Journal of Computer and System Sciences 81 (2015) 383–397

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Communication-optimal eventually perfect failure detection in partially synchronous systems ☆

Alberto Lafuente *, Mikel Larrea, Iratxe Soraluze, Roberto Cortiñas

University of the Basque Country UPV/EHU, Faculty of Computer Science, Lardizabal 1, 20018 San Sebastián, Spain

ARTICLE INFO

Article history:
Received 11 March 2009
Received in revised form 18 July 2013
Accepted 19 May 2014
Available online 2 July 2014

Keywords:
Distributed algorithms
Fault tolerance
Consensus
Partial synchrony
Unreliable failure detectors
Communication optimality

ABSTRACT

Since Chandra and Toueg introduced the failure detector abstraction for crash-prone systems, several algorithms implementing failure detectors in partially synchronous systems have been proposed. Their performance can be measured by their *Communication efficiency*, defined as the number of links used forever. In this regard, in a *communication-efficient* algorithm only n links are used forever, n being the number of processes in the system. In this paper, we present *communication optimality*, a communication efficiency degree reached when only c links are used forever, c being the number of correct processes. We show that c is the minimum number of links used forever required to implement $\diamond\mathcal{P}$ and that c is also optimal for $\diamond\mathcal{S}$ and Ω when $c < n$. Finally, we propose two communication-optimal $\diamond\mathcal{P}$ algorithms following respectively one-to-all and one-to-one communication patterns to manage suspicions, showing that there is a trade-off between detection latency and sporadic communication overhead.

1. Introduction

1.1. Background and related work

Unreliable failure detectors, proposed by Chandra and Toueg in [1], are a mechanism providing (possibly incorrect) information about process failures. This mechanism has been used to solve several problems in asynchronous distributed systems where processes may crash by prematurely halting, in particular the *consensus* problem [2]. The specific classes of failure detectors proposed in [1] are defined by a *completeness* property, which characterizes the failure detector's capability of suspecting incorrect processes, and by an *accuracy* property, which restricts the mistakes the failure detector can make. More specifically, Chandra and Toueg defined, among others, the following two completeness properties and two accuracy properties that a failure detector may satisfy:

- *Strong Completeness:* eventually every process that crashes is permanently suspected by *every* correct process.
- *Weak Completeness:* eventually every process that crashes is permanently suspected by *some* correct process.
- *Eventual Strong Accuracy:* there is a time after which correct processes are not suspected by any correct process.
- *Eventual Weak Accuracy:* there is a time after which some correct process is never suspected by any correct process.

☆ Research partially supported by the Spanish Research Council, under grant TIN2013-41123-P, the Basque Government, under grants IT395-10 and S-PE12UN109, and the University of the Basque Country UPV/EHU, under grant UFI11/45.

* Corresponding author. Fax: +34 943015590.
E-mail addresses: alberto.lafuente@ehu.es (A. Lafuente), mikel.larrea@ehu.es (M. Larrea), iratxe.soraluze@ehu.es (I. Soraluze), roberto.cortinas@ehu.es (R. Cortiñas).

http://dx.doi.org/10.1016/j.jcss.2014.06.010
0022-0000/

Table 1
Four classes of failure detectors.

	Eventual strong accuracy	Eventual weak accuracy
Strong completeness	Eventually perfect ($\diamond\mathcal{P}$)	Eventually strong ($\diamond\mathcal{S}$)
Weak completeness	Eventually quasi-perfect ($\diamond\mathcal{Q}$)	Eventually weak ($\diamond\mathcal{W}$)

The combination of these properties gives us four classes of failure detectors, which are shown in Table 1. Consensus can be solved using a failure detector of any of those four classes. In particular, there is another failure detector class denoted by Ω, which is equivalent[1] to $\diamond\mathcal{S}$ and $\diamond\mathcal{W}$, and which has been proved to be sufficient and necessary, i.e., weakest, for solving consensus [3]. The Ω failure detector class provides eventual agreement on a common and correct *leader* among all non-faulty processes in a system.

Specific algorithms for implementing Ω and/or $\diamond\mathcal{S}$ have been proposed, e.g. [4–11]. Observe that $\diamond\mathcal{P}$ trivially satisfies the properties of $\diamond\mathcal{S}$. Also, $\diamond\mathcal{P}$ can be easily transformed into Ω, e.g., by choosing as leader the non-suspected process with the lowest identifier.

Failure detectors of the class $\diamond\mathcal{P}$, being strictly stronger than Ω and $\diamond\mathcal{S}$,[2] can also be used to solve consensus. Several algorithms implementing $\diamond\mathcal{P}$ have been proposed in the literature. The algorithm proposed by Chandra and Toueg in [1] uses a heartbeat mechanism and all-to-all communication to detect faulty processes. The algorithms proposed by Aguilera et al. in [12] and by Larrea et al. in [13] use heartbeats too, and rely on a leader-based approach. On the other hand, the algorithms proposed by Larrea et al. in [14,15] use a polling—or query/reply—mechanism on a ring arrangement of processes. Roughly speaking, the leader-based and the ring-based algorithms are more efficient than the all-to-all algorithm regarding the number of sent messages (linear vs. quadratic). Observe also that, compared to polling, the heartbeat mechanism reduces the number of messages to the half. Moreover, heartbeats inherently provide a certain level of communication reliability in a system with fair lossy links, while polling usually requires reliable communication.

Chandra and Toueg showed in [1] that $\diamond\mathcal{Q}$ can also be used to solve consensus. To do so, they first showed that classes $\diamond\mathcal{Q}$ and $\diamond\mathcal{P}$ are equivalent from a problem solvability point of view, i.e., a problem which is solvable with $\diamond\mathcal{P}$ is also solvable with $\diamond\mathcal{Q}$ and vice versa. It is worth noting that the equivalence of $\diamond\mathcal{Q}$ and $\diamond\mathcal{P}$ does not come for free, i.e., not all failure detectors in $\diamond\mathcal{Q}$ are in $\diamond\mathcal{P}$. Instead, it means that any failure detector in $\diamond\mathcal{Q}$ can be extended with a simple distributed algorithm to obtain a failure detector in $\diamond\mathcal{P}$.

Since consensus cannot be solved in crash-prone, pure asynchronous systems [2], algorithms that implement failure detectors make some weak timing assumptions. Specifically, a partially synchronous model [1,16] is considered in this work. In such a model, 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. In practice, the bounds depend on parameters such as the network speed or the size of the system, and hold easier in, for example, local area networks than in wide area networks. Actually, the bounds must exist and hold only for the links that connect correct processes. Hence, it is important to design failure detection algorithms that use a low number of links, e.g., by arranging the processes in a logical ring topology.

As shown recently in [17], algorithms for the class $\diamond\mathcal{P}$ combining a heartbeat-based detection mechanism on a logical ring arrangement of processes outperform the aforementioned ones in terms of the number of links permanently used, while preserving good quality-of-service. In this regard, algorithms in [17] are communication-efficient following Aguilera et al. [12], i.e., eventually only n unidirectional links are used forever. With regard to this performance measure, heartbeat-based ring algorithms outperform other ring algorithms based on polling [14,15] or algorithms using a centralized communication pattern [12,13]. By all means, algorithms using an all-to-all communication pattern, such as Chandra–Toueg's algorithm [1], are far from being communication-efficient.

Other failure detection algorithms specifically designed for wide area networks, e.g. [18], are based on local failure detectors that provide their properties in a neighborhood of processes (using all-to-all communication inside each neighborhood), and propagate the information about suspicions among neighborhoods. The advantage of a ring based approach is that, once the logical ring is defined, neighborhoods are implicit, and eventually involve just two processes for every correct process, i.e., its correct predecessor and correct successor in the ring.

1.2. Implementing failure detectors

Obviously, an algorithm implementing a failure detector of a given class must satisfy the properties defined for that class. When implementing a failure detector, besides satisfying the properties of the class it belongs to, performance should also be taken into account. In this work, we focus on the following two performance issues:

[1] Two failure detectors are equivalent if they are reducible to each other [1]. Informally, a failure detector D is reducible to a failure detector D' if there is a distributed algorithm that can transform D into D'. The concepts of reducibility and equivalence can be naturally extended to classes of failure detectors.

[2] While $\diamond\mathcal{P}$ can be transformed into Ω or $\diamond\mathcal{S}$ *asynchronously*, i.e, without any additional synchrony assumption, neither Ω nor $\diamond\mathcal{S}$ can be transformed into $\diamond\mathcal{P}$.

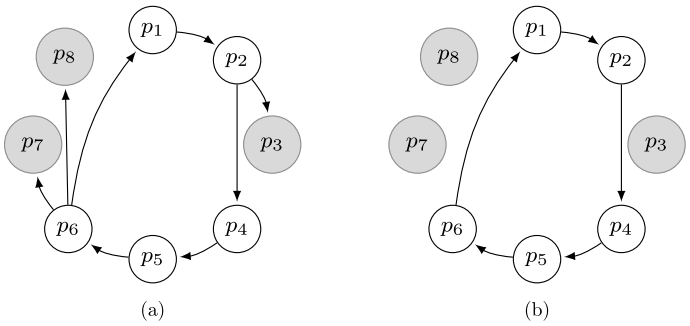


Fig. 1. Links used permanently in (a) a communication-efficient algorithm, and (b) a communication-optimal algorithm.

i) Scalability, which allows a failure detector to be deployed in networks with a high number of nodes and/or heterogeneous links. A scalable failure detection algorithm should use a low number of links and avoid all-to-all communication patterns.
ii) Quality-of-service, which involves several parameters, such as detection latency or stabilization time [19]. They allow to measure the responsiveness of the system. For example, when a crashed process q is suspected by a process p, every correct process should be informed as soon as possible in order to provide low detection latencies. In this regard, a one-to-all communication pattern for suspicion propagation can be helpful. However, such a communication pattern could become a drawback when the suspicion is erroneous, since it propagates the erroneous suspicion in the system.

Keeping in mind scalability and quality-of-service, we now consider a set of interesting properties for algorithms that implement a failure detector:

Communication efficiency. In a communication-efficient algorithm only n unidirectional links are used forever, n being the number of processes in the system [12]. Observe that communication efficiency refers to a *permanent* behavior that will hold eventually. In large systems, communication-efficient algorithms will be more scalable than non-communication-efficient algorithms.

Low sporadic overhead. Besides the permanent communication cost due to periodic messages (also known as heartbeats), which is addressed by the previous communication efficiency property, an algorithm implementing a failure detector can involve *sporadic* extra messages caused by failure suspicions, resulting in a peak overhead. There is a trade-off in the way this sporadic traffic is managed. On the one hand, a one-to-one communication pattern can be used (e.g., by using a logical ring arrangement); this way communication overhead is reduced and, henceforth, it provides better scalability. On the other hand, one-to-all (or even all-to-all) communication leads to provide low detection latencies.

Communication locality. This property is twofold. On the one hand, we will say that a failure detection algorithm has *periodic communication locality* when periodic, permanent monitoring messages (i.e., heartbeats), are sent to some process(es) in the neighborhood of the sender process. Observe that a logical ring arrangement of processes based on proximity provides this property naturally. Similarly, we will say that a failure detection algorithm has *sporadic communication locality* when messages sent as a consequence of a suspicion are sent likely to some process(es) in the neighborhood of the suspecting process. Again, trade-offs should be considered between the good scalability associated to communication locality, and the potential good quality-of-service provided by the use of one-to-all and all-to-all communication patterns.

1.3. Using communication-optimal $\diamond\mathcal{P}$ to solve consensus

Despite $\diamond\mathcal{S}$ and Ω have been extensively used to solve consensus [1,3,20–23], we focus our work on implementing *communication-optimal* failure detectors of the class $\diamond\mathcal{P}$. As we will show, in a communication-optimal $\diamond\mathcal{P}$ algorithm only c unidirectional links are used forever, c being the number of correct processes. The choice of $\diamond\mathcal{P}$ is mainly justified by the fact that, as we will also show, communication optimality is *almost the same* for $\diamond\mathcal{P}$, $\diamond\mathcal{S}$ and Ω (they all require the same number c of links used forever when $c < n$), and by the fact that a failure detector of the class $\diamond\mathcal{P}$ trivially implements $\diamond\mathcal{S}$ and Ω. Hence, from a resource usage viewpoint, solutions based on either $\diamond\mathcal{S}$, Ω, or on an equivalent leader election mechanism, e.g. [24–26,13], can also execute efficiently on top of communication-optimal implementations of $\diamond\mathcal{P}$. Moreover, for certain problems [27] and consensus protocols [28] failure detector $\diamond\mathcal{P}$, being stronger than $\diamond\mathcal{S}$ and Ω, is required. Finally, failure detectors of the class $\diamond\mathcal{P}$ are more natural, in the sense that all the correct processes can provide a precise set composed of exclusively crashed processes, providing stronger accuracy than $\diamond\mathcal{S}$ and Ω (Eventual Strong vs Eventual Weak).

Fig. 1 shows an example of the number of unidirectional links used permanently for a system composed of eight processes, out of which five are correct, i.e., $n = 8$ and $c = 5$. Faulty processes are represented by gray circles. Observe that in

a communication-efficient algorithm, e.g. [17], n links are used permanently,[3] while in a communication-optimal algorithm only c links are used permanently, which is optimal when implementing $\diamond\mathcal{P}$.

1.4. Summary of contributions

The contributions of this paper are the following:

1. We study the communication efficiency of implementing $\diamond\mathcal{P}$ in partially synchronous systems where processes can fail by crashing, and introduce the notion of *communication optimality*.
2. We show that the minimum number of unidirectional links used forever needed for an algorithm to provide the properties of $\diamond\mathcal{P}$ is c, i.e., the number of correct processes in the system.
3. We show that c is also minimal for Ω when at least one process crashes, i.e., $c < n$. Hence, communication optimality is almost the same for $\diamond\mathcal{P}$, $\diamond\mathcal{S}$ and Ω, which makes communication-optimal $\diamond\mathcal{P}$ algorithms good candidates to be used in a consensus algorithm.
4. We show that communication-optimal $\diamond\mathcal{P}$ algorithms can be implemented. More precisely, we propose two ring-based communication-optimal $\diamond\mathcal{P}$ algorithms; one of them using a one-to-all communication pattern for communicating suspicions, and the second one using exclusively a one-to-one communication pattern. Hence, we close algorithmically the gap in efficiency with respect to [17] (both the present contribution and [17] consider the same system model).

With respect to the communication efficiency property, the two approaches proposed in this paper lead to communication-optimal $\diamond\mathcal{P}$ algorithms, i.e., eventually only c links are used forever. Concerning the sporadic overhead involved by a suspicion, the first algorithm has a higher overhead due to the use of a reliable broadcast communication primitive, while the second algorithm has a low overhead. Finally, regarding communication locality, the first algorithm has only periodic communication locality, while the second algorithm has both periodic and sporadic communication locality. In this regard, our notion of neighborhood is dynamic and related to the estimation of the correct predecessor and successor of a process in the ring. Note that according to the properties of $\diamond\mathcal{P}$, the neighborhood of every correct process will eventually stabilize. If the logical ring is arranged using proximity criteria, e.g., the number of physical hops between consecutive processes in a wide area network, our second algorithm minimizes the network traffic, which could provide benefits in geographically dispersed networks.

1.5. Roadmap

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 show the communication optimality results for $\diamond\mathcal{P}$ and Ω. In Section 4, we give two communication-optimal $\diamond\mathcal{P}$ algorithms. In Section 5, we analyze the performance of the algorithms, and compare them with previously proposed $\diamond\mathcal{P}$ algorithms. Finally, Section 6 concludes the paper.

2. System model

We consider a distributed system composed of a finite set Π of $n > 1$ processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$, which communicate only by sending and receiving messages. We also use the alternative notation $p, q, r, \ldots$ to denote processes. Every pair of processes (p, q) is connected by two unidirectional and reliable logical communication links $p \to q$ and $q \to p$. This means that process p can send a message to process q using a *send* primitive and vice versa. The definition of reliable link that we consider is the following: if both the sender and the receiver do not crash, then all messages that are sent are eventually received. Reliable communication is usually implemented using retransmission techniques and acknowledgment messages.

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 by *correct*, and the subset of processes that do fail, denoted by *crashed*. We use 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., $c = |correct| \geq 1$.

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 $succ(p)$ to respectively denote the predecessor and the successor of a process p in the ring.

Concerning timing assumptions, we consider a partially synchronous model [1,16] 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*). The communication links supporting this behavior are also called *eventually timely* links [29]. Our model is actually a variant of the models of partial synchrony of [1,16]. The difference is that we assume reliable communication links connecting correct processes in

[3] In the algorithm in [17], for every process (correct or faulty), the link coming from its correct predecessor in the ring is used permanently.

Algorithm 1: Reliable Broadcast by message diffusion.

```
1 { Every process p executes the following }
2 To execute R-broadcast(m):
3     send m to all (including p)
4 R-deliver(m) occurs as follows:
5     when receive m for the first time do
6         if sender(m) ≠ p then
7             send m to all
8         R-deliver(m)
```

a ring. Since a reliable link which is eventually timely actually is always timely, in our model there is an unknown bound on message transmission time that always holds for these reliable links. When presenting the algorithms, we will refine the minimal assumptions on communication reliability and synchrony required by each algorithm.

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.

2.1. Reliable broadcast

Reliable Broadcast is a communication primitive for asynchronous systems that we use in one of our algorithms. It guarantees that all correct processes deliver the same set of messages. This set includes at least all messages broadcast by correct processes. Formally, Reliable Broadcast is defined in terms of two primitives, *R-broadcast*(m) and *R-deliver*(m), and satisfies the following properties [30]:

- Validity. If a correct process R-broadcasts a message m, then it eventually R-delivers m.
- Agreement. If a correct process R-delivers a message m, then all correct processes eventually R-deliver m.
- Uniform integrity. For any message m, every process R-delivers m at most once, and only if m was previously R-broadcast by $sender(m)$.[4]

Algorithm 1, proposed in [1], presents a simple Reliable Broadcast algorithm for asynchronous systems with up to $n-1$ crash failures. Informally, when a process receives a message for the first time, it relays the message to all processes and then R-delivers it.[5] Although in [1] all the links are considered reliable, observe that a ring of reliable links connecting correct processes is sufficient for Algorithm 1 to work. As a particular case, consider a system where only those reliable links exist. In that case, a message m that is broadcast would exactly make a complete tour of the ring, such that every correct process relays (and delivers) m once.

3. On communication optimality

In this section, we show that c, i.e., the number of correct processes in the system, is the minimum number of unidirectional links used forever necessary for an algorithm to provide the properties of $\diamond\mathcal{P}$. Then, we show that, assuming that at least one process crashes, i.e., $c < n$, c is also minimal for implementing Ω.[6] Both results hold when there are at least two correct processes in the system, i.e., $c \geq 2$.

Theorem 1. *c is the minimum number of unidirectional links used forever necessary for an algorithm to provide the properties of $\diamond\mathcal{P}$ in a crash-prone system.*

Proof. Otherwise, if only less than c unidirectional links were used forever, there would be some correct process p that eventually would stop sending messages. Let t be the time instant in which p stops sending messages. Consider now another run R', identical to R until time t, and assume that p crashes at time t in R'. For any correct process q, if q does not eventually and permanently suspect p, then the strong completeness property of $\diamond\mathcal{P}$ is violated. Hence, q will eventually and permanently suspect p in R'. Observe that both executions R and R' are indistinguishable for q. Hence, in run R q will also eventually and permanently suspect p, violating the eventual strong accuracy property of $\diamond\mathcal{P}$. □

Theorem 2. *If at least one process crashes, i.e., $c < n$, then c is the minimum number of unidirectional links used forever necessary for an algorithm to provide the property of Ω in a crash-prone system.*

[4] We assume that messages include the identity of the sender and a sequence number, which make every message unique.

[5] An optimization consists in not relaying m to p, $sender(m)$ and the process q from which m has been received for the first time (if $q \neq sender(m)$).

[6] By the equivalence relation between Ω and $\diamond\mathcal{S}$ [1], the reasoning regarding Ω applies to $\diamond\mathcal{S}$ too.

Proof. The proof is by contradiction. Assume that we have an implementation of Ω in which only $c-1$ unidirectional links are used forever (by the leader in order to propagate heartbeat messages). Consider a run R of the algorithm in which c processes are correct and let t be the time instant after which only $c-1$ unidirectional links are used forever. Consider now another run R', identical to R until time t, and assume that a process q different from the leader, which is correct in R, crashes at time t in R'. Observe that both executions R and R' are indistinguishable for the leader, and there is no way for the leader to know that q has crashed, and hence it will not stop sending messages to q. Since the number of correct processes in run R' is $c-1$, the algorithm should use only $c-2$ unidirectional links forever, which contradicts the fact that the leader will not stop sending messages to q. □

Aguilera et al. propose in [31] an algorithm implementing Ω such that eventually only f links are used forever, f being the maximum number of processes that can crash. They also show that in the crash failure model no algorithm using fewer than f links exists. Hence, if $f=n-1$ (as in our system model), Ω can be implemented with $n-1$ links used forever, even if no process crashes, i.e., $c=n$. The system model considered in [31] is weaker than ours, but, as indicated by the authors, it is too weak for implementing $\diamond\mathcal{P}$. Also, the algorithm of [31] uses always $n-1$ links, independently of the actual number of correct processes c. As we will see, the algorithms proposed in this paper, besides implementing $\diamond\mathcal{P}$, dynamically adapt the number of links used forever to the actual number of correct processes.

Similar results can be deduced from the work by Fernández et al. in [32]. They study the minimal system conditions to implement unreliable failure detectors, and focus on the set *Reach* of correct processes that can reach all correct processes via exclusively eventually timely links and other correct processes. They show that $\diamond\mathcal{P}$ cannot be implemented if *Reach* does not contain all the correct processes. Similarly, they show that $\diamond\mathcal{S}$ (and hence Ω) cannot be implemented if *Reach* does not contain at least one correct process. In both cases, the subgraph formed by correct processes and eventually timely links in their system model must contain at least c arcs (e.g., in a ring topology), with $c \leq n-1$ if at least one process crashes. In terms of our system model, these arcs correspond to our c links used forever.

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

In this section we introduce two approaches to the design of communication-optimal failure detection algorithms implementing $\diamond\mathcal{P}$. The approaches differ in how failure suspicions are managed. The first one uses an eager strategy in order to get low detection latencies. The second one is much more conservative in order to have a low sporadic communication overhead.

The first approach is based on every process consistently managing a local balance of suspicions and refutations for any other process. When a process p suspects another process q, p broadcasts a suspicion to every process, including q. If q has not failed, upon delivery of that suspicion it will broadcast a refutation in the same way. Suspicions increment the corresponding balance, while refutations decrement it. With this strategy, eventually every correct process will permanently have a positive balance for every incorrect process, and a zero balance for every correct process. Observe that a reliable broadcast of suspicions and refutations is required in order to have consistent balances.

The alternative approach to the global spread of suspicions and refutations is to let a process to manage only suspicions in its *neighborhood* in the ring. In this approach, a process p will be in charge of the detection of incorrect processes between p's correct predecessor in the ring and p. The ring arrangement is used to propagate information about failures, piggybacked on periodic heartbeat messages, to all processes in a lazy way.

4.1. An algorithm using reliable broadcast

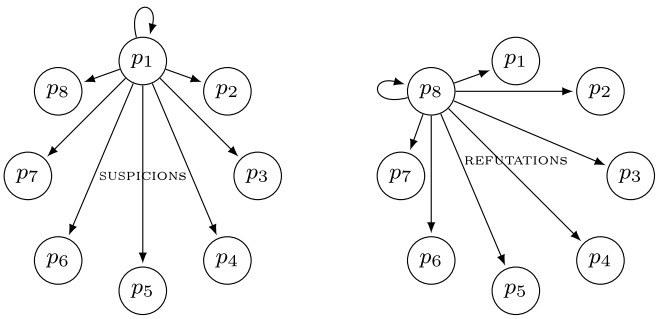
In this section, we propose a first communication-optimal implementation of $\diamond\mathcal{P}$ that uses Reliable Broadcast. In the algorithm, each process sends heartbeats to its successor in the ring, and monitors its predecessor by waiting heartbeats from it. Algorithm 2 presents the algorithm in detail, which uses a $Balance_p$ vector for every process p, accounting suspicions and refutations for every process. If $Balance_p(q) > 0$ with $q \neq p$, then p suspects q; else, q is trusted by p. As we will see, $Balance_p$ 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), as shown in Fig. 2a (p_1 suspects p_8). 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 (Fig. 2b). In Task 4, when p R-delivers a (REFUTATION, q) message, p decrements $Balance_p(q)$, increments $\Delta_p(q)$, in order to avoid premature suspicions in the future, 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.[7] If all the components of the $Balance_p$ vector are positive, then p sets both $pred_p$ and $succ_p$ to p.

[7] Here we informally use the terms *nearest predecessor* (or *nearest successor*) of a process p to denote the first process preceding (or succeeding) p following the ring arrangement and fitting a particular condition.

Algorithm 2: Communication-optimal $\diamond\mathcal{P}$ using Reliable Broadcast.

{Every process p executes the following}

```
1  Procedure main()
2      pred_p ← pred(p);                          {p's estimation of its nearest correct predecessor in the ring}
3      succ_p ← succ(p);                          {p's estimation of its nearest correct successor in the ring}
4      foreach q ∈ Π do
5          Δ_p(q) ← default time-out interval;     {Δ_p(q) denotes the duration of p's time-out for q}
6          Balance_p(q) ← 0;

7  || Task 1: repeat periodically                 {Sending heartbeats}
8      if succ_p ≠ p then send (ALIVE, p) to succ_p;

9  || Task 2: repeat periodically                 {Checking time-outs}
10     if ( pred_p ≠ p and p did not receive (ALIVE, pred_p)   ) then   {time-out}
       (    during the last Δ_p(pred_p) ticks of p's clock     )
11         R-broadcast (SUSPICION, p, pred_p);

12 || Task 3: when R-deliver (SUSPICION, q, r)    {Processing SUSPICIONS}
13     Balance_p(r) ← Balance_p(r) + 1;
14     update_pred_and_succ();
15     if r = p then R-broadcast (REFUTATION, p);

16 || Task 4: when R-deliver (REFUTATION, q)      {Processing REFUTATIONS}
17     Balance_p(q) ← Balance_p(q) − 1;
18     Δ_p(q) ← Δ_p(q) + 1;                       {not needed if q = p}
19     update_pred_and_succ()

20 Procedure update_pred_and_succ()
21     if ∀r : Balance_p(r) > 0 then
22         pred_p ← p;
23         succ_p ← p
24     else
25         pred_p ← p's nearest predecessor r in the ring such that Balance_p(r) ≤ 0;
26         succ_p ← p's nearest successor r in the ring such that Balance_p(r) ≤ 0
```



(a) p_1 suspects p_8

(b) p_8 refutes p_1's suspicion

Fig. 2. Sporadic communication in the communication-optimal Algorithm 2. Note that relayed messages caused by Reliable Broadcast are not shown.

Correctness Proof. Now we show that Algorithm 2 implements a failure detector of class $\diamond\mathcal{P}$ and that it is communication-optimal. In the proof, we consider the time after which all the incorrect processes have already crashed, and all the messages they have sent/R-broadcast before crashing have already been received/R-delivered. We consider also that SUSPICION and REFUTATION messages are always broadcast by using R-broadcast and always delivered by using R-deliver.

Observation 1. $\forall p \in \Pi, Balance_p(r) =$ [number of (SUSPICION, $-$, r) messages delivered by p] $-$ [number of (REFUTATION, $-$, r) messages delivered by p]

This derives directly from the fact that $Balance_p(r)$ is initialized to 0 (Line 6) and that it is only incremented or decremented when p respectively delivers a SUSPICION (by Task 3) or a REFUTATION (by Task 4).

Lemma 1. $\forall r \notin correct, \forall p, q \in correct$, *eventually and permanently* $Balance_p(r) = Balance_q(r)$.

Proof. $Balance_s(r)$ is only updated when s delivers either a SUSPICION or a REFUTATION message about r.

Since r is an incorrect process, it will eventually crash. Observe that after that time, r will not be able to broadcast any more REFUTATION messages. Note also that, by the algorithm, a process p suspecting r does not suspect r again unless p delivers a REFUTATION message broadcast by r. As a consequence, the number of subsequent SUSPICION messages about r is limited (at most $n-1$, in case every remaining process is correct).

By the properties of Reliable Broadcast, every message delivered by a correct process is also delivered by the rest of correct processes. Hence, all correct processes deliver the same amount of SUSPICIONS and REFUTATIONS about r. As a result, and by Observation 1, eventually and permanently $Balance_p(r) = Balance_q(r)$. □

Observation 2. $\forall p_i \in correct;\ p_p = pred_{p_i} \Leftrightarrow (\forall p_j \in \{p_{p+1}, \cdots, p_{i-1}\} : Balance_{p_i}(p_j) > 0) \wedge (p = i \vee Balance_{p_i}(p_p) \leq 0)$

This derives directly from the fact that $pred_{p_i}$ is only updated by p_i inside the procedure *update_pred_and_succ*().

Lemma 2. *$\forall r \notin correct, \forall p \in correct$, eventually and permanently $Balance_p(r) > 0$.*

Proof. Let q be the nearest correct successor of r in the ring. By the algorithm and Observation 2, at some time $pred_q = r$ (assuming that the processes in between have already crashed). Since r is incorrect, it will eventually crash and, as a result, by Task 2 q will broadcast a suspicion on r, that r will not be able to refute. Consequently, by Task 3 q will set $Balance_q(r) > 0$ forever. Finally, by Lemma 1, $\forall p \in correct$, eventually and permanently $Balance_p(r) > 0$. Observe that suspicions broadcast on r by other correct processes will result, by Lemma 1, in the increment of $Balance_p(r)$. □

Lemma 3. *$\forall p, q \in correct$, for every (SUSPICION, $-$, q) message delivered by p, p also delivers a (REFUTATION, $-$, q) message.*

Proof. If a correct process p delivers a (SUSPICION, $-$, q) message then all correct processes also deliver such a message due to the fact that suspicions are broadcast by using Reliable Broadcast. As a result, since q is correct, q will deliver that suspicion too (by Task 3) and will consequently broadcast a REFUTATION message, which will be also delivered by all correct processes (by Task 4), since it uses Reliable Broadcast. □

Lemma 4. *$\forall p, q \in correct$, for every (SUSPICION, $-$, q) message delivered by p at time t, there is a time $t' > t$ such that $Balance_p(q) = 0$.*

Proof. Observe that initially $\forall p, q \in correct, Balance_p(q) = 0$. By Lemma 3, for every SUSPICION message delivered by p, p also delivers a REFUTATION message. Since (1) every time a REFUTATION is delivered the waiting time for the timer is increased and (2) the communication channels that connect correct processes are eventually timely, then all messages will eventually be delivered before new suspicion messages are broadcast. As a consequence, by Observation 1, eventually $Balance_p(q) = 0$. □

Lemma 5. *$\forall p \in correct$, eventually and permanently $pred_p$ and $succ_p$ will be set to p's nearest correct predecessor and successor respectively.*

Proof. Let p_i be a correct process (remember that there is at least one in the system) and p_p its nearest correct predecessor. By Lemma 2, $\forall p_j \in \{p_{p+1}, \cdots, p_{i-1}\}$, eventually and permanently $Balance_{p_i}(p_j) > 0$. By Observation 2, eventually $pred_{p_i} = p_p$. In the same way, since by Lemma 2, $\forall p_j \in \{p_{p+1}, \cdots, p_{i-1}\}$, eventually and permanently $Balance_{p_i}(p_j) > 0$, and by Observation 2 eventually p_p will set $succ_{p_p} = p_i$. As a result, eventually each correct process p_i monitors its nearest correct predecessor p_p and each correct process p_p sends heartbeats to its nearest correct successor.

Note that, although time-outs can generate new suspicions, by Lemma 4 every correct process p will eventually set its nearest correct predecessor/successor again. Observe also that each time a SUSPICION is broadcast, its subsequent REFUTATION (by Lemma 3) will allow to increase waiting time $\Delta_p(pred_p)$ (Line 18). Hence, from some time on $\Delta_p(pred_p)$ will be large enough so that the timeout does not expire between the reception of two messages. As a result, there will not be any more suspicions and, thus, $\forall p \in correct$, eventually and permanently $pred_p$ and $succ_p$ will be set to p's nearest correct predecessor and successor respectively. □

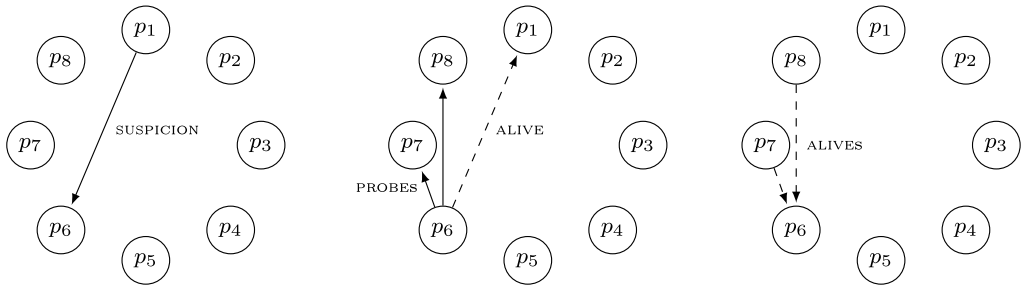
Lemma 6. *$\forall p, q \in correct$, eventually and permanently $Balance_p(q) = 0$.*

Proof. Observe that initially $\forall p, q \in correct, Balance_p(q) = 0$. Also, by Lemma 4, $\forall p, q \in correct$, for every (SUSPICION, $-$, q) message delivered by p at time t, there is a time $t' > t$ such that $Balance_p(q) = 0$.

Now, we show that eventually and permanently there will be no more suspicions. On the one hand, observe that incorrect processes eventually crash, so after that time they will not be able to generate new suspicions. On the other hand, by Lemma 6, $\forall p \in correct$; $pred_p$ and $succ_p$ eventually and permanently stabilize, which implies that they will not issue any more suspicions.

As a result, $\forall p, q \in correct$, eventually and permanently $Balance_p(q) = 0$. □

Theorem 3. *Algorithm 2 implements a failure detector of class $\diamond\mathcal{P}$.*



(a) p_1 suspects p_6

(b) p_6 probes p_7 and p_8, and sends ALIVE to p_1

(c) p_7 and p_8 reply to PROBE with ALIVE

Fig. 3. Sporadic communication in the communication-optimal Algorithm 3.

Proof. From Lemma 6 and Lemma 2, $\forall p, q \in \mathit{correct}$ and $\forall r \notin \mathit{correct}$ eventually and permanently $\mathit{Balance}_p(q) = 0$ and $\mathit{Balance}_p(r) > 0$. The rule "if $\mathit{Balance}_p(q) > 0$, then p suspects q; else, p does not suspect q" provides the properties of strong completeness and eventual strong accuracy of $\diamond\mathcal{P}$. □

Theorem 4. *Algorithm 2 is communication-optimal, i.e., eventually only c links are used forever.*

Proof. From Lemma 6, for every correct process p, eventually and permanently succ_p will be set to p's correct successor in the ring and, by Task 1, p will send (ALIVE, p) messages to it forever. No other periodical messages will be sent. Furthermore, since no more suspicions will occur, no new SUSPICION (and thus REFUTATION) messages will be broadcast. Hence, if there are c correct processes in the system, just a number of c unidirectional links will be permanently used. □

Observe that if there is just one correct process in the system, i.e., $c = 1$, Algorithm 2 will eventually use no links, by an optimization introduced in Task 1. Hence, when $c = 1$ both $\diamond\mathcal{P}$ and Ω can be implemented using 0 links used forever.

Finally, although we have initially assumed in the system model that all the communication links were reliable and eventually timely, in a given execution of Algorithm 2 it is sufficient that this behavior applies only to the c links that eventually form the ring of correct processes, i.e., the link from every correct process to its correct successor in the ring. The rest of links can be asynchronous and/or lossy. This makes c reliable and eventually timely links out of the $n(n-1)$ links in the system.

4.2. An algorithm using one-to-one local communication

In this section, we present a communication-optimal $\diamond\mathcal{P}$ algorithm that uses only one-to-one local communication to manage suspicions. In the algorithm, a process p will be in charge of the detection of incorrect processes between p's correct predecessor in the ring and p. Assuming that simple heartbeat messages circulate around the ring, this strategy only gives *weak* completeness and eventual strong accuracy, and hence the resulting failure detector will be of the class $\diamond\mathcal{Q}$. Henceforth, a further transformation is needed to get a failure detector of the class $\diamond\mathcal{P}$. In this regard, the approach we follow to design the algorithm is incremental: first we present the algorithm implementing $\diamond\mathcal{Q}$ and prove its correctness, and next we give a simple transformation into $\diamond\mathcal{P}$ which preserves communication optimality and low sporadic communication overhead.

4.2.1. Implementing $\diamond\mathcal{Q}$

Algorithm 3 presents a communication-optimal implementation of $\diamond\mathcal{Q}$. Every process p has a local set of suspected processes, L_p, and two variables, pred_p and succ_p, denoting respectively the process that p is monitoring and the process to which p is periodically sending heartbeat messages (ALIVE, p) by Task 1. When a process p suspects by Task 2 the process it is monitoring, pred_p, p includes pred_p in L_p, sends a suspicion message (SUSPICION, p) to pred_p (in Fig. 3a, p_1 suspects p_6), and updates pred_p (and succ_p if required) accordingly. Observe that Task 2 does not include any explicit mechanism for p to tell its new predecessor to start sending heartbeats to it. The new predecessor of p will have to be suspected once by p in order to set its successor to p by Task 3, as we will explain next.

If a suspected process p is correct or has not crashed yet, when it receives (SUSPICION, q) by Task 3, p will suspect every process from p to q (both excluded), since all of them have been also suspected by q. Consequently, process q becomes the new successor of p, and hence, if p does not crash, q will receive periodical (ALIVE, p) messages from p, as shown in Fig. 3b. Furthermore, a sporadic (PROBE, p) message is sent by p to every process r from p to q (both excluded), in order to know if r has actually crashed or not (also shown in Fig. 3b, in which p_6 probes p_7 and p_8). When a process p receives a (PROBE, q) message, it just sends an (ALIVE, p) message to q, in order to give q the opportunity to set succ_q (and exceptionally pred_q) to p (in Fig. 3c, p_7 and p_8 notify p_6 that they are alive).

Algorithm 3: Communication-optimal $\diamond\mathcal{Q}$ using local one-to-one communication.

{Every process p executes the following}

```
1  Procedure main()
2      pred_p ← pred(p);                          {p's estimation of its nearest correct predecessor in the ring}
3      succ_p ← succ(p);                          {p's estimation of its nearest correct successor in the ring}
4      foreach q ∈ Π do Δ_p(q) ← default time-out interval;   {Δ_p(q): duration of p's time-out for q}
5      L_p ← ∅;                                   {L_p provides the properties of ◇Q}

6  || Task 1: repeat periodically                 {Sending heartbeats}
7      if succ_p ≠ p then send (ALIVE, p) to succ_p;

8  || Task 2: repeat periodically                 {Checking time-outs}
9      if ( pred_p ≠ p and p did not receive (ALIVE, pred_p)
            during the last Δ_p(pred_p) ticks of p's clock ) then   {time-out}
10         L_p ← L_p ∪ {pred_p};                  {p suspects pred_p has crashed}
11         send (SUSPICION, p) to pred_p;
12         update_pred_and_succ();

13 || Task 3: when receive (SUSPICION, q) from some q   {Processing SUSPICIONS}
14     L_p ← L_p ∪ {p, ..., q} − {p, q};
15     update_pred_and_succ();
16     foreach r ∈ {p, ..., q} − {p, q} do send (PROBE, p) to r;
17     send (ALIVE, p) to q;

18 || Task 4: when receive (ALIVE, q) from some q   {Processing ALIVES}
19     if q ∈ L_p then
20         L_p ← L_p − {q};
21         update_pred_and_succ();
22         Δ_p(q) ← Δ_p(q) + 1;

23 || Task 5: when receive (PROBE, q) from some q   {Processing PROBES}
24     send (ALIVE, p) to q;

25 Procedure update_pred_and_succ()
26     pred_p ← p's nearest predecessor r in the ring such that r ∉ L_p;
27     succ_p ← p's nearest successor r in the ring such that r ∉ L_p;
28     if pred_p ≠ p then L_p ← {pred_p, ..., succ_p} − {pred_p, p, succ_p}
```

On the reception of an (ALIVE, q) message coming from a process $q \in L_p$, by Task 4 a process p will remove q from L_p, update $pred_p$ (and $succ_p$ if required), and increment the time-out interval with respect to q, $\Delta_p(q)$.

The PROBE messages used in this algorithm avoid a process to send periodically ALIVE messages to crashed processes, a scenario that can happen in communication-efficient algorithms (see Fig. 1, where processes p_7 and p_8 have crashed). Hence, the proposed probing mechanism is key to get communication optimality.

Correctness Proof. We show now that Algorithm 3 implements a failure detector of class $\diamond\mathcal{Q}$ and that it is communication-optimal. Given any process p, we denote by $corr\_pred_p$ the correct predecessor of p in the ring. Similarly, we denote by $corr\_succ_p$ the correct successor of p in the ring. The key of the proof is to show that eventually and permanently $pred_p = corr\_pred_p$ and $succ_p = corr\_succ_p$ for every correct process p. In other words, the ring stabilizes in terms of both the *pred* and *succ* variables of processes, which guarantees the correct construction of the sets L_p of suspected processes.

We assume that every task is executed as a critical section. We start by making the following observations:

Observation 3. $p \notin L_p$ permanently for every process p.

Observation 4. $L_p = \{pred_p, \ldots, succ_p\} - \{pred_p, p, succ_p\}$ permanently after the execution of any task for every process p.

Observe that, whenever L_p is modified by Task 2, Task 3 or Task 4 of p, $pred_p$ and $succ_p$ (and L_p itself) are updated accordingly by the procedure *update_pred_and_succ*().

Observation 5. Whenever a process q is included by a correct process p in L_p, p will send a message (of type SUSPICION in Task 2 or type PROBE in Task 3) to q, and, if q is correct, p will eventually receive an (ALIVE, q) message sent by Task 3 or Task 5 of q.

For the rest of the proof we will assume that any time instant t considered is larger than a time t_{base} that occurs after the stabilization time GST (i.e., $t_{base} > GST$), after every incorrect process has crashed, and after all messages sent by incorrect processes have been received. Note that this eventually happens. Hence, any new message received has necessarily been sent by a correct process.

Lemma 7. *For every correct process p, eventually and permanently $pred_p = corr\_pred_p$.*

Proof. For a correct process p, let $pred_p = r$ such that $r \neq corr\_pred_p$. For a given time in the execution of the algorithm, one of the following cases applies:

Case 1: assume first that $r \in \{corr\_pred_p, \ldots, p\} - \{corr\_pred_p, p\}$. Observe that, with this assumption, r is by definition not correct, as well as any other process in that range. Therefore, r will be eventually included in L_p (by Task 2 or Task 3 of p), and, since r has already crashed, it will not be able to send an (ALIVE, r) message to p, remaining in L_p forever.

Case 2: assume now that $r \notin \{corr\_pred_p, \ldots, pred(p)\}$. By Observation 4, $corr\_pred_p \in L_p$. Observe that $corr\_pred_p$ could have been included in L_p by Task 2 or by Task 3 of p. Since $corr\_pred_p$ is by definition correct, by Observation 5 eventually p will receive and (ALIVE, $corr\_pred_p$) message. At this point, $pred_p$ will be set to $corr\_pred_p$.

Since every time $corr\_pred_p$ has been suspected by p, $\Delta_p(corr\_pred_p)$ is incremented by Task 4 of p, and since the communication link between $corr\_pred_p$ and p is eventually timely, eventually $\Delta_p(corr\_pred_p)$ will reach the unknown bound on message transmission times, after which p will receive the periodical (ALIVE, $corr\_pred_p$) messages always before timer $\Delta_p(corr\_pred_p)$ expires and $corr\_pred_p$ will not be suspected by p any more. □

Lemma 8. *For every correct process p, eventually and permanently $succ_p = corr\_succ_p$.*

Proof. The proof is by contradiction. By Lemma 7, eventually and permanently $pred_{corr\_succ_p} = p$. Assume that $succ_p \neq corr\_succ_p$. Since p is not sending by Task 1 periodical (ALIVE, p) messages to $corr\_succ_p$, then by Task 2 $corr\_succ_p$ will eventually suspect p and modify $pred_{corr\_succ_p}$, which contradicts Lemma 7. □

Theorem 5. *Algorithm 3 implements a failure detector of class $\diamond\mathcal{Q}$.*

Proof. From Lemmas 7 and 8, for every correct process p, eventually p will be in the stable ring formed by correct processes. Otherwise, p is incorrect, and by Lemma 7 eventually and permanently $pred_{corr\_succ_p} = corr\_pred_p$. By Observation 4, p will eventually and permanently be included in $L_{corr\_succ_p}$. As a consequence, eventually and permanently p will be included in the set of suspected processes of some correct process. This provides the weak completeness property of $\diamond\mathcal{Q}$.

By Observation 3, p is never included in L_p. Once the ring has stabilized, no correct process is included (in Task 2) in the set L_p of any correct process p. Hence, once the ring has stabilized, no correct process will be present in any set of suspected processes. This provides the eventual strong accuracy property of $\diamond\mathcal{Q}$. □

Theorem 6. *Algorithm 3 is communication-optimal, i.e., eventually only c links are used forever.*

Proof. From Lemma 8, for every correct process p, eventually and permanently $succ_p$ will be set to p's correct successor in the ring and, by Task 1, p will send (ALIVE, p) messages to it forever. No other periodical messages will be sent. Furthermore, since no more suspicions will occur, no new SUSPICION (and hence sporadic PROBE or ALIVE) messages will be sent. Thus, if there are c correct processes in the system, just a number of c unidirectional links will be permanently used. □

A similar reasoning as the one made for the previous algorithm can be made for Algorithm 3 regarding the minimal assumptions on communication reliability and synchrony required by the algorithm. In this case, in a given execution of Algorithm 3, besides the reliable and eventually timely link from every correct process to its correct successor in the ring, the link from every correct process to its correct predecessor in the ring must be reliable (but not eventually timely). The rest of links can be asynchronous and/or lossy. This makes $2c$ reliable links, half of which must also be eventually timely (i.e., the links that are used forever), out of the $n(n-1)$ links in the system.

4.2.2. Transforming $\diamond\mathcal{Q}$ into $\diamond\mathcal{P}$

In this section we present a transformation of the $\diamond\mathcal{Q}$ algorithm of the previous section into $\diamond\mathcal{P}$. The transformation preserves the communication optimality and low sporadic communication overhead of Algorithm 3.

Algorithm 4, which implements $\diamond\mathcal{P}$, is obtained from Algorithm 3 by adding a set G_p to every process p. G_p is included into the ALIVE messages sent by p. In the algorithm, additions and removals of processes to L_p are also applied to G_p. To build G_p from the set G_{pred_p} received in Task 4, process p adds the processes between $pred_p$ and p to G_{pred_p} (both $pred_p$ and p excluded). In other words, p relies on its predecessor for suspicions beyond its directly monitored neighborhood.

Correctness Proof. We show now that Algorithm 4 implements a failure detector of class $\diamond\mathcal{P}$ and that it is also communication-optimal. The key of the proof is to show that Algorithm 4 is a transformation of Algorithm 3 into $\diamond\mathcal{P}$

Algorithm 4: Communication-optimal $\diamond\mathcal{P}$ using local one-to-one communication.

```
{Every process p executes the following}
1  Procedure main()
2      pred_p ← pred(p);                                   {p's estimation of its nearest correct predecessor in the ring}
3      succ_p ← succ(p);                                   {p's estimation of its nearest correct successor in the ring}
4      foreach q ∈ Π do Δ_p(q) ← default time-out interval; {Δ_p(q): duration of p's time-out for q}
5      L_p ← ∅ ; G_p ← ∅;                                  {L_p provides the properties of ◇Q; G_p provides the properties of ◇P}

6  || Task 1: repeat periodically                          {Sending heartbeats}
7      if succ_p ≠ p then send (ALIVE, p, G_p) to succ_p;

8  || Task 2: repeat periodically                          {Checking time-outs}
9      if ( pred_p ≠ p and p did not receive (ALIVE, pred_p, −)
           during the last Δ_p(pred_p) ticks of p's clock ) then   {time-out}
10         L_p ← L_p ∪ {pred_p}; G_p ← G_p ∪ {pred_p};      {p suspects pred_p has crashed}
11         send (SUSPICION, p) to pred_p;
12         update_pred_and_succ();

13 || Task 3: when receive (SUSPICION, q) from some q      {Processing SUSPICIONS}
14     L_p ← L_p ∪ {p, ..., q} − {p, q}; G_p ← G_p ∪ {p, ..., q} − {p, q};
15     update_pred_and_succ();
16     foreach r ∈ {p, ..., q} − {p, q} do send (PROBE, p) to r;
17     send (ALIVE, p, G_p) to q;

18 || Task 4: when receive (ALIVE, q, G_q) from some q     {Processing ALIVES}
19     if q ∈ L_p then
20         L_p ← L_p − {q};
21         update_pred_and_succ();
22         Δ_p(q) ← Δ_p(q) + 1;
23     if q = pred_p then G_p ← (G_q ∪ {pred_p, ..., p}) − {pred_p, p};

24 || Task 5: when receive (PROBE, q) from some q         {Processing PROBES}
25     send (ALIVE, p, G_p) to q;

26 Procedure update_pred_and_succ()
27     pred_p ← p's nearest predecessor r in the ring such that r ∉ L_p;
28     succ_p ← p's nearest successor r in the ring such that r ∉ L_p;
29     if pred_p ≠ p then L_p ← {pred_p, ..., succ_p} − {pred_p, p, succ_p}
```

preserving communication optimality. Observe that in Task 2 and Task 3 exactly the same operations are applied to L_p and to G_p. Finally, in Task 4, the global set of suspected processes G_p is built by adding to G_{pred_p} the processes between $pred_p$ and p (both $pred_p$ and p excluded).

Observation 6. $p \notin G_p$ permanently for every process p.

Theorem 7. *Algorithm 4 implements a failure detector of class $\diamond\mathcal{P}$.*

Proof. Observe that, by the construction of the set G_p in Tasks 2, 3, and 4, G_p is equivalent to L_p in the range $\{pred_p, \ldots, p\}$. By Lemmas 7 and 8, and by Task 4 of p, eventually G_p will include every incorrect process, providing the strong completeness property of $\diamond\mathcal{P}$. By Observation 6 and again by Lemmas 7 and 8 and Task 4 of p, no correct process will be eventually included in G_p, preserving the eventual strong accuracy property of $\diamond\mathcal{P}$. □

Theorem 8. *Algorithm 4 is communication-optimal, i.e., eventually only c links are used forever.*

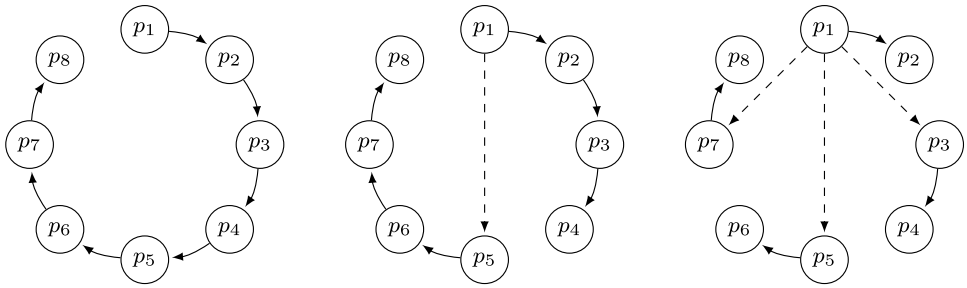
Proof. Follows directly from the fact that no additional message is sent in Algorithm 4 with respect to Algorithm 3. □

5. Performance analysis

In Subsection 1.2 we have defined a set of properties to be taken into account when designing failure detectors, namely communication efficiency, low sporadic overhead and communication locality. In the light of these properties, in this section we analyse the performance of the communication-optimal $\diamond\mathcal{P}$ algorithms. We also include in the analysis Chandra–Toueg's algorithm [1] as a reference, as well as the simplest of the communication-efficient algorithms in [17], which considers the same system model as the present contribution. The goal is to highlight the strengths and weakness of each approach and

Table 2
Performance analysis of $\diamond\mathcal{P}$ algorithms. Sporadic overhead and detection latency figures are approximate (ℓ denotes the cardinality of the local set of suspected processes, usually $\ell \ll n$).

Algorithm	Comm. efficiency (# links used forever)	Sporadic overhead (# extra messages)	Detection latency	Comm. locality	
				Periodic	Sporadic
Chandra–Toueg [1]	$c(n-1)$	0	T_h	No	–
Comm.-efficient [17]	n	0	cT_h	Yes	–
Comm.-optimal Algorithm 2	c	$2n^2$	T_h	Yes	No
Comm.-optimal Algorithm 4	c	2ℓ	cT_h	Yes	Yes



(a) Without shortcuts (b) Adding one shortcut (c) Adding three shortcuts

Fig. 4. Example of the shortcut mechanism to improve responsiveness.

set the trade-offs to be considered in the implementation of failure detectors for particular scenarios. Specifically, we focus on the following performance parameters:

- Communication efficiency/optimality. Number of unidirectional links that are used forever.
- Sporadic communication overhead. Number of extra messages exchanged to manage an erroneous suspicion.
- Detection latency (or system responsiveness). Upon a crash, time elapsed until every alive process permanently suspects the crashed process.
- Communication locality. Scope of the messages. An algorithm exhibits (periodic or sporadic) communication locality when processes send (periodic or sporadic) messages mainly to their neighborhood.

Table 2 summarizes the performance of the algorithms. Note that sporadic communication locality is not applicable to the algorithms in [1,17], since they do not have sporadic communication at all (sporadic overhead is null). Observe also that Algorithm 4, besides being communication-optimal, has a low sporadic overhead due to its one-to-one communication pattern, and has both periodic and sporadic communication locality. This is at the cost of a linear detection latency, in contrast to the uniform detection latency of the algorithms using a one-to-all communication pattern.

In Table 2, detection latency is estimated on the basis of the term T_h, which denotes the mean delay for sending a new, periodic, heartbeat message, by which the information about suspected processes is relayed to the successor process in the ring in the algorithm in [17] and in Algorithm 4. Assuming that the transmission delay of any message is negligible with respect to T_h, the value of T_h can be estimated as one half of the periodicity of the task that sends periodic heartbeat messages (Task 1 in the algorithms proposed in this paper).

The linear detection latency in Algorithm 4 is due to the fact that the information about a new suspicion has to circulate around the whole ring to reach every alive process. Nevertheless, a mechanism to speed-up the detection latency, at the price of losing sporadic communication locality, can be easily added. The mechanism consists in the suspecting process p directly sending the suspicion message not only to the suspected process but also to a subset of processes $\Lambda \subset \Pi$, i.e., adding shortcuts in the ring. In general, if $k = |\Lambda|$, while the communication overhead is incremented in at most $2k$ messages, assuming that shortcuts have been uniformly distributed, an estimation of the maximum detection latency results in $(\frac{n}{k+1})T_h$. In the limit, $\Lambda = \Pi - \{p\}$, and the detection latency is reduced to the minimum. The mechanism can be also applied to the communication-efficient algorithm in [17].

Fig. 4 shows an example of the shortcut mechanism to improve responsiveness in Algorithm 4. Without shortcuts, a failure suspicion propagates by heartbeats and reaches all alive processes in approximately $c\ T_h$ time. One or more shortcuts allow the suspicion to be propagated in parallel, reducing the detection latency. Of course, in case the suspicion is erroneous, such a mechanism makes the error to be propagated faster in the system. A way of avoiding this apparent drawback consists in adding a complementary shortcut mechanism for the refutation of a recent, erroneous, suspicion. Observe that neither of these suspicion/refutation shortcut mechanisms compromises the optimality of Algorithm 4, since eventually, i.e., when the ring formed by correct processes stabilizes, they are not used any more.

In general, a trade-off results between detection latency and communication overhead, being the latter either periodic or sporadic. It is worth noting that parameters related to communication overhead have been expressed in terms of the number of links and messages. Communication costs, however, strongly depend on network parameters as topology and link delays, specifically in wide area networks, resulting in not uniform message transmission delays. This fact can be exploited when using a ring arrangement as a pattern for communication. More precisely, communication costs associated to pairs of processes provide a hint for the initial ring configuration. Periodic heartbeat communication will take advantage of this. Furthermore, sporadic communication overhead, which is inherent to communication-optimal algorithms, results in practice less harmful for algorithms with this kind of local communication than for algorithms with global communication. As we have shown, the ability of using shortcuts in the formers provide an interesting basis to fit trade-offs between sporadic communication overhead and detection latency.

Finally, observe that in Algorithm 4 we have ignored message size for simplicity, assuming that it is not a big issue. With n processes we need $\log_2 n$ bits per process identifier. Hence, the size of the lists of processes is bounded by $n \log_2 n$ bits. Alternatively, we could use a Boolean bit per process, sending always n bits (e.g., 0 not suspected, 1 suspected). Depending on the average number of suspicions, we could choose in practice—even changing dynamically—the smallest value of the two.

6. Conclusions

In this paper, we have explored communication efficiency, a performance measure that refers to the number of unidirectional links that are used forever in an algorithm. We have shown that failure detector class $\diamond\mathcal{P}$ requires at least c unidirectional links to be used forever, c being the number of correct processes. Moreover, when at least one process crashes, i.e., $c < n$, c links are also required for $\diamond\mathcal{S}$ and Ω. We have proposed two ring-based communication-optimal $\diamond\mathcal{P}$ algorithms. Since these algorithms use exactly c unidirectional links forever, it can be derived that communication optimality for $\diamond\mathcal{P}$ is achieved. Since $\diamond\mathcal{P}$ trivially implements Ω, communication optimality can be considered achieved also for $\diamond\mathcal{S}$ and Ω failure detectors when $c < n$.

One of the algorithms uses a Reliable Broadcast primitive to communicate suspicions and refutations, involving a quadratic number of messages. Since this can be a drawback in some scenarios, e.g., very large networks, we have proposed a second algorithm that uses exclusively one-to-one communication. This algorithm has some interesting properties which make it suitable for distributed applications deployed over wide area networks: (1) communication optimality, i.e., only c links are used forever, (2) low sporadic communication overhead to manage failure suspicions, i.e., the number of messages exchanged as a consequence of a suspicion is linear, and (3) communication locality, i.e., managing a failure suspicion at a process p only implies communicating with processes at p's neighborhood in the ring. This is of particular interest in networks where communication costs between pairs of processes are not uniform. If the logical ring of processes is arranged regarding communication cost criteria, both communication overhead and delays of periodic heartbeats will be minimized.

The second algorithm admits a flexible implementation. In its basic form, i.e., using exclusively local communication, it exhibits a linear detection latency. However, a mechanism based on shortcuts can reduce it. Moreover, this mechanism can be enabled or disabled without affecting the correctness of the algorithm. As a consequence, the mechanism can be applied partially and be precisely tuned in order to fit trade-offs between network overhead and QoS parameters.

References

[1] T. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, J. ACM 43 (2) (1996) 225–267.
[2] M. Fischer, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (2) (1985) 374–382.
[3] T. Chandra, V. Hadzilacos, S. Toueg, The weakest failure detector for solving consensus, J. ACM 43 (4) (1996) 685–722.
[4] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg, On implementing omega in systems with weak reliability and synchrony assumptions, Distrib. Comput. 21 (4) (2008) 285–314.
[5] F. Chu, Reducing Ω to $\diamond\mathcal{W}$, Inf. Process. Lett. 67 (6) (1998) 289–293.
[6] A. Fernández, E. Jiménez, M. Raynal, Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony, in: Proceedings of the IEEE International Conference on Dependable Systems and Networks, DSN'2006, Philadelphia, Pennsylvania, 2006, pp. 166–178.
[7] M. Hutle, D. Malkhi, U. Schmid, L. Zhou, Chasing the weakest system model for implementing Ω and consensus, IEEE Trans. Dependable Secure Comput. 6 (4) (2009) 269–281.
[8] E. Jiménez, S. Arévalo, A. Fernández, Implementing unreliable failure detectors with unknown membership, Inf. Process. Lett. 100 (2) (2006) 60–63.
[9] M. Larrea, A. Fernández, 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, Nurenberg, Germany, 2000, pp. 52–59.
[10] D. Malkhi, F. Oprea, L. Zhou, Omega meets Paxos: leader election and stability without eventual timely links, in: Proceedings of the 19th International Symposium on Distributed Computing, DISC'2005, Krakow, Poland, in: Lect. Notes Comput. Sci., vol. 3724, Springer-Verlag, 2005, pp. 199–213.
[11] A. Mostéfaoui, E. Mourgaya, M. Raynal, C. Travers, A time-free assumption to implement eventual leadership, Parallel Process. Lett. 16 (2) (2006) 189–208.
[12] 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, in: Lect. Notes Comput. Sci., vol. 2180, Springer-Verlag, 2001, pp. 108–122.
[13] M. Larrea, A. Fernández, S. Arévalo, Eventually consistent failure detectors, J. Parallel Distrib. Comput. 65 (3) (2005) 361–373.
[14] M. Larrea, S. Arévalo, A. Fernández, Efficient algorithms to implement unreliable failure detectors in partially synchronous systems, in: Proceedings of the 13th International Symposium on Distributed Computing, DISC'99, Bratislava, in: Lect. Notes Comput. Sci., vol. 1693, Springer-Verlag, 1999, pp. 34–48.

[15] M. Larrea, A. Fernández, S. Arévalo, On the implementation of unreliable failure detectors in partially synchronous systems, IEEE Trans. Comput. 53 (7) (2004) 815–828.
[16] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323.
[17] M. Larrea, A. Lafuente, I. Soraluze, R. Cortiñas, J. Wieland, Designing efficient algorithms for the eventually perfect failure detector class, J. Softw. 2 (4) (2007) 1–11.
[18] M. Hutle, Failure detection in sparse networks, Ph.D. thesis, Vienna University of Technology, Wien, August 2005.
[19] W. Chen, S. Toueg, M. Aguilera, On the quality of service of failure detectors, IEEE Trans. Comput. 51 (5) (2002) 561–580.
[20] M. Hurfin, M. Raynal, A simple and fast asynchronous consensus protocol based on a weak failure detector, Distrib. Comput. 12 (4) (1999) 209–223.
[21] A. Mostéfaoui, M. Raynal, Solving consensus using Chandra–Toueg's unreliable failure detectors: a general quorum-based approach, in: Proceedings of the 13th International Symposium on Distributed Computing, DISC'99, Bratislava, in: Lect. Notes Comput. Sci., vol. 1693, Springer-Verlag, 1999, pp. 49–63.
[22] A. Mostéfaoui, M. Raynal, Leader-based consensus, Parallel Process. Lett. 11 (1) (2001) 95–107.
[23] A. Schiper, Early consensus in an asynchronous system with a weak failure detector, Distrib. Comput. 10 (3) (1997) 149–157.
[24] F. Greve, M. Hurfin, R. Macedo, M. Raynal, Consensus based on strong failure detectors: a time and message-efficient protocol, in: Proceedings of the 15th Parallel and Distributed Processing Workshop, IPDPS'2000, Cancun, Mexico, in: Lect. Notes Comput. Sci., vol. 1800, Springer-Verlag, 2000, pp. 1258–1265.
[25] R. Guerraoui, M. Raynal, The information structure of indulgent consensus, IEEE Trans. Comput. 53 (4) (2004) 453–466.
[26] L. Lamport, The part-time parliament, ACM Trans. Comput. Syst. 16 (2) (1998) 133–169.
[27] R. Guerraoui, M. Kapalka, P. Kouznetsov, The weakest failure detector to boost obstruction-freedom, in: Proceedings of the 20th International Symposium on Distributed Computing, DISC'2006, Stockholm, Sweden, in: Lect. Notes Comput. Sci., vol. 4167, Springer-Verlag, 2006, pp. 399–412.
[28] W. Wu, J. Cao, J. Yang, 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, Montbeliard-Sochaux, France, IEEE Computer Society, 2006, pp. 64–72.
[29] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg, On implementing Ω with weak reliability and synchrony assumptions, in: Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing, PODC'2003, Boston, Massachusetts, 2003, pp. 306–314.
[30] V. Hadzilacos, S. Toueg, Fault-tolerant broadcasts and related problems, Chapter 5 in: S.J. Mullender (Ed.), Distributed Systems, 2nd edition, Addison-Wesley, 1993, pp. 97–146.
[31] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, 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, St. John's, Newfoundland, Canada, 2004, pp. 328–337.
[32] A. Fernández, E. Jiménez, S. Arévalo, Minimal system conditions to implement unreliable failure detectors, in: Proceedings of the 12th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC'2006, Riverside, USA, University of California, 2006, pp. 63–72.