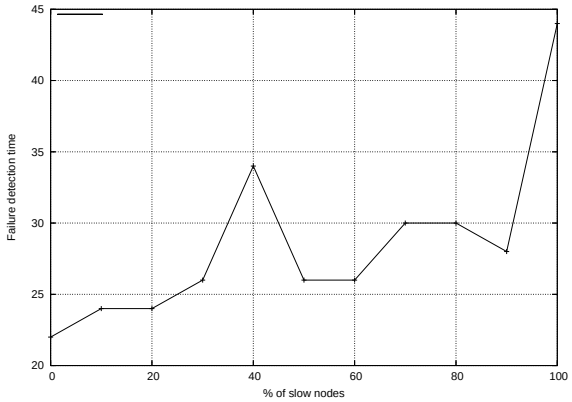


FIG. 7 – Impact of slow nodes on detection time

mistake at time 17 and then diffuse response message including such mistake information. These messages are propagated to all correct processes. The mistake is corrected by the processes and at time 23 *Node 17* is considered to be alive by all process. Mistake duration took 22 units of time.

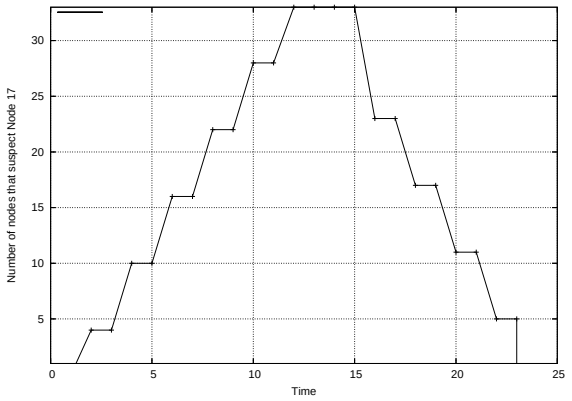


FIG. 8 – Accuracy evaluation

Finally, we have evaluated the accuracy property when one node moves around. To overestimate the mistake duration time, we directly move one node, *Node 1*, from the first range to the last one. Initially, the first range contains 5 nodes (nodes 0 to 4). When finding itself in its new range, *Node 1* will not receive response message from the nodes of its old range (*Node 0*, *Node 2*, *Node 3*, *Node4*). However, since they belong to *Node 1*'s knowledge (K_1), *Node 1* will start suspecting them as being crashed. It will then include in its response message the four nodes as being suspected of crash. Similarly, the four nodes will start suspecting *Node 1* and will diffuse response messages that include the latter as suspected of

crash. When *Node 1* (resp. *Node 0*, *Node 2*, *Node 3*, *Node4*)'s reaches *Node 1*'s old (resp. new) range, nodes of this range detect the mistake and will include it in their respective response messages. Such messages propagation are illustrated in figure 9. The mistake duration took 36 units of time.

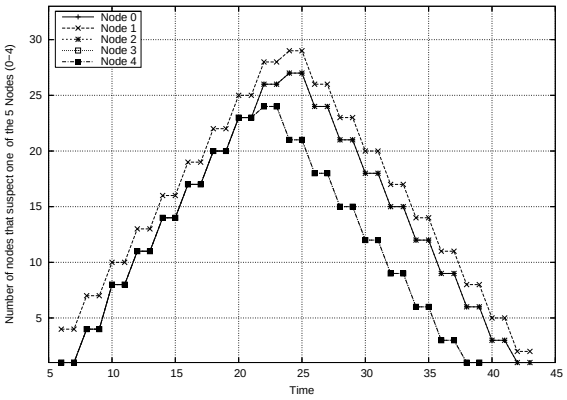


FIG. 9 – Accuracy : impact of moving node 1

9 Related Work

As in our approach, some scalable failure detector implementations do not require a fully connected network. Larrea et al. proposed in [LFA00] an implementation of a failure detector based on a ring arrangement of processes. In their approach the number of messages is linear, however the propagation of failure information is low. In [GCG01], Gupta et al. proposed a randomized distributed failure detector algorithm which balances the network communication load. Each process randomly chooses some processes whose aliveness is checked. Practically, the randomization makes the definition of timeout values difficult. In [BMS03], a scalable hierarchical failure adapted for Grid configurations is proposed. However, the configuration has to be initially known. It is worth remarking that none of these work tolerate node's mobility.

Few implementations of unreliable failure detector found in the literature focus on MANET environments. On the other hand, all of them are timer-based ones. In [FT05], the authors propose a failure detector implementation where a vector is included in every *heartbeat* messages. Each entry of the vector corresponds to a node of the network which stores the highest known *heartbeat* sent from the node in question. When receiving a vector, a node updates its vector to the maximum of its local vector and the former. In order to cope with higher communication delay due to nodes' mobility, the protocol allows a gap of some heartbeats between two adjacent heartbeat arrivals. Contrary to our approach, the authors assume a known number of nodes and that failures include message omissions too.

Sridhar presents in [Sri06] the design of an hierarchical failure detection which consists of two independent layers : a local one that builds suspected list of crashed neighbors of the corresponding node and a second one that detects mobility of nodes across network correcting possible mistakes. Contrary to our approach that allows the implementation of FD of class $\diamondsuit S$, the author's failure detector is an eventually perfect *local* failure detector of class $\diamondsuit P$ i.e., it provides strong completeness and eventual strong accuracy but with regard to a node's neighborhood.

Cavin et al. [CSS05] have adapted the failure detector definition of [CT96] to the case where the participants are unknown in order to solve the problem of reaching agreement in mobile networks where processes can crash. They have introduced the concept of local participant detectors, which are oracles that give the subset of processes that participating in the consensus. Therefore, the accuracy property does not apply to the entire set of nodes but to the set of nodes that each process discovers through its participant detector. The authors construct an algorithm that solves consensus with unknown number of participants based on the assumption that participant detectors satisfy the it One Sink Reducibility (OSR) property. They have also defined a *safe crash pattern*, necessary and sufficient for imposing that the combination of participant and failure detectors satisfies OSR property. The authors have then proved that in this particular case, a perfect failure detector P is required for solving the fault-tolerant consensus with unknown participants problem.

10 Conclusion

This paper has presented a new implementation of failure detector for dynamic environment such as MANET where the number of nodes is not initially known and the network is not fully connected. Our algorithm does not use any timer and is based on an extension of the query-response mechanism proposed by [MMR03]. We assume that network has the *f-covering* property, where f is the maximum number of failures. Such property guarantees that there is always a path between two nodes despite of failures.

Our algorithm can implement failure detectors of class $\diamondsuit S$ when the behavioral properties, which are defined in the article, are satisfied by the underlying system. Basically, we assume that responses from some of the processes of the network always arrive among the $d - f$ first ones (d is the minimum number of neighbors that a node must have).

Assuming that the $f-covering$ property of the network always holds in the presence of node mobility, we also propose an extension of the algorithm where some nodes can move.

Références

[BMS03] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *Proceedings of the International Conference on Dependable Systems and Networks*, San Francisco, CA, USA, june 2003. 18

[CSA02] W. Chen, S.Toueg, and K. Aguilera. On the quality of service failure detectors. *IEEE Transaction of Computers*, 51(1) :13–32, 2002. 13

[CSS05] D. Cavin, Y. Sasson, and A. Schiper. Reaching Agreement with Unknown Participants in Mobile Self-Organized Networks in Spite of Process Crashes. Technical report, 2005. 19

[CT96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 1996. 3, 4, 19

[DT00] B. Devianov and S. Toueg. Failure detector service for dependable computing. In *Proceedings of the First International Conference on Dependable Systems and Networks*, pages 14–15, juin 2000. 3

[FT05] R. Friedman and G. Tcharny. Evaluating failure detection in mobile ad-hoc networks. *International Journal of Wireless and Mobile Computing*, 1(8), 2005. 18

[Gaf98] E. Gafni. Round-by-round fault detectors (extended abstract) : unifying synchrony and asynchrony. In *PODC '98 : Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 143–152, New York, NY, USA, 1998. ACM Press. 8

[GCG01] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 170–179. ACM Press, 2001. 18

[LFA00] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC 2000)*, pages 334–334, NY, July 16–19 2000. ACM Press. 3, 18

[MMR03] A. Mostefaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Proceedings of International Conference on Dependable Systems and Networks*, June 2003. 1, 2, 3, 8, 19

[SM01] I. Sotoma and E. Madeira. Adaptation - algorithms to adaptative fault monitoring and their implementation on corba. In *Proceedings of the IEEE 3rd International Symposium on Distributed Objects and Applications*, pages 219–228, september 2001. 3

[Sri06] N. Sridhar. Decentralized local failure detection in dynamic distributed systems. *The 25th IEEE Symposium on Reliable Distributed Systems*, 0 :143–154, 2006. 19

[VCF00] P. Verissimo, A. Casimiro, and C. Fetzer. The timely computing base : Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, New York City, USA, june 2000. IEEE Computer Society Press. 3

Table des matières

1 Introduction **3**

2 Chandra-Toueg's Failure detectors **4**

3 Model **5**

4 Asynchronous algorithm **6**

5 Implementation of a failure detector of class $\diamondsuit S$ **8**
5.1 Behavioral Properties 8
5.2 Sketch of Proof 9
5.2.1 Strong completeness 9
5.2.2 Eventual weak accuracy 9

6 Example of failure detection **10**

7 Extension for mobility management **10**

8 Evaluation **13**
8.1 Completeness 13
8.2 Accuracy 16

9 Related Work **18**

10 Conclusion **19**

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

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