Communication-efficient and crash-quiescent Omega with unknown membership ☆

Sergio Arévalo [a], Ernesto Jiménez [a,*], Mikel Larrea [b], Luis Mengual [c]

[a] EUI, Universidad Politécnica de Madrid, 28031 Madrid, Spain
[b] Universidad del País Vasco, 20018 San Sebastián, Spain
[c] FI, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Spain

ARTICLE INFO

Keywords:
Distributed computing
Fault tolerance
Unreliable failure detectors

ABSTRACT

The failure detector class Omega (Ω) provides an eventual leader election functionality, i.e., eventually all correct processes permanently trust the same correct process. An algorithm is communication-efficient if the number of links that carry messages forever is bounded by n, being n the number of processes in the system. It has been defined that an algorithm is crash-quiescent if it eventually stops sending messages to crashed processes. In this regard, it has been recently shown the impossibility of implementing Ω crash quiescently without a majority of correct processes. We say that the membership is unknown if each process p_i only knows its own identity and the number of processes in the system (that is, i and n), but p_i does not know the identity of the rest of processes of the system. There is a type of link (denoted by ADD link) in which a bounded (but unknown) number of consecutive messages can be delayed or lost.

In this work we present the first implementation (to our knowledge) of Ω in partially synchronous systems with ADD links and with unknown membership. Furthermore, it is the first implementation of Ω that combines two very interesting properties: communication-efficiency and crash-quiescence when the majority of processes are correct. Finally, we also obtain with the same algorithm a failure detector ($\diamond \overline{P}$) such that every correct process eventually and permanently outputs the set of all correct processes.

1. Introduction

Unreliable failure detectors were proposed by Chandra and Toueg [4] as an elegant way of circumventing the well known FLP impossibility result [6] on solving deterministically the consensus problem [7] in crash-prone asynchronous environments. Informally, an unreliable failure detector provides hints about which processes are correct and which ones are incorrect, i.e., have crashed so far. The failure detector can make mistakes, e.g., by erroneously suspecting a correct but slow process, or by not suspecting (yet) a crashed process. Nevertheless, to be useful it is required that failure detectors fulfill some completeness (i.e., suspect permanently crashed processes) and accuracy (i.e., stop suspecting correct processes) properties. For example, a failure detector such that every correct process eventually and permanently suspects all incorrect processes, and eventually does not suspect any correct process is named Eventually Perfect ($\diamond P$).

Another important failure detector is Omega (Ω) [3], defined in terms of a trusted process, such that eventually all correct processes permanently trust the same correct process. It is said that Ω provides an eventual leader election functionality. This type of failure detector is important

☆ Research partially supported by the Spanish Research Council (MCeI), under grant TIN2010-19077, the Basque Government, under grants S-PE09UN50 and IT395-10, and the Comunidad de Madrid, under grant S2009/TIC-1692.

* Corresponding author.
E-mail addresses: sergio.arevalo@eui.upm.es (S. Arévalo), ernes@eui.upm.es (E. Jiménez), mikel.larrea@ehu.es (M. Larrea), lmengual@fi.upm.es (L. Mengual).

because it has been shown that Ω is the weakest failure detector for solving consensus [3].

We say that an algorithm implementing Ω is communication-efficient if the number of links that carry messages forever is bounded by n, being n the number of processes in the system [2].

A new property called crash-quiescence has been recently proposed [9]. A distributed algorithm is said crash-quiescent if it eventually stops sending messages to crashed processes. Sastry et al. have shown in [9] the impossibility of implementing Ω (and thus $\diamond P$) crash quiescently without a majority of correct processes in a partially synchronous system with links where a bounded, but unknown, number of consecutive messages can be arbitrarily late or lost (called ADD links). They also propose the first implementation of a (non-communication-efficient) $\diamond P$ algorithm that is crash-quiescent in any run with a majority of correct processes. ADD links [8] are interesting and realistic enough because they allow that an infinite number of messages could be lost or arbitrarily delayed, but guarantee that some subset of the messages sent on them will be received in a timely manner and such messages are not too sparsely distributed in time. In some sense, an ADD link successfully combines the losses property of a fair lossy link [1] (in which if an infinite number of messages are sent, then an infinite number of messages are received), and the timeliness property of an eventually timely link [1] (in which there is a time after which all the messages that are sent are received timely).

A system with unknown membership allows each process to execute the algorithm of Ω without having to know initially the identifiers of all processes (only knowing its own identifier and the total number of processes in the system is enough). Hence, you can execute the same code of Ω in runs with different processes without having to statically change the set of addresses of processes that participate in each run. This is possible because processes dynamically learn about the existence of other processes from the reception of messages. Note that this reduction of initial knowledge also supposes that if some process crashes before sending a message, the rest of processes will not be able to find out its existence as member of the system. For example, let us suppose you have a p2p environment in Internet where each server is identified by an IP-address. We know that the number of servers will be, for example, 10. In systems where the membership is known, each process has to know each of the 10 server's IP-address previously to start the execution of Ω to elect some of them as leader. In systems with unknown membership, the leader can be chosen without this knowledge by learning IP-addresses from the messages received from other processes.

Jiménez et al. show in [5] that it is necessary for each process to know the identity of the rest of processes of the system (that is, the membership) to implement a failure detector of anyone of the original eight classes proposed by Chandra and Toueg in [4] (and, hence, $\diamond P$). Interestingly, they also show in the same paper that Ω can be implemented without this knowledge. What it is possible to implement is the complement of $\diamond P$, that is, a failure detector such that every correct process eventually and permanently only trusts (instead of suspects) all correct (instead of incorrect) processes. We denote this failure detector as $\diamond \overline{P}$. Note that when the membership is known, $\diamond \overline{P}$ can be easily transformed into $\diamond P$ (we can obtain the set of suspected processes simply by removing from the set of all processes the set of trusted processes output by $\diamond \overline{P}$).

1.1. Our results

In this work we present the first implementation (to our knowledge) of Ω in partially synchronous systems with ADD links and where the membership is unknown. Furthermore, it is the first implementation that combines two very interesting properties: communication-efficiency and crash-quiescence when the majority of processes are correct. Finally, we also want to emphasize that the implementation presented in this paper also satisfies the properties of $\diamond \overline{P}$.

2. The model

We consider a system S composed of a finite set Π of n processes. The process identifiers are totally ordered, but they do not need to be consecutive. We denote them by $p_i, p_l, p_r, \ldots$. We assume that processes only know about Π their own identifier and what is the number n of processes in S (actually, they only need to known the majority, that is, $n/2$). For example, process p_i initially only knows i and n.

Processes communicate only by sending and receiving messages. We assume that processes have a broadcasting primitive to send messages (called *broadcast*(m)). This primitive allows a process to send simultaneously the same message m to the rest of processes in the system (that is, the same mechanism used in Ethernet or in IP-multicast). Furthermore, processes also have the possibility to use the primitive send (called *send*(m) to q) to send the message m to process q only.

A process can fail by crashing permanently. We say that a process is correct if it does not fail, and we say a process is crashed (or it is a faulty process) if it fails. We use C to denote the set of correct processes, and F to denote the set of faulty processes. We consider that in a system S there may be any number of crashed processes, and that this number is not known a priori. We assume that processes are synchronous in the sense that there are lower and upper bounds on the number of processing steps they execute per unit of time. These bounds are not known by processes. For simplicity, the local processing time is negligible with respect to message communication delays. Processes have timers that accurately measure intervals of time.

We assume that every pair of processes is connected by a pair of directed links. We consider that all links belong to type ADD (Average Delayed/Dropped) [8]. An ADD link allows an infinite number of messages to be lost or arbitrarily delayed, but guarantees that some subset of the messages sent on it will be received in a timely manner and such messages are not too sparsely distributed in time. More precisely, there are two unknown constants Δ and B

```
initialization:
(01) correct_i ← {i};
(02) membership_i ← {i};
(03) leader_i ← i;
(04) start tasks T1 and T2;

task T1:
(05) repeat forever each η time
(06)     if (|correct_i| > n/2) then
(07)         successor_i ← next_to_i_in_correct_i();
(08)         send(correct_i) to successor_i;
(09)     else
(10)         broadcast({i});
(11)     end_if
(12) end_repeat

task T2:
(13) upon reception of message (set_j) : j ≠ i do
(14)     for_each (k ∈ set_j : k ≠ i) do
(15)         if (k ∉ membership_i ) then
(16)             membership_i ← membership_i ∪ {k};
(17)             create timer_i(k) and timeout_i[k];
(18)             timeout_i[k] ← 1;
(19)         end_if
(20)         correct_i ← correct_i ∪ {k};
(21)         set timer_i(k) to timeout_i[k];
(22)     end_for
(23)     leader_i ← min(correct_i);

(24) upon expiration of timer_i(k):
(25)     timeout_i[k] ← timeout_i[k] + 1;
(26)     correct_i ← correct_i \ {k};
(27)     leader_i ← min(correct_i);
```

Fig. 1. Implementation of Ω (and $\Diamond\overline{P}$) in a system where each process initially only knows its identity and n (code for process p_i).

such that for all intervals of time in which a process p_i sends at least B messages to another process p_j, at least one of these messages is received by p_j in at most Δ time.

3. The Omega and $\Diamond\overline{P}$ failure detectors

Chandra et al. proposed the Omega (Ω) failure detector [3]. This failure detector guarantees that eventually all correct processes always have the same correct process as leader. More formally, the Omega failure detector satisfies the following property: there is a time after which each process $p_i \in C$ always has $leader_i = p_l$, where $p_l \in C$.

Chandra and Toueg defined in [4] the Eventually Perfect ($\Diamond P$) failure detector. This failure detector guarantees that there is a time after which every correct process permanently only suspects all faulty processes. In [5] it is proven that if processes do not know a priori all members of the system (i.e., the membership is unknown), it is impossible to eventually suspect all faulty processes. For example, if we do not know the membership and process p_i crashes at the very beginning of a run (before sending or receiving any message), it is not possible for another process p_j to learn about p_i's presence in the system, and therefore, it is impossible for p_j to suspect p_i.

Nevertheless, we can have in a system S of our model a failure detector (we denote it as $\Diamond\overline{P}$) such that there is a time after which every correct process permanently only trusts all correct processes. More formally, there is a time after which each process $p_i \in C$ always has $|correct_i| = |C|$, and $p_j \in correct_i$, for all $p_j \in C$. Note that in a system with known membership this failure detector can be easily transformed into $\Diamond P$ (changing the output of each process p_i by $\Pi \setminus correct_i$). Interestingly, our protocol also implements $\Diamond\overline{P}$.

4. The implementation

Fig. 1 implements Ω (and $\Diamond\overline{P}$) in a system S where the membership is unknown. If the majority of processes are correct, we will show that this protocol is communication-efficient and crash-quiescent.

In Fig. 1, each process p_i initially broadcasts heartbeats repeatedly to indicate that it is alive (line 10). To know which processes are in the system, process p_i has a set $membership_i$ (initially only contains itself). The set $correct_i$ maintains the processes it believes are alive (always containing at least itself). If a majority of processes are alive, then eventually $|correct_i| > n/2$ and process p_i will send heartbeats periodically to $successor_i$, instead of broadcast them (line 08). This variable $successor_i$ contains the process returned by function $next\_to\_i\_in\_correct_i()$ (line 07). This function obtains the identifier of the process closest to the identifier of p_i in the sequence formed by all elements of $correct_i$, in increasing order and cyclic (that is, like operation *mod* but with the elements of $correct_i$ instead of a subset of natural numbers). For example, if process p_3 has $correct_3 = \{1, 3, 5, 8\}$, then $next\_to\_i\_in\_correct_3()$ will return 5. In another example, if process p_3 has $correct_3 = \{1, 2, 3\}$, then $next\_to\_i\_in\_correct_3()$ will return 1. As we will show, if $|correct_i| > n/2$, each correct process p_i will eventually send messages only to one correct process, establishing a cycle (we will define it below as ring) formed by all correct processes. The variable $leader_i$ has the identifier of the process that p_i considers its leader. Its value is the smallest value in $correct_i$.

5. Correctness proof

Let D be a subset of correct processes ($D \subseteq C$). We say that there is a relation $p_i \to p_j$, $p_i, p_j \in D$, if there is a time after which process p_i is permanently sending heartbeats to p_j. We say that p_i is the predecessor of p_j, and p_j is the successor of p_i in D. We say there is a ring among all processes in D, denoted by $ring(D)$, if each process $p_i \in D$ has a unique predecessor and a unique successor with respect to all processes in D. For example, if D contains the subset of correct processes $\{p_l, p_k, p_r, p_s\}$, and $p_r \to p_k \to p_l \to p_s \to p_r$, then we say that there is a $ring(D)$.

The following lemma states that eventually crashed processes cannot be in the set $correct_i$ of any correct process p_i.

Lemma 1. *For every process $p_i \in C$, there is a time after which if $j \in correct_i$, then $p_j \in C$.*

Proof. Let us consider a process $p_j \in F$. We prove that eventually and permanently $j \notin correct_i$, $\forall p_i \in C$. Let seq_j be a permutation of the processes in C, where the first element of the sequence seq_j is the process whose identifier $next\_to\_i\_in\_correct_j()$ would return if $correct_j = C$. Let $seq_j(x)$ be, for $x \geqslant 1$, the xth element of the sequence seq_j. The element $seq_j(x)$, $x \geqslant 2$, is the process whose identifier $next\_to\_i\_in\_correct_{seq_j(x-1)}()$ would return if $correct_{seq_j(x-1)} = C$.

We are going to prove that, for all $x \geqslant 1$, eventually and permanently $j \notin correct_{seq_j(x)}$, by induction on x. Base case is $x = 1$. Let us consider a time τ at which all faulty processes are crashed, and all messages sent by these processes have disappeared from the system (i.e., they have been delivered or lost). After τ, process $seq_j(1)$ will never receive any heartbeat with a set containing j. This is so, because a process sending a heartbeat by execution of line 10, it only includes its own identifier (which cannot be j since process p_j is already crashed). On the other hand, if the heartbeat is sent by execution of line 08, the values in this set $correct_i$, $\forall p_i \in C$, sent to process $seq_j(1)$ do not include j (otherwise, the heartbeat would have been sent to p_j instead). For example, let us consider that $j = 2$, $\Pi = \{p_0, p_1, p_2, p_3\}$, $p_j \in F$ and $t > \tau$. We know that $seq_2 = 3 \mapsto 0 \mapsto 1 \mapsto 2$ (where $a \mapsto b$ denotes that a precedes to b in the sequence), and, hence, $seq_2(1) = 3$. Let us also consider that at time $t > \tau$ we have $correct_0 = \{2, 0\}$ and $correct_1 = \{3, 0, 1\}$. Then, process p_0 sends its heartbeats to process p_2 (which is crashed), and process p_1 sends its heartbeats to process p_3 (and as base case states, $correct_1$ does not contain 2). If $j \in correct_{seq_j(1)}$ at time τ, eventually the timer $timer_{seq_j(1)}(j)$ will expire and j will be permanently removed from $correct_{seq_j(1)}$. If $j \notin correct_{seq_j(1)}$ at time τ, j is never included after τ. Hence, there is a time $\tau_1 \geqslant \tau$ after which $j \notin correct_{seq_j(1)}$ permanently.

Let us now consider $x \geqslant 2$. By induction hypothesis, after τ_{x-1}, we have that $j \notin correct_y$, for all $y \in \{1, \dots, x-1\}$. Note that now process $seq_j(x)$ will receive heartbeats from $seq_j(x-1)$ with a set $correct_{seq_j(x-1)}$ not containing j. Then, following the same reasoning of the base case, process $seq_j(x)$ will never receive any heartbeat with a set containing j. Hence, there is a time $\tau_x \geqslant \tau_{x-1}$ after which $j \notin correct_{seq_j(x)}$ permanently.

Therefore, for all $x \geqslant 1$, eventually and permanently $j \notin correct_{seq_j(x)}$. As the sequence seq_j includes all correct processes, and the proof holds for all $p_j \in F$, the lemma follows. □

The following lemma assures that if eventually a correct process p_i believes that a minority of processes are alive, then eventually all correct processes in the system will believe that p_i is alive.

Lemma 2. *If there is a process $p_i \in C$ such that eventually $|correct_i| \leqslant n/2$, then there is a time after which $i \in correct_j$, for every $p_j \in C$.*

Proof. If there is a time after which a process $p_i \in C$ always has $|correct_i| \leqslant n/2$, then it will be permanently broadcasting heartbeats with i each η time (line 10). As all links are ADD, then at least one from each B messages is receive in each process p_j in at most $\eta B + \Delta$ time. Hence, eventually $timer_j(i)$ will never expire, and process p_j will always have $i \in correct_j$. □

The following lemma shows that if a majority of processes are crashed, eventually all correct processes have $|correct| \leqslant n/2$.

Lemma 3. *If $|C| \leqslant n/2$, then there is a time after which $|correct_i| \leqslant n/2$, for every process $p_i \in C$.*

Proof. From Lemma 1, there is a time after which every correct process $p_i \in C$ will have $|correct_i| \leqslant |C|$. Hence, if $|C| \leqslant n/2$, then $|correct_i| \leqslant n/2$. □

In the following lemma we show that if there is a minority of correct processes, eventually all correct processes have the same set *correct*.

Lemma 4. *If $|C| \leqslant n/2$, there is a time after which $correct_i = correct_j$, for all $p_i, p_j \in C$.*

Proof. From Lemma 3, if $|C| \leqslant n/2$, there is a time after which every correct process p_i will broadcast heartbeats permanently with its identifier i each η time (line 10). Then, eventually no heartbeats with j, where $p_j \in F$, will be received by correct processes. Then, $timer_i(j)$ eventually will expire (if it was previously set by p_i) and j will not be in $correct_i$ anymore. Hence, there is a time after which $correct_i$ does not contain identifiers of crashed processes. From Lemma 2 and Lemma 3, if $|C| \leqslant n/2$, there is a time after which every $p_i \in C$ will have $j \in correct_i$, for every process $p_j \in C$. Therefore, if $|C| \leqslant n/2$, there is a time after which $correct_i = correct_j$, for all $p_i, p_j \in C$. □

The following lemma shows that if a minority of processes are crashed, eventually all correct processes have $|correct| > n/2$.

Lemma 5. *If $|C| > n/2$, then there is a time after which $|correct_i| > n/2$, for every process $p_i \in C$.*

Proof. By contradiction, let us suppose that $|C| > n/2$, and there is a subset $E \neq \emptyset$ such that eventually each process $p_k \in E$ has $|correct_k| \leqslant n/2$. Then, there is also a complementary subset G such that eventually each process $p_l \in G$ has $|correct_l| > n/2$ (being $E \cap G = \emptyset$, and $E \cup G = C$). We have two cases to study:

Case 1. $G = \emptyset$. That is, every correct process $p_k \in E$. From Lemma 2, there is a time after which $k \in correct_j$, for every $p_j \in C$. So, as by hypothesis of contradiction $|C| > n/2$, we eventually have $|correct_j| > n/2$, for every $p_j \in C$, and hence, we reach a contradiction.

Case 2. $G \neq \emptyset$. By hypothesis of contradiction we also have that $E \neq \emptyset$. Note that eventually each process $p_l \in G$, as it has $|correct_l| > n/2$, sends heartbeats permanently with

$correct_l$ each η time to its successor (line 08). As links are ADD, then at least one from each B messages will be received by every correct process in at most $\eta B + \Delta$ time. Hence, there is a time after which the successor of p_l will receive on time heartbeats of p_l. From Lemma 1, eventually $correct_i$ can only contain correct processes, for every process $p_i \in C$. From Lemma 2, there is a time after which each process $p_j \in C$ has $k \in correct_j$, for every process $p_k \in E$. Note that for each process its set *correct* is updated with the values received from heartbeats (line 20). Hence, eventually the successor of some process $p_g \in G$ will be one of the processes p_s of E. So, eventually $correct_g \subseteq correct_s$, and as $|correct_g| > n/2$, we also have that $|correct_s| > n/2$, and we reach a contradiction. □

The following lemma proves that if the majority of processes are correct, a unique ring is formed among all of them.

Lemma 6. *If $|C| > n/2$, then there is a unique $ring(C)$.*

Proof. If there is a time after which some process $p_i \in C$ has $|correct_i| > n/2$, then eventually process p_i sends heartbeats with $correct_i$ each η time to its successor (line 08). As links are ADD, then at least one from each B messages will be received by every correct process in at most $\eta B + \Delta$ time. So, there is a time after which the successor of $p_i \in C$ will receive on time heartbeats of p_i.

From Lemma 1, eventually $correct_i$ can only contain correct processes, for every process $p_i \in C$. From Lemma 5, if $|C| > n/2$ then eventually $|correct_i| > n/2$, for every process $p_i \in C$. So, process p_i will form a ring with other correct processes, and the intersection between each pair of rings is not empty. More formally, let us suppose that there are m subsets of correct processes, denoted by $C_1, C_2, \ldots, C_m$, such that: (a) $ring(C_1), ring(C_2), \ldots, ring(C_m)$, (b) $C_1 \cup C_2 \cup \cdots \cup C_m = C$, and (c) $ring(C_1) \cap ring(C_2) \cap \cdots \cap ring(C_m) \neq \emptyset$.

Let us use induction in the number n of rings from each process $p_a \in ring(C_k)$ and each process $p_b \in ring(C_{k+n})$ to show that, if $|C| > n/2$, then there is a unique $ring(C)$.

Let $n = 1$, that is, p_a, $p_b \in ring(C_k)$. The claim is trivially true.

Let us now suppose that the claims holds for any distance $0 \leqslant n \leqslant r$. Let us now prove that the claim is also held for $n + 1$. We have process $p_a \in ring(C_{k+n})$, process $p_b \in ring(C_{k+n+1})$, and, from the induction hypothesis we have that $|C| > n/2$, and hence, from Lemmas 1 and 5, the intersection between these two rings is not empty. That is, there is some correct process p_x such that $p_x \in ring(C_{k+n})$, and $p_x \in ring(C_{k+n+1})$. Hence, process p_x receives the set *correct* of some process $p_s \in ring(C_{k+n})$ and some process $p_t \in ring(C_{k+n+1})$ (i.e., process p_x is the successor of p_s and p_t). As correct process p_x updates its set $correct_x$ with the values received from heartbeats (line 20), it eventually sends its set $correct_x$ with the union of *correct* from $ring(C_{k+n})$ and $ring(C_{k+n+1})$. Hence, a unique ring will be formed among all correct processes of both rings. Therefore, if $|C| > n/2$, then there is a unique $ring(C)$. □

In the following lemma we show that if there is a majority of correct processes, eventually all correct processes have the same set *correct*.

Lemma 7. *If $|C| > n/2$, there is a time after which $correct_i = correct_j$, for all $p_i, p_j \in C$.*

Proof. From Lemma 6, if $|C| > n/2$, there is $ring(C)$. From Lemma 1, the set $correct_i$ of each process $p_i \in C$ eventually can only contain correct processes. Then, each process $p_i \in C$ sends $correct_i$ to its successor (line 08). This successor of p_i (for example p_k) includes the processes sent by process p_i in its set $correct_k$ (line 20). Hence, as there is a ring, eventually $correct_i = correct_j$, for all $p_i, p_j \in C$. Therefore, if $|C| > n/2$, there is a time after which $correct_i = correct_j$, for all $p_i, p_j \in C$. □

Theorem 1. *Let process $p_l \in C$. There is a time after which every process $p_i \in C$ permanently has $leader_i = l$.*

Proof. From Lemma 1, there is a time after which all processes in $correct_i$ have to be correct, for every $p_i \in C$. From Lemmas 4 and 7, there is a time after which $correct_i = correct_j$, for every $p_i, p_j \in C$. Note that $correct_i$ is never empty, for every process p_i. This is so because initially process p_i includes itself in this set (line 01), and this value i is never removed from $correct_i$ (because $timer_i(i)$ is never started, lines 14 and 21). Hence, there is a time after which process p_i permanently has $leader_i = l$, being $l = min(correct_i)$ (lines 23 and 27). □

Theorem 2. *If $|C| > n/2$, the algorithm of Fig. 1 is communication–efficient and crash-quiescent.*

Proof. From Lemmas 5 and 6, there is a $ring(C)$ if $|C| > n/2$. Then, this ring will be formed by sending heartbeats permanently among all correct processes in a cyclic way, and thus, the number of links that eventually carry messages forever is $|C|$. Hence, the algorithm of Fig. 1 is communication-efficient and crash-quiescent. □

In the following theorem we show that the algorithm of Fig. 1 also implements $\diamond\overline{P}$.

Theorem 3. *There is a time after which each process $p_i \in C$ always has $|correct_i| = |C|$, and $p_j \in correct_i$, for all $p_j \in C$.*

Proof. From Lemma 1, eventually crashed processes cannot be in the set $correct_i$, $\forall p_i \in C$.

If $|C| \leqslant n/2$, from Lemmas 2 and 3, eventually each process $p_i \in C$ has $j \in correct_i$, $\forall p_j \in C$. Hence, if $|C| \leqslant n/2$, there is a time after which each process $p_i \in C$ always has $|correct_i| = |C|$, and $p_j \in correct_i$, for all $p_j \in C$.

If $|C| > n/2$, from Lemma 7 we have that eventually $correct_i = correct_j$, for all $p_i, p_j \in C$, and from Lemma 6 there is a unique ring formed by all correct processes. Looking at the algorithm of Fig. 1, each process p_i of this unique ring sends heartbeats to its successor (i.e., process p_j) with $correct_i$ (line 08), and this successor p_j includes the values of $correct_i$ in its $correct_j$ (lines 14 and 20). We know that $correct_i$ always contains at least i, for all process p_i. This is so because initially process p_i includes itself in this set (line 01), and this value i is never removed from $correct_i$ (because $timer_i(i)$ is never started, lines 14 and 21). Hence, if $|C| > n/2$, there is a time after which each process $p_i \in C$ always has $|correct_i| = |C|$, and $p_j \in correct_i$, for all $p_j \in C$.

Therefore, there is a time after which each process $p_i \in C$ always has $|correct_i| = |C|$, and $p_j \in correct_i$, for all $p_j \in C$. $\square$

References

[1] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg. On implementing omega with weak reliability and synchrony assumptions, in: Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 2003, pp. 306–314.

[2] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg, Stable leader election, in: Proceedings of the 15th International Symposium on Distributed Computing (DISC'2001), Lisbon, Portugal, October 2001, in: LNCS, vol. 2180, Springer-Verlag, 2001, pp. 108–122.

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

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

[5] E. Jiménez, S. Arévalo, A. Fernández, Implementing unreliable failure detectors with unknown membership, Information Processing Letters 100 (2) (October 2006) 60–63.

[6] M. Fischer, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (April 1985) 374–382.

[7] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, Journal of the ACM 27 (2) (April 1980) 228–234.

[8] S. Sastry, S. Pike, Eventually perfect failure detection using ADD channels, in: Proceedings of the 5th International Symposium on Parallel and Distributed Processing and Applications, ISPA 2007, Niagara Falls, Canada, August 2007, in: LNCS, vol. 4742, Springer-Verlag, 2007, pp. 483–496.

[9] S. Sastry, S. Pike, J. Welch, Crash-quiescent failure detection, in: Proceedings of the 23rd International Symposium on Distributed Computing, DISC'2009, Elche, Spain, September 2009, in: LNCS, vol. 5805, Springer-Verlag, 2009, pp. 326–340.