Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 100 (2006) 60–63
Information Processing Letters
www.elsevier.com/locate/ipl

Implementing unreliable failure detectors with unknown membership [☆]

Ernesto Jiménez [a,*], Sergio Arévalo [b], Antonio Fernández [b]

[a] *EUI, Universidad Politécnica de Madrid, 28031 Madrid, Spain*
[b] *LADyR, GSyC, Universidad Rey Juan Carlos, 28933 Móstoles, Spain*

Received 15 July 2005; received in revised form 24 April 2006; accepted 9 May 2006

Available online 12 June 2006

Communicated by A.A. Bertossi

Keywords: Failure detectors; Crash failures; Membership; Fault tolerance

1. Introduction

Unreliable failure detectors [3] are useful devices to solve several fundamental problems in fault-tolerant distributed computing, like consensus or atomic broadcast. In their original work [3], Chandra and Toueg proposed 8 different classes of unreliable failure detectors, and showed that all of them can be used to solve consensus in a crash-prone asynchronous system with reliable links. All these detectors have at least a property called *Weak Completeness:* eventually, every process that fails is permanently suspected by some correct process. In a follow up work with Hadzilacos [2], they proposed another type of failure detector, Ω, which guarantees that eventually all correct processes permanently choose the same correct process as *leader*. They show in [2] that Ω is the weakest detector that can be used for solving consensus in this type of systems. When the membership is known, an Ω failure detector trivially also implements a $\diamond\mathcal{S}$ failure detector ($\diamond\mathcal{S}$ is one of the 8 original classes).

A number of papers propose algorithms to implement these failure detectors. Most of the effort in these papers has gone to reduce the level of reliability and synchrony of the system. For instance, let us consider systems with crash failures and whose links can be lossy asynchronous (in which messages can be lost or arbitrarily delayed). Aguilera et al. [1] have proposed an algorithm that implements Ω in these systems as long as at least the *outgoing* links from some unknown correct process are eventually timely (in which, once the link is stable, all messages are delivered within an unknown time bound).

Our contributions. To our knowledge, all algorithms proposed implementing failure detectors require every process to know the identity of the rest of processes of the system (the *membership*). In this paper we show that this assumption is not trivial, since we show here that, without it, *no failure detector class with weak completeness can be implemented, even in a fully synchronous system with reliable links.* Since the original 8 classes proposed by Chandra and Toueg have at least weak completeness, none of them can be implemented without membership knowledge. Note that it is very common in today's distributed systems (peer-to-peer, ad-hoc

[☆] Partially supported by the Spanish MEC under grants TIN2005-09198-C02-01, TIN2004-07474-C02-02, and TIN2004-07474-C02-01, and the Comunidad de Madrid under grant S-0505/TIC/0285.

[*] Corresponding author.
E-mail address: ernes@eui.upm.es (E. Jiménez).

0020-0190/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2006.05.009

networks, etc.) to have partial or no knowledge of the membership.

Surprisingly, we show that Ω can be implemented without membership knowledge. This implies, for instance, that an Ω failure detector cannot be transformed into a $\diamond S$ detector anymore. The fact that Ω can be implemented, while classical detector classes cannot, is mainly caused by the different approaches taken. While in Ω it is enough to know one alive process, in classical detectors it is necessary to know every faulty process to provide completeness.

Then, we present here an algorithm that implements an Ω failure detector even if the membership is unknown. Interestingly, our algorithm requires very weak reliability and synchrony assumptions for correctness.[1]

2. Impossibility without membership knowledge

We show in this section that even in a synchronous system, some knowledge of the system membership is required to provide weak completeness. Furthermore, the result holds even if the *number of processes* is known.

Theorem 1. *Let S be a synchronous system with reliable links in which at least one process can crash. If there is some process in S such that the rest of processes have no knowledge whatsoever of its identity, there is no algorithm that implements a failure detector with weak completeness in S.*

Proof. Let us assume there is an algorithm A that implements a failure detector with weak completeness in these systems. Let us consider first a system S_0 with a process q whose identity is unknown for the rest of processes. Consider a run R_0 of A in S_0 in which q fails at time 0 without sending any message (and hence it will be eventually suspected). Let p be a process that is not in system S_0. Clearly, in R_0, p cannot be in any of the lists of suspected processes. Let us consider now a system S_1 obtained from S_0 by replacing process q with process p, whose identity is also unknown for the rest of processes. Consider a run R_1 of the algorithm A in S_1 in which process p fails at time 0 without sending any message, and the rest of processes behave like in R_0. Note that this can be so because no process knows the identity of p and q. Then, no process suspects p in R_1 and weak completeness is not achieved. □

[1] In fact, from the results in [4], these assumptions are minimal in systems whose links are either eventually timely or lossy asynchronous.

3. Implementation of Ω without membership knowledge

3.1. Definitions

We consider a system S composed by a finite set Π of n processes. The process identifiers are totally ordered, but need not be consecutive. Furthermore, the processes have no knowledge about Π whatsoever, and in particular they have no knowledge of n. Processes communicate only by sending and receiving messages. To send messages they have a broadcasting primitive. This primitive allows a process p to simultaneously send the same message m to the rest of processes in the system (e.g., like in Ethernet networks or IP-multicast).

A process can fail by permanently crashing. We say that a process is correct if it does not fail, and we say a process is crashed if it fails. We denote by *correct* the set of correct processes. We consider that in system S there may be any number of crashed processes, and that this number is not known. Processes execute by taking steps. We assume the existence of a lower bound σ on the number of steps per unit of time taken by any non-faulty process. For simplicity, we assume that each line of our algorithm represents one step. Processes have timers that accurately measure intervals of time.

We assume that every pair of processes is connected by a pair of directed links (with opposite directions). We consider that S has only two types of links: *eventually timely* and *lossy asynchronous*. In eventually timely links, there is an unknown bound Δ on message delays and an unknown (system-wide) global stabilization time T, such that if a message is sent through any of these links at a time $t \geqslant T$, then this message is received by time $t + \Delta$. In lossy asynchronous links, messages can be lost or arbitrarily delayed. Every message sent through a lossy asynchronous link may be lost, but every message that is not lost is eventually delivered. We consider that no link in S modifies its messages nor generates spontaneous messages. However, it may duplicate messages (a finite number of times) or deliver them out of order. For simplicity, we assume that messages are unique, in the sense that we can determine whether a message received is a duplicate of a previously received message. One way to achieve this is, for example, to consider that each message contains the sender process identifier and a sequence number.

The following is the property required by our algorithm to implement Ω[2]. Let us denote by $G(S)$ the directed graph, obtained from S, with the vertex set

[2] Nicely, this property is minimal in this class of systems [4].

```
init:
  (1) $mship_p \leftarrow \{p\}$
  (2) $punish_p \leftarrow \{(0, p)\}$
  (3) $leader_p \leftarrow p$
  (4) start Tasks 1 and 2
Task 1:
  (5) broadcast $(p, punish_p)$ every $\eta$ time
Task 2:
  (6) upon expiration of $timer_p(q)$ do
  (7)     $Timeout_p[q] \leftarrow Timeout_p[q] + 1$
  (8)     remove $(\cdot, q)$ from $punish_p$
  (9)     broadcast $(p, punish_p)$
 (10)     $leader_p \leftarrow$ process in $\min\{punish_p\}$

(11) upon reception of message $(q, set_q)$: $q \neq p$ do
(12)   if (message $(q, set_q)$ has not been received before) then
(13)       broadcast $(q, set_q)$
(14)       if $(q \notin mship_p)$ then
(15)           $mship_p \leftarrow mship_p \cup \{q\}$
(16)           create $timer_p(q)$ and $Timeout_p[q]$
(17)           $Timeout_p[q] \leftarrow \eta$
(18)       reset $timer_p(q)$ to $Timeout_p[q]$
(19)       if $(\exists (v, q) \in punish_p)$ then
(20)           replace in $punish_p$ $(v, q)$ by $(\max\{v, v'\}, q)$: $(v', q) \in set_q$
(21)       else
(22)           include in $punish_p$ $(v', q)$: $(v', q) \in set_q$
(23)       if $((\cdot, p) \notin set_q)$ then
(24)           replace in $punish_p$ $(v, p)$ by $(v + 1, p)$
(25)       $leader_p \leftarrow$ process in $\min\{punish_p\}$
```

Fig. 1. Implementation of Ω in system S. The code is for process p.

correct and the set of eventually timely links that connect processes in *correct* as edge set.

Property 1. *There is some process $p \in correct$ such that every process $q \in correct$ can be reached from p in $G(S)$.*

In an Ω failure detector, each process chooses some process in the system as *leader*. The detector must guarantee that all correct processes eventually agree on a single correct process as leader. More formally, Ω failure detectors must satisfy the following property.

Property 2. *There is a time after which every process $p \in correct$ permanently has the same process $l \in correct$ as leader.*

3.2. The Ω algorithm and its proof

Fig. 1 presents the algorithm that implements Ω in system S. Processes send messages periodically to show they are alive. These messages are re-broadcast to attempt reaching all processes. Each process p maintains a set $mship_p$ which contains all processes it currently knows (initially only itself). It also maintains a set $punish_p$ of pairs (v, q), where q is a process that p "believes" is alive, and $v \geqslant 0$ is roughly the number of times processes have "suspected" q. Every message sent by p contains this set $punish_p$. A process p holds in a variable $leader_p$ its current leader, which is the process q whose pair (v, q) in $punish_p$ has the smallest value v, using the process id to break ties.

Since the algorithm proposed by Aguilera et al. in [1] is the closest to the algorithm of Fig. 1, we briefly compare them. In [1], each process p periodically and permanently broadcasts *heartbeats* to the rest of processes of the system, and also sends *accusations* to each process q from which it did not receive heartbeats recently, *even if p never received any message from q*. Processes choose as leader the *least accused* alive process, choosing the process with smallest identifier to break ties. This algorithm does not work if p does not know q even in a system with only these two processes. To see this, assume $q < p$ and consider an execution in which all p's heartbeats are received on time by q and no heartbeat from q is received by p. Clearly, p chooses itself as leader. Since no accusation is sent (note that p does not know q and hence it cannot send accusations to q), q uses process identifiers to break ties and chooses itself as leader.

As we have previously mentioned, in Fig. 1 each process p periodically and permanently broadcasts *heartbeats* (line 5), which include the processes it *considers* alive (those from which it received a heartbeat recently). Then, *q punishes itself* every time it receives a heartbeat whose list of alive processes *does not* contain q (lines 23–24). A process chooses as leader the least punished process among those it considers alive (lines 25 and 10). Note that this algorithm works in the above example in which the algorithm in [1] failed because every heartbeat from p received by q will make q to punish itself. Hence, now q will correctly choose p as leader.

The proof of correctness follows.

Lemma 1. *For every process $q \notin correct$, there is a time after which no process $p \in correct$ has a pair $(\cdot, q)$ in $punish_p$.*

Proof (*Sketch*). Consider a time at which q has already failed. Note first that eventually all messages sent by q disappear from the system, because there can be only a finite number of duplicates of a message, messages are not held forever in any link, messages are processed at a minimum rate, and the same message is broadcast at most once by each process. Then, either p never received a message sent by q, and hence a pair $(\cdot, q)$ was never included in $punish_p$, or p eventually stops receiving messages from q. In the second case, the timer $timer_p(q)$ will eventually expire, and the pair $(\cdot, q)$ is permanently removed from $punish_p$. □

We will concentrate now on correct processes only. Observe in the algorithm that every process $p \in correct$ always holds a pair $(\cdot, p)$ in its set $punish_p$. We will denote by V_p the largest value v in a pair (v, p) ever held in $punish_p$. We define the set B as the subset of *correct* that contains every process p whose V_p is bounded. We define the set R as the subset of *correct* that contains every process p from which all correct processes can be reached in $G(S)$. Note that $R \neq \emptyset$ by Property 1. For the rest of the section we will assume that any time instant t is larger than the stabilization time T. Note that, from Property 1, a message sent at time $t > T$ and (re-)broadcast by a process in R at time t' will reach all correct processes by $t' + n(\Delta + 2/\sigma)$ time.

Lemma 2. $\emptyset \neq R \subseteq B$.

Proof (*Sketch*). Let $p \in R$. After time T a correct process q receives new messages from p with intervals of at most $\eta + n(\Delta + 2/\sigma)$ time. Then, eventually $timer_q(p)$ never expires and $(\cdot, p) \in punish_q$ permanently. All messages sent by q will contain a pair $(\cdot, p)$. This is true for all correct processes. Hence, there is a time after which p will stop increasing its local value v in the pair $(v, p) \in punish_p$ (line 24). □

Lemma 3. *For every process $p \in B$, there is a time after which every process $q \in correct$ permanently has (V_p, p) in $punish_q$.*

Proof (*Sketch*). Let us consider that $p \notin R$. Once p has included (V_p, p) in $punish_p$, every process $s \in R$ must receive (and rebroadcast) *n*ew messages from p within bounded intervals. Otherwise, $timer_s(p)$ would expire, s would remove any pair $(\cdot, p)$ from $punish_s$, and s would send a message with this set $punish_s$. This message would reach p, which would have to increase $(V_p + 1, p)$ in $punish_p$ (line 24). Then, independently of whether $p \in R$, messages from p with (V_p, p) are (re-)broadcast within bounded intervals by some process in R. Hence, eventually every process $q \in correct$ receives the message, and includes/replaces the pair (V_p, p) in $punish_q$. □

We define $V_{\min} = \min_{p \in correct}\{V_p\}$.

Lemma 4. *For every correct process $p \notin B$, there is a time after which no correct process q has a pair (v, p) with $v \leqslant V_{\min}$ in its set $punish_q$.*

Proof (*Sketch*). Consider the time t at which process p inserts the pair $(V_{\min} + 1, p)$ in its set $punish_p$. All messages sent by p before t eventually disappear from the system, say at time t'. If process $q \in correct$ receives after t' some message sent by p after t, it inserts a pair (v, p) in $punish_q$ with $v > V_{\min}$. If q does not receive after t' any such message, eventually $punish_q$ will not contain a pair $(\cdot, p)$, either because it was not there at time t' or because timer $timer_q(p)$ expired and the pair was removed. □

Theorem 2. *There is a process $l \in correct$ and a time after which every process $p \in correct$ permanently has $leader_p = l$.*

Proof. The variable $leader_p$ of process p always holds the process q whose pair (v, q) is minimal in $punish_p$. From Lemma 1, there is a time after which only correct processes have pairs in $punish_p$. Then, if we look at the set B, Lemma 2 shows that B is not empty, and Lemma 3 shows that for each $s \in B$, eventually $punish_p$ will contain (V_s, s) forever. Finally, Lemma 4 shows that for each correct process $s \notin B$, there is a time after which no correct process q has a pair (v, s) with $v \leqslant V_{\min}$ in its set $punish_q$. Hence, there is a time after which $leader_p$ permanently holds the process $l \in correct$ such that $l = \min\{q \in B\colon\ V_q = V_{\min}\}$. □

Corollary 1. *The algorithm of Fig.* 1 *implements an Ω failure detector without knowledge of the membership in any system S that satisfies Property* 1.

References

[1] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg, On implementing omega with weak reliability and synchrony assumptions, in: PODC 2003, July 2003, pp. 306–314.

[2] T.D. Chandra, V. Hadzilacos, S. Toueg, The weakest failure detector for solving consensus, Journal of the ACM 43 (4) (1996) 685–722.

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

[4] A. Fernández, E. Jiménez, S. Arévalo, Brief announcement: Minimal system conditions to implement unreliable failure detectors, in: PODC 2005, Las Vegas, Nevada, July 2005. Full version available at http://gsyc.escet.urjc.es/publicaciones/tr/RoSaC-2005-3.pdf.