An Evaluation of Communication-Optimal $\diamond\mathcal{P}$ Algorithms *

Mikel Larrea, Iratxe Soraluze, Roberto Cortiñas, Alberto Lafuente
The University of the Basque Country
20018 San Sebastián, Spain
{*mikel.larrea, iratxe.soraluze, roberto.cortinas, alberto.lafuente*}*@ehu.es*

Abstract

This paper presents an evaluation of several communication-optimal algorithms implementing the $\diamond\mathcal{P}$ class of failure detectors. The first algorithm is based on a Reliable Broadcast primitive, involving a quadratic number of messages to manage a suspicion. The second algorithm uses exclusively one-to-one communication, involving a linear number of messages to manage a suspicion, but with a higher latency to propagate the suspicion to the rest of processes. A third algorithm reduces this latency using an additional one-to-all communication mechanism. We evaluate the quality of service provided by these algorithms, in terms of the capability of the failure detector to provide right answers and the reaction time after a failure.

1. Introduction

Unreliable failure detectors, proposed by Chandra and Toueg [4], have been used to address the *consensus* problem [11] and several related problems in asynchronous crash-prone distributed systems. In this paper, we mainly focus on the *Eventually Perfect* failure detector class, denoted $\diamond\mathcal{P}$, which satisfies (1) strong completeness: eventually every process that crashes is permanently suspected by every correct process, and (2) eventual strong accuracy: there is a time after which correct processes are not suspected by any correct process.

Consensus can also be solved with a weaker failure detector class called *Eventually Strong*, denoted $\diamond\mathcal{S}$, which satisfies strong completeness and eventual *weak* accuracy: there is a time after which *some* correct process is not suspected by any correct process. Specifically, a particular failure detector called Ω, equivalent to $\diamond\mathcal{S}$, has been proved to be the weakest failure detector to solve consensus [3]. The Ω failure detector provides eventual agreement on a common *leader* among all non-faulty processes in a system. Specific algorithms for implementing Ω and/or $\diamond\mathcal{S}$ have been proposed in the literature [1, 2, 8]. However, note that since $\diamond\mathcal{P}$ is strictly stronger than $\diamond\mathcal{S}$, any implementation of $\diamond\mathcal{P}$ trivially implements $\diamond\mathcal{S}$. Observe also that $\diamond\mathcal{P}$ can also be easily transformed into Ω, e.g., by choosing as leader the non-suspected process with lowest identifier. Also, for certain problems [7] and consensus protocols [12] failure detector $\diamond\mathcal{P}$ is required. These facts lead us to look for $\diamond\mathcal{P}$ based solutions.

In [9] it has been proposed a family of heartbeat-based algorithms which implement $\diamond\mathcal{P}$ using a logical ring arrangement of processes. In these algorithms, every process p tries to determine its correct successor in the ring, i.e., the process to which p should send heartbeats forever, and also its correct predecessor in the ring, i.e., the process from which p should receive heartbeats forever. The algorithms are *communication-efficient* [1], i.e., eventually only n unidirectional links carry messages forever. Recently, it has been proposed in [10] a *communication-optimal* implementation of $\diamond\mathcal{P}$, in which eventually only $\mathcal{C}$ unidirectional links carry messages forever, being $\mathcal{C}$ the number of correct processes in the system.

The first algorithm presented in this paper is the communication-optimal implementation of $\diamond\mathcal{P}$ proposed in [10], where every process communicates suspicions (and refutations) to the rest of processes using a Reliable Broadcast primitive [4], i.e., a reliable form of *one-to-all* communication. A characteristic of the algorithm is that, due to the use of Reliable Broadcast, the number of messages exchanged when a suspicion occurs is quadratic. This can be a serious drawback in some scenarios, e.g., very large networks, in which traffic-load is a critical issue.

As a second algorithm, we propose a communication-optimal implementation of $\diamond\mathcal{P}$ which uses exclusively

*Research partially supported by the Spanish Research Council, grants TIN2007-67353-C02-02 and TIN2006-15617-C03-01, the Basque Government, grant S-PE06IK01, and the Comunidad de Madrid, grant S-0505/TIC/0285.

one-to-one communication, even for communicating suspicions and refutations. In this algorithm, information about suspicions will be included into heartbeat messages and propagated around the ring. Since Reliable Broadcast is not used, the overhead to manage suspicions is reduced considerably. A drawback of this algorithm is its linear crash detection time. We reduce it by defining a new algorithm that introduces sporadic *one-to-all* communication.

We evaluate the performance of these three communication-optimal implementations of $\diamond\mathcal{P}$ in terms of QoS measures, comparing them to Chandra-Toueg's *all-to-all* based $\diamond\mathcal{P}$ algorithm.

The rest of the paper is organized as follows. In Section 2, we describe the system model considered in this work. In Section 3, we present the three communication-optimal algorithms implementing $\diamond\mathcal{P}$. In Section 4, we analyze the complexity and evaluate the performance of the algorithms. Finally, Section 5 concludes the paper.

2. System model

We consider a distributed system composed of a finite set Π of n processes, $\Pi = \{p_1, p_2, \dots, p_n\}$, that communicate only by sending and receiving messages. Every pair of processes (p_i, p_j) is connected by two unidirectional and reliable communication links $p_i \rightarrow p_j$ and $p_j \rightarrow p_i$.

Processes can only fail by crashing, that is, by prematurely halting. Moreover, crashes are permanent, i.e., crashed processes do not recover. In every run of the system we identify two complementary subsets of Π: the subset of processes that do not fail, denoted $correct$, and the subset of processes that do fail, denoted $crashed$. We use $\mathcal{C}$ to denote the number of correct processes in the system in the run of interest, which we assume is at least one, i.e., $\mathcal{C} = |correct| \geq 1$.

We consider that processes are arranged in a logical ring. Without loss of generality, process p_i is preceded by process p_{i-1}, and followed by process p_{i+1}. As usual, p_1 follows p_n in the ring. In general, we will use the functions $pred(p)$ and $succ(p)$ respectively to denote the predecessor and the successor of a process p in the ring.

Concerning timing assumptions, we consider a partially synchronous model [4, 6] which stipulates that, in every run of the system, there are bounds on relative process speeds and on message transmission times, but these bounds are not known and they hold only after some unknown but finite time (called *GST* for *Global Stabilization Time*). Actually, the bounds must exist and hold only for the $\mathcal{C}$ links that eventually form the ring of correct processes, i.e., the links from every correct process to its correct successor in the ring.

Finally, in the algorithms presented in this paper we assume that a local clock that can measure real-time intervals is available to each process. Clocks are not synchronized.

3. Communication-optimal implementations of $\diamond\mathcal{P}$

3.1. Reliable Broadcast based optimal $\diamond\mathcal{P}$

We describe here a first communication-optimal algorithm that implements $\diamond\mathcal{P}$ using Reliable Broadcast [10].

Algorithm 1: Communication-optimal $\diamond\mathcal{P}$ using Reliable Broadcast.

{*Every process p executes the following*}

Procedure $update\_pred\_and\_succ()$
    if $\forall r : Balance_p(r) > 0$ **then**
        $pred_p \leftarrow p$
        $succ_p \leftarrow p$
    else
        $pred_p \leftarrow$ p's nearest predecessor r in the ring such that $Balance_p(r) \leq 0$
        $succ_p \leftarrow$ p's nearest successor r in the ring such that $Balance_p(r) \leq 0$

$pred_p \leftarrow pred(p)$
$succ_p \leftarrow succ(p)$
forall $q \in \Pi$ **do**
    $\Delta_p(q) \leftarrow$ default time-out interval
    $Balance_p(q) \leftarrow 0$

cobegin
    || <u>Task 1</u>: **repeat periodically**
        if $succ_p \neq p$ **then**
            send $(ALIVE, p)$ to $succ_p$

    || <u>Task 2</u>: **repeat periodically**
        if *(*$pred_p \neq p$*)* **and** *p didn't receive* $(ALIVE, pred_p)$ *during the last* $\Delta_p(pred_p)$ *ticks of p's clock* **then**
            r-broadcast $(SUSPICION, p, pred_p)$

    || <u>Task 3</u>: **when r-deliver**$(SUSPICION, q, r)$
        $Balance_p(r) \leftarrow Balance_p(r) + 1$
        $update\_pred\_and\_succ()$
        if $r = p$ **then**
            r-broadcast $(REFUTATION, p)$

    || <u>Task 4</u>: **when r-deliver**$(REFUTATION, q)$
        $Balance_p(q) \leftarrow Balance_p(q) - 1$
        $\Delta_p(q) \leftarrow \Delta_p(q) + 1$
        $update\_pred\_and\_succ()$

coend

As shown in Algorithm 1, each process sends heartbeats to its successor in the ring, and monitors its predecessor by hearing heartbeats from it. Every process p uses a $Balance_p$ variable for every process q, accounting suspicions and refutations for q. If $Balance_p(q) > 0$, with $q \neq p$, then p suspects q; else, q is trusted by p. This way, $Balance_p$ provides the properties of $\diamond\mathcal{P}$. Every process p starts sending periodically an $(ALIVE, p)$ message to its successor in the ring, denoted by the variable $succ_p$ (Task 1). Also, every process p waits for periodical $(ALIVE, pred_p)$ messages from its predecessor in the ring, denoted by the variable $pred_p$. If p does not receive such a message on a specific time-out interval of $\Delta_p(pred_p)$, then p suspects that $pred_p$ has crashed, and r-broadcasts a $(SUSPICION, p, pred_p)$ message (Task 2). In Task 3, when p r-delivers a $(SUSPICION, q, r)$ message, p increments $Balance_p(r)$ and calls the $update\_pred\_and\_succ$ procedure. Besides this, if $r = p$, i.e., p has been erroneously suspected by q, p r-broadcasts a $(REFUTATION, p)$ message. In Task 4, when p r-delivers a $(REFUTATION, q)$ message, p decrements $Balance_p(q)$, increments $\Delta_p(q)$, and calls the $update\_pred\_and\_succ$ procedure. Variables $pred_p$ and $succ_p$ are updated from $Balance_p$ to the nearest predecessor and the nearest successor in the ring having a non-positive balance respectively. If all the components of the $Balance_p$ vector are positive, then p sets both $pred_p$ and $succ_p$ to p.

3.2. One-to-one communication based optimal $\diamond\mathcal{P}$

In this section, we propose a second communication-optimal algorithm implementing $\diamond\mathcal{P}$, that uses one-to-one communication exclusively. As shown in Algorithm 2 every process p uses a $Suspected_p$ list to provide the properties of $\diamond\mathcal{P}$. Every process p starts sending periodically an $(ALIVE, p, Suspected_p)$ message to its successor in the ring, denoted by the variable $succ_p$ (Task 1). Also, every process p waits for periodical $(ALIVE, pred_p, -)$ messages from its predecessor in the ring, denoted by the variable $pred_p$. If p does not receive such a message on a specific time-out interval of $\Delta_p(pred_p)$, then p suspects that $pred_p$ has crashed, includes $pred_p$ in its list $Suspected_p$, and sends a $(SUSPICION, p, pred_p)$ message to $pred_p$ (Task 2). As we will see, if $pred_p$ has not crashed, it will reply to p with a $(REFUTATION, pred_p)$ message. Finally, p calls the $update\_pred\_and\_succ$ procedure, which updates its $pred_p$ and $succ_p$ variables to its nearest unsuspected predecessor and successor in the ring, respectively.

In Task 3, when a process p receives a $(SUSPICION, q, p)$ message for some q, p knows that it has been erroneously suspected by q, and hence p sends a $(REFUTATION, p)$ message to q. Also, p sets $succ_p$ to q. In Task 4, when a process p receives a $(REFUTATION, q)$ message for some q, p knows

Algorithm 2: Communication-optimal $\diamond\mathcal{P}$ using exclusively one-to-one communication.

{*Every process p executes the following*}

Procedure $update\_pred\_and\_succ()$
- $pred_p \leftarrow p$'s nearest predecessor r in the ring such that $r \notin Suspected_p$
- $succ_p \leftarrow p$'s nearest successor r in the ring such that $r \notin Suspected_p$

$pred_p \leftarrow pred(p)$
$succ_p \leftarrow succ(p)$
forall $q \in \Pi$ **do**
- $\Delta_p(q) \leftarrow$ default time-out interval

$Suspected_p \leftarrow \emptyset$

cobegin

|| **Task 1: repeat periodically**
- **if** $succ_p \neq p$ **then**
  - send $(ALIVE, p, Suspected_p)$ to $succ_p$

|| **Task 2: repeat periodically**
- **if** *$(pred_p \neq p)$* **and** *p didn't receive $(ALIVE, pred_p, -)$ during the last $\Delta_p(pred_p)$ ticks of p's clock* **then**
  - $Suspected_p \leftarrow Suspected_p \cup \{pred_p\}$
  - send $(SUSPICION, p, pred_p)$ to $pred_p$
  - $update\_pred\_and\_succ()$

|| **Task 3: when** receive $(SUSPICION, q, p)$ for some q
- send $(REFUTATION, p)$ to q
- $succ_p \leftarrow q$

|| **Task 4: when** receive $(REFUTATION, q)$ for some q
- $Suspected_p \leftarrow Suspected_p - \{q\}$
- $\Delta_p(q) \leftarrow \Delta_p(q) + 1$
- $update\_pred\_and\_succ()$

|| **Task 5: when** receive $(ALIVE, pred_p, Suspected_{pred_p})$ from $pred_p$
- **forall** $q \in \Pi$ **except** $pred_p$ *and* p **do**
  - **if** $q \in Suspected_{pred_p}$ **and** $q \notin Suspected_p$ **then**
    - $Suspected_p \leftarrow Suspected_p \cup \{q\}$
    - send $(SUSPICION, p, q)$ to q
- $update\_pred\_and\_succ()$

coend

that it erroneously suspected q, and hence p removes q from its list $Suspected_p$, increments $\Delta_p(q)$ in order to avoid future erroneous suspicions, and calls the $update\_pred\_and\_succ$ procedure.

Since suspicions are not broadcast to all processes in Task 2, we need a mechanism allowing processes to learn about suspicions made by other processes. Task 5 implements such a mechanism, and is activated whenever a process p receives an $(ALIVE, pred_p, Suspected_{pred_p})$ message from $pred_p$. For every process q in the system except $pred_p$ and p, p verifies if q is suspected by $pred_p$ but not by p, in which case p includes q in its list $Suspected_p$, and sends a $(SUSPICION, p, q)$ message to q. Finally, p calls the $update\_pred\_and\_succ$ procedure.

A drawback of this one-to-one approach is that the suspicion messages are not sent until the suspected list piggybacked into heartbeat messages is received by Task 5, resulting in a higher latency for the detection of a crashed process by the rest of processes. We address this issue in the following subsection.

3.3. One-to-all communication to reduce the detection latency

We present here a modification to Algorithm 2 that reduces the detection latency of real failures by sending additional messages upon suspicions. We will evaluate the improvement of the modified algorithm in Section 4.2. The modification, presented in Algorithm 3, affects Task 2 and introduces a new task (Task 6).

Algorithm 3: Reducing the detection latency using one-to-all communication of suspicions.

{*Every process p executes the following*}

...
cobegin
  ...
  || **Task 2: repeat periodically**
    if $(pred_p \neq p)$ **and** *p didn't receive* $(ALIVE, pred_p, -)$ *during the last* $\Delta_p(pred_p)$ *ticks of p's clock* **then**
      $Suspected_p \leftarrow Suspected_p \cup \{pred_p\}$
      send $(SUSPICION, p, pred_p)$ to $pred_p$
(=>)   send $(SUSP\_TO\_ALL, p, pred_p)$ to all **except** $pred_p$ and p
      $update\_pred\_and\_succ()$
  ...
(=>) || **Task 6: when** receive $(SUSP\_TO\_ALL, q, r)$ for some $r \neq p$ for some q
(=>) $Suspected_p \leftarrow Suspected_p \cup \{r\}$
(=>)   send $(SUSPICION, p, r)$ to r
(=>) $update\_pred\_and\_succ()$
coend

In Task 2, when p suspects that $pred_p$ has crashed, besides sending the ($SUSPICION$, p, $pred_p$) message to $pred_p$ (Line 16), p also sends a message ($SUSP\_TO\_ALL$, p, $pred_p$) to all processes except $pred_p$ and p. This new type of message is handled in Task 6. There, when a process p receives a ($SUSP\_TO\_ALL$, q, r) message for some q, p includes r in $Suspected_p$, sends a ($SUSPICION$, p, r) to verify if r has crashed, and calls the procedure $update\_pred\_and\_succ$. Observe that if r has really crashed then the suspicion will be permanent, because r will not send any $REFUTATION$ message to p.

It is simple to see that the proposed modification speeds-up the detection of real failures, since the rest of processes will send almost simultaneously the $SUSPICION$ message to the suspected process, while in Algorithm 2 the $SUSPICION$ messages are sent following the ring by means of Task 5. On the other hand, in the case of an erroneous suspicion the proposed modification introduces an additional overhead of approximately $3n$ messages ($SUSPICION$ and $REFUTATION$).

The modification does not affect neither the correctness of the algorithm nor its communication optimality, since after the stabilization of the ring no more $SUSPICION$ or $SUSP\_TO\_ALL$ messages will be sent.

4. Analysis and performance evaluation

4.1. Complexity analysis

Table 1 summarizes the communication costs of the communication-optimal algorithms presented in this paper, in terms of the number of unidirectional links used forever (which corresponds eventually to the number of regular heartbeat messages exchanged periodically) and the number of messages needed to manage an erroneous suspicion. Chandra-Toueg's all-to-all algorithm is also included for comparative purposes.

Algorithm	Periodic cost (#links used forever)	Sporadic cost (#msgs to manage a suspicion)
Algorithm 1	$\mathcal{C}$	$2n^2$
Algorithm 2	$\mathcal{C}$	$2n$
Algorithm 3	$\mathcal{C}$	$3n$
Chandra-Toueg [4]	$\mathcal{C}(n-1)$	0

Table 1. Communication costs of different algorithms implementing $\diamond\mathcal{P}$.

As it can be observed, Algorithms 2 and 3, while communication-optimal, have a linear overhead for managing an erroneous suspicion. The benefits obtained with respect to the quadratic communication-optimal Algorithm 1 are evident, and can be explained by the fact that now suspicions and refutations are managed following one-to-one (Algorithm 2) one-to-all (Algorithm 3) communication patterns, respectively. As explained in the introduction, Algorithm 1 uses a *reliable* one-to-all communication pattern, which considering the implementation of Reliable Broadcast, results in practice in an all-to-all pattern.

4.2. Performance evaluation

Besides communication optimality, there are QoS measures that are of interest when evaluating the performance of failure detector algorithms. We have considered here two different performance measures to compare the algorithms presented. The first measure is related to the accuracy of the information provided to querying processes. In particular, we focus on the *query accuracy probability*, defined as the probability that a failure detection module which is queried by its associated process gives the right answer. This measure is based on [5], but has been extended in this work to scenarios with more than two processes. The second measure tries to quantify how fast the failure detector reacts. This has been measured by the time interval between the crash of a process and the time in which the rest of the processes suspect it in a permanent way.

To test the comparative performance of the algorithms, we have used the ns-2 simulator (http://www.isi.edu/nsnam/ns/). In Table 2 we show the simulation settings for a typical local area network scenario. The simulation generates message delays at random with a uniform distribution. However, we have set minimum and maximum message bounds. Apparently, this contradicts our partially synchronous system model. Nevertheless, the algorithms do not exploit the knowledge of the maximal message delay when initializing the timeouts. This allows us to generate erroneous suspicions under the same conditions for both algorithms. Moreover, from a practical point of view the setting of a maximum message delay allows to determine the duration of the simulations.

Parameter	Value
Minimum message delay	0.001
Maximum message delay	0.005
Periodicity of $ALIVE$ messages	0.5
Initial timeouts	0.5
Timeout increment	0.001

Table 2. Simulation settings (in seconds).

The tests have been carried out for a number of nodes going from 3 to 24, using the settings of Table 2. The bad answer probability has been measured executing the algorithm during 2000 seconds, that has been empirically proved to be sufficient for comparative purposes. In fact, after this time the simulations have either stabilized or are near stabilization. We assume that no process crashes during the 2000 seconds. This assumption does not really lose any generality. On the one hand, in our algorithms erroneous suspicions are actually more complex to handle than real crashes. On the other hand, although a crash during the execution of the Reliable Broadcast may delay the delivery of the message, the probability of such a failure in practice is very low. Also, this delay is really small in a LAN, thus our assumption has not any impact in the accuracy of the failure detector. The crash detection time has been measured in a longer execution, introducing a crash in a time instant (2500 seconds) in which the system is stabilized. In both cases, every simulation has been executed a sufficiently large number of times.

Figures 1 and 2 show the average results obtained.

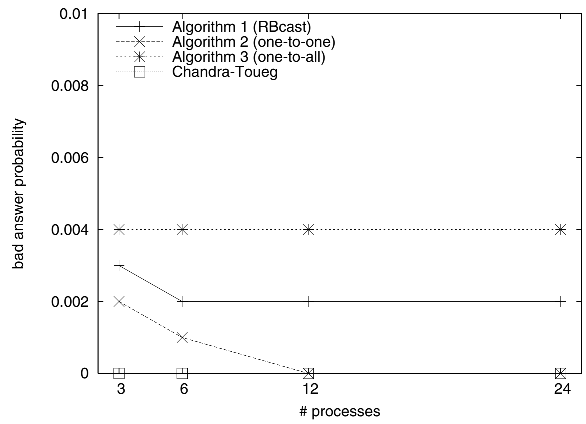

Figure 1. Query accuracy, expressed as bad answer probability (percentage).

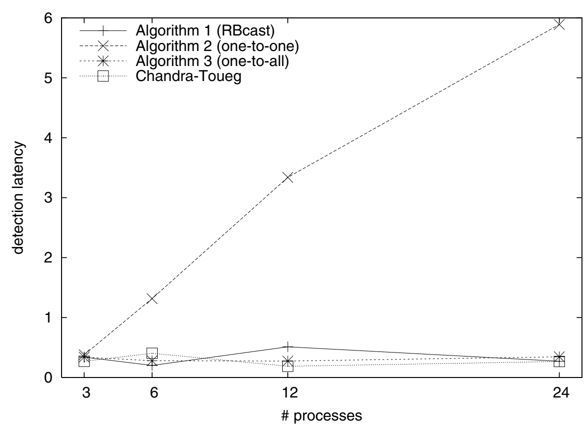

Figure 2. Crash detection latency (in seconds).

In Figure 1, for clarity, values express the complement of the right answer probability, i.e., the probability that a failure detection module gives a wrong answer. The bad answer probability is low for all the communication optimal algorithms, and does not increase with the number of processes. Although Algorithm 2 uses less messages to manage suspicions, the bad answer probability is lower for this algorithm than for Algorithm 1. This result is due to the fact that in Algorithm 2 when a process suspects its predecessor in the ring the rest of processes do not receive any information about the new suspicion most of the times. In fact, in practice the refutation message from a suspected process q usually arrives to the suspecting process p before the next heartbeat message is sent by p. This reduces the number of false suspicions in the system and makes the bad answer probability near negligible. For Chandra-Toueg's algorithm the bad answer probability is negligible too, at the cost of using an all-to-all communication pattern periodically and forever.

In Figure 2, it can be observed that Algorithm 2 has a higher crash detection latency than the other algorithms. Even worse, the detection latency increases linearly with the number of processes, hence the algorithm does not scale well for a large number of processes. The same mechanism that makes Algorithm 2 more accurate when false suspicions occur makes it slower for real crashes. If we consider the improvement presented in Algorithm 3, we observe that the crash detection latency is constant in this case and similar to the crash detection latency of Algorithm 1. Note that Reliable Broadcast involves a quadratic number of messages to manage suspicions, while the improvement used to reduce the crash detection latency of Algorithm 3 keeps the extra messages linear. Going back to Figure 1, it can be observed that the bad answer probability of Algorithm 3 is slightly higher because it takes a bit longer to correct false suspicions. In order to get an optimal performance, it could be interesting to switch from Algorithm 2 to Algorithm 3 once the system is considered stabilized.

5. Conclusion

In this paper, we have analysed three communication-optimal algorithms implementing the $\diamond\mathcal{P}$ failure detector class. The first algorithm uses Reliable Broadcast to communicate suspicions and refutations, involving a quadratic number of messages. The second algorithm uses one-to-one communication exclusively, involving a lower overhead to manage suspicions. The third algorithm consists in adding sporadic one-to-all communication to the second one, in order to improve the crash detection time.

We have evaluated the performance of the algorithms in terms of two QoS measures: one of them is related to the accuracy of the information provided by the failure detector and the other concerns the crash detection time. Although the algorithm using one-to-one communication is better in terms of accuracy, it does not scale well when the crash detection latency is considered. That's the reason why we have proposed the third algorithm, that uses some extra messages when a suspicion occurs, reducing the crash detection latency from linear to constant. This new algorithm also involves a linear number of messages to manage a suspicion.

An interesting aspect regarding the second and third algorithms presented in this paper is that they could be used together, switching from the second to the third once the system is considered stabilized.

References

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

[2] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'2004)*, pages 328–337, St. John's, Newfoundland, Canada, July 2004.

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

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

[5] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(5):561–580, 2002.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[7] R. Guerraoui, M. Kapalka, and P. Kouznetsov. The weakest failure detector to boost obstruction-freedom. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC'2006)*, pages 399–412, Stockholm, Sweden, September 2006. LNCS 4167, Springer-Verlag.

[8] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000)*, pages 52–59, Nurenberg, Germany, October 2000.

[9] M. Larrea, A. Lafuente, I. Soraluze, R. Cortiñas, and J. Wieland. Designing efficient algorithms for the eventually perfect failure detector class. *Journal of Software*, 2(4):1–11, October 2007.

[10] M. Larrea, A. Lafuente, I. Soraluze, R. Cortiñas, and J. Wieland. On the implementation of communication-optimal failure detectors. In *Proceedings of the Third Latin-American Symposium on Dependable Computing (LADC'2007)*, pages 25–37, Morelia, Mexico, September 2007. LNCS 4746, Springer-Verlag.

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

[12] W. Wu, J. Cao, J. Yang, and M. Raynal. A hierarchical consensus protocol for mobile ad hoc networks. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'2006)*, pages 64–72, Montbeliard-Sochaux, France, February 2006. IEEE Computer Society.