On the Implementation of Unreliable Failure Detectors in Partially Synchronous Systems

Mikel Larrea, Antonio Fernández, *Senior Member*, *IEEE*, and Sergio Arévalo

Abstract—Unreliable failure detectors were proposed by Chandra and Toueg as mechanisms that provide information about process failures. Chandra and Toueg defined eight classes of failure detectors, depending on how accurate this information is, and presented an algorithm implementing a failure detector of one of these classes in a partially synchronous system. This algorithm is based on all-to-all communication and periodically exchanges a number of messages that is quadratic on the number of processes. In this paper, we study the implementability of different classes of failure detectors in several models of partial synchrony. We first show that no failure detector with perpetual accuracy (namely, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$) can be implemented in these models in systems with even a single failure. We also show that, in these models of partial synchrony, it is necessary a majority of correct processes to implement a failure detector of the class Θ proposed by Aguilera et al. Then, we present a family of distributed algorithms that implement the four classes of unreliable failure detectors with eventual accuracy (namely, $\diamond\mathcal{P}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{W}$). Our algorithms are based on a logical ring arrangement of the processes, which defines the monitoring and failure information propagation pattern. The resulting algorithms periodically exchange at most a linear number of messages.

Index Terms—Consensus problem, crash failures, distributed systems, failure detection, partial synchrony, unreliable failure detectors.

1 Introduction

The Consensus problem [22] is considered one of the fundamental problems in distributed computing. However, it was shown by Fischer et al. [12] that the Consensus problem cannot be solved deterministically in an asynchronous system in which processes can fail. This result, known as the *FLP impossibility*, generated a series of works that tried to identify the amount of synchrony needed to solve Consensus in the presence of failures, and showed how to solve Consensus in these *partially synchronous* systems [9], [10].

An alternative and elegant approach to circumvent the unsolvability of Consensus in asynchronous systems was proposed by Chandra and Toueg [6]. They augmented the asynchronous model of computation with *unreliable failure detectors*. Informally, an unreliable failure detector is a distributed "oracle" that gives (possibly incorrect) hints about which processes of the system have crashed. Based on two basic abstract properties (namely, *completeness* and *accuracy*), Chandra and Toueg proposed eight different classes of unreliable failure detectors and showed that Consensus could be solved in an asynchronous system with any of them.

Chandra-Toueg's model of unreliable failure detectors can be viewed as an abstract way of incorporating partial synchrony assumptions into the model of computation. Instead of focusing on the timing assumptions of a given model of partial synchrony, their model of failure detectors considers abstract properties that must be satisfied in order to solve Consensus. However, the synchrony assumptions are in fact encapsulated in the failure detector. Clearly, systems using these unreliable failure detectors are no longer truly asynchronous; they merely produce the illusion of an asynchronous system by encapsulating all references to time in the failure detector. This leads to the practical problem of *implementing* a given failure detector in a specific model of synchrony.

From the FLP impossibility result [12] and the possibility of solving Consensus using unreliable failure detectors [6], one can establish the impossibility of implementing any of Chandra-Toueg's classes of failure detectors in a purely asynchronous system. (Such an implementation could be used to solve Consensus in an asynchronous system, contradicting the FLP impossibility result.) On the other hand, in a fully synchronous system even a *Perfect* failure detector (i.e., one that does not make mistakes) can be implemented. In such a system, one can build a simple timeout-based algorithm that reliably detects the failure of processes.

1.1 Partial Synchrony

Distributed algorithms can be designed under different assumptions of system behaviors, i.e., system models. One of the main assumptions in which system models can differ is related to the timing aspects. Most models focus on two timing attributes: the time taken for message delivery across a communication channel and the time taken by a processor to execute a piece of code. Depending on whether these attributes are bounded or not, and on the knowledge of these bounds, they can be classified as synchronous, asynchronous, or partially synchronous [10]. A timing attribute is *synchronous* if there is a known fixed upper bound on it. On the other hand, it is *asynchronous* if there is

- *M. Larrea is with the Departamento de Arquitectura y Tecnología de Computadores, Universidad del País Vasco, Paseo Manuel de Lardizabal 1, 20018 San Sebastián, Spain. E-mail: mikel.larrea@si.ehu.es.*
- *A. Fernández and S. Arévalo are with the Escuela de Ciencias Experimentales y Tecnología, Universidad Rey Juan Carlos, c/Tulipán s/n, 28933 Móstoles, Spain.*
  E-mail: afernandez@acm.org, s.arevalo@escet.urjc.es.

Manuscript received 5 Dec. 2002; revised 7 Mar. 2003; accepted 22 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117887.

Completeness	**Accuracy**			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\diamond\mathcal{P}$	*Eventually Strong* $\diamond\mathcal{S}$
Weak	*Quasi-Perfect* $\mathcal{Q}$	*Weak* $\mathcal{W}$	*Eventually Quasi-Perfect* $\diamond\mathcal{Q}$	*Eventually Weak* $\diamond\mathcal{W}$

Fig. 1. Eight classes of failure detectors defined in terms of completeness and accuracy.

no bound on it. Finally, a timing attribute is *partially synchronous* if it is neither synchronous nor asynchronous. Dwork et al. [10] consider two kinds of partial synchrony. In the first one, the timing attributes are bounded, but the bounds are unknown. In the second one, the timing attributes are bounded and the bounds are known, but they hold only after an unknown stabilization interval. They showed that Consensus can be solved in both models and that a majority of correct processes is required. Chandra and Toueg [6] propose another kind of partial synchrony, in which the timing attributes are bounded, but the bounds are unknown and hold only after an unknown stabilization interval. They showed how to implement a failure detector strong enough to solve Consensus in this model.

Although the asynchronous model (in which at least one of the timing attributes is asynchronous) is attractive for designing distributed algorithms, it is well known that a number of distributed problems cannot be solved deterministically in asynchronous systems in which processes can fail. For instance, as we said above, Consensus cannot be solved deterministically in an asynchronous system that is subject to even a single process failure [12], while it can be solved in both synchronous and partially synchronous systems [6], [9], [10]. In fact, the ability to solve these synchronization distributed problems closely depends on the ability to detect failures. In a synchronous system, reliable failure detection is possible. One can reliably detect failures using timeouts. (The timeouts can be derived from the known upper bounds on message delivery time and processing time.) On the other hand, in an asynchronous system it is impossible to distinguish a failed process from a very slow one. Thus, reliable failure detection is impossible.

However, even if it is sufficient, reliable failure detection is not necessary to solve most of these problems. As we already mentioned, Chandra and Toueg [6] introduced unreliable failure detectors (i.e., failure detectors that can make mistakes), and showed how they can be used to solve Consensus and Atomic Broadcast. Guerraoui et al. [13] showed how unreliable failure detectors can be used to solve the Nonblocking Atomic Commitment problem.

1.2 Unreliable Failure Detectors

An unreliable failure detector is an oracle that gives hints about crashed processes. In a system with a failure detector, each process has access to a local *failure detector module*, which monitors other processes in the system and maintains a set of those that it currently suspects to have crashed. A failure detector module can make mistakes by not suspecting a crashed process or by erroneously adding processes to its set of suspects, i.e., it can suspect that a process p has crashed even though p is still running. If it later finds that suspecting p was a mistake, it can remove p from its set of suspects. Thus, each module may repeatedly add and remove processes from its set of suspected processes. Furthermore, at any given time the failure detector modules at two different processes may have different sets of suspects.

Chandra and Toueg characterized a *class* of failure detectors by specifying the *completeness* and *accuracy* properties that failure detectors in that class must satisfy. Roughly speaking, the completeness property requires that every process that actually crashes is eventually suspected, while the accuracy property restricts the mistakes (i.e., false suspicions) that a failure detector can make. Chandra and Toueg defined two completeness and four accuracy properties in [6], which combined gave rise to eight classes of failure detectors. Regarding completeness, they proposed the following two properties:

- *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.

And regarding accuracy, the following four properties:

- *(Perpetual) Strong Accuracy.* No process is suspected before it crashes.
- *(Perpetual) Weak Accuracy.* Some correct process is never suspected.
- *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.

Failure detectors with eventual accuracy may suspect *every* process at one time or another, while failure detectors with perpetual accuracy require that at least one correct process is never suspected.

Note that, in isolation, completeness and accuracy are useless. For example, strong completeness can be satisfied by forcing every process to permanently suspect every other process in the system. Similarly, strong accuracy can be satisfied by forcing every process to never suspect any process in the system. Such failure detectors are clearly useless since they provide no information about failures. To be useful, a failure detector must have some completeness and some accuracy.

Combining one of the two completeness properties with one of the four accuracy properties we obtain one *class* of failure detectors. There are eight different classes, which are presented in Fig. 1. In this paper, we denote the four classes with perpetual accuracy as *perpetual* and the four classes with eventual accuracy as *eventual*. As we said, Chandra and Toueg showed in [6] that any of the failure detectors of Fig. 1 can be used to solve Consensus.

Chandra and Toueg [6] also proposed a timeout-based implementation of a $\diamond\mathcal{P}$ failure detector in a system with partial synchrony. Since any failure detector of class $\diamond\mathcal{P}$ also belongs to classes $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{W}$, they in fact showed the implementability of the four eventual classes of failure detectors. In their algorithm, each process periodically sends a message to the rest of processes in order to inform them that it has not crashed. If there are n processes in the system and $\mathcal{C}_r$ of them do not fail in a run r, at least $n\mathcal{C}_r$ messages are periodically exchanged with this algorithm. Concerning the perpetual classes of failure detectors, i.e., $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$, they were neither shown to be implementable nor shown to be impossible to implement in models of partial synchrony.

Another class of failure detectors was proposed by Aguilera et al. in [2]. They called this class Θ and showed that the failure detectors in this class are the weakest required to solve *uniform reliable broadcast*. We can define the class Θ in terms of completeness and accuracy properties. Let us say that a process p *trusts* another process q at a given time t if p does not suspect q at time t; then, a failure detector belongs to class Θ if it satisfies the following properties:

- Θ-*completeness*. There is a time after which correct processes do not *trust* any process that crashes.[1]
- Θ-*accuracy*. If there is a correct process then, at every time, every process trusts at least one correct process.

Note that a process may be trusted even if it has actually crashed. Moreover, the correct process trusted by a process p is allowed to change over time (in fact, it can change infinitely often), and it is not necessarily the same as the correct process trusted by another process q. Aguilera et al. proposed in [2] an algorithm implementing Θ in an asynchronous system with a majority of correct processes.

1.3 Our Results

In this paper, we study the possibility of implementing several classes of failure detectors in partially synchronous systems. We start with the four *perpetual* classes ($\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$) out of the eight classes of failure detector classes proposed in [6]. We show that none of these four classes can be implemented in a partially synchronous system with failures (even with one single failure). The partial synchrony assumptions we make in our system are at least as strong as those made in [6], [10], which means that our results apply to their models of partial synchrony as well.

At first glance, our result may seem evident. Nevertheless, even if the proofs are not very difficult, the result itself is far from being trivial. To understand it, note that Consensus can be solved—without perpetual failure detectors—in the models of partial synchrony considered in this paper, while we show that no one of the perpetual failure detectors defined by Chandra and Toueg can be implemented in these models. This means that, even if they suffice, perpetual failure detectors are not necessary to solve Consensus. Actually, *eventual* failure detectors suffice,[2] which is not strange, knowing that Consensus requires that the unanimous decision has to be reached eventually. From the previous, it can also be derived that the problem of implementing perpetual failure detectors is harder than solving Consensus in the models of partial synchrony considered in this paper.

We complete the impossibility results of this paper showing that it is impossible to implement a failure detector of class Θ in these partially synchronous systems if there is not a majority of correct processes. Since Aguilera et al. [2] also presented an algorithm implementing Θ in an asynchronous system with a majority of correct processes, our result identifies the majority of correct processes as a necessary and sufficient condition for Θ failure detectors to be implemented in partially synchronous systems.

Then, we present algorithms that implement unreliable failure detectors of each of the four *eventual* classes ($\diamond\mathcal{P}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{W}$) in partially synchronous systems. Our algorithms (and, particularly, our $\diamond\mathcal{P}$ algorithm) are more efficient in terms of messages periodically exchanged than the $\diamond\mathcal{P}$ algorithm presented in [6]. Moreover, they strictly implement failure detectors of the corresponding class. Then, the $\diamond\mathcal{W}$ failure detector implemented does not belong to $\diamond\mathcal{Q}$ nor $\diamond\mathcal{S}$, and the $\diamond\mathcal{Q}$ and $\diamond\mathcal{S}$ failure detectors implemented do not belong to $\diamond\mathcal{P}$. This shows the possibility of implementing each class without implementing a stronger one.

Our algorithms have been designed and are presented as a sequence of refinements. First, we present an algorithm that provides weak completeness. Next, we show how to extend this algorithm to provide eventual weak accuracy. This extended algorithm implements a $\diamond\mathcal{W}$ failure detector. Next, we present two other extensions which strengthen the accuracy and the completeness, respectively, implementing the stronger failure detectors.

In all these algorithms, each correct process monitors only one other process in a cyclic fashion. The monitoring process performs this task by repeatedly polling the monitored process. Each polling involves only two messages exchanged between the monitoring and monitored processes. If the pollings were done periodically, a total of no more than $2n$ messages would be periodically exchanged. Eventually, in a run r, this amount becomes at most $2\mathcal{C}_r$ (being $\mathcal{C}_r$ the number of processes that do not fail in r), which is a significant improvement over the at least $n\mathcal{C}_r$ messages of Chandra and Toueg's algorithm.

The rest of the paper is organized as follows: In Section 2, we present the system model we will consider in the paper. In Section 3, we present the impossibility results. In Section 4, we present the algorithms that implement the eventual classes of failure detectors. Finally, Section 5, concludes the paper.

1. Θ-completeness is the same as strong completeness since a trust is just the complement of a suspicion.

2. This follows directly from the fact that $\diamond\mathcal{W}$ is the weakest failure detector for solving Consensus [5], and perpetual failure detectors are stronger than $\diamond\mathcal{W}$.

1.4 Related Work

Several works have considered the implementation of some of Chandra-Toueg's classes of failure detectors in partially synchronous systems [1], [4], [11], [15], [17], [18], [23]. Basically, the protocols proposed in all those works obey the same principle as the ones presented in this paper: Using successive approximations, each process dynamically determines a value Δ that eventually becomes an upper bound on message transfer delays.

Unreliable failure detectors with limited completeness and/or accuracy are studied in [3], [14], [20], [21], [24], [25]. *Realistic* failure detectors have been introduced and investigated in [8]. The notion of *Quality of Service* of failure detectors has been introduced and studied in [7] (namely, how *fast* a failure detector detects failures and how *well* it avoids false detection).

Recently, [19] proposes an asynchronous implementation of failure detectors based on a query-response mechanism, which assumes that the query/response messages exchanged obey a precise pattern regarding the order in which the responses from some processes to a query arrive.

2 System Model

2.1 Processes and Failures

We consider a distributed system consisting of a finite set Π of n processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$, that communicate only by sending and receiving messages. Every pair of processes is assumed to be connected by a reliable communication channel.

Processes can fail by *crashing*, that is, by prematurely halting. Crashes are permanent, i.e., crashed processes do not recover. In every run r of the system, we identify two complementary subsets of Π: The subset of processes that do not fail, denoted $correct_r$, and the subset of processes that do fail, denoted $crashed_r$. We use f to denote a known upper bound on the number of crashed processes in the system in any run, which we assume is always less than n, i.e., $\forall r : |crashed_r| \leq f < n$. We also assume that failures are symmetric, i.e., a priori *any* process in the system can crash. For a given run r, for every process p in $crashed_r$ we use $Tcrash_p^r$ to denote the instant at which p crashes in r. Finally, when needed we use $\mathcal{C}_r$ to denote the number of correct processes in run r, which we assume is at least one, i.e., $\forall r : \mathcal{C}_r = |correct_r| > 0$.

2.2 Partial Synchrony

In order to define the level of synchrony of a system, we use two parameters, the transmission delay of messages and the relative speeds of processes. In the *asynchronous model*, there are no upper bounds on one or both of these parameters. Thus, to say that a system is asynchronous is to make no timing assumptions. In the *synchronous model*, there are known upper bounds, which we denote by Δ and Φ, respectively, on the transmission delay of messages and the relative speeds of processes. Between the synchronous and the asynchronous models, there is a whole range of possible models of synchrony. We call these *partially synchronous models.* Dwork et al. [10] consider the following two models of partial synchrony:

- M_1: In every run of the system, there are upper bounds Δ and Φ on the transmission delay of messages and the relative speeds of processes, respectively, but these bounds are not known.
- M_2: bounds exist and are known, but they hold only after some unknown (but finite) time GST (for *Global Stabilization Time*).

A system that conforms to model M_2 can be seen as asynchronous up to GST and as synchronous after GST. Thus, M_2 can be seen as an *eventually synchronous* model. However, it is important to note that the actual value of GST is not known and can vary from run to run.

In the original model M_2 of Dwork et al., messages sent before GST can get lost. To be consistent with the assumption of reliable channels, we will assume here that no message is lost. Since we only use model M_2 to show impossibility results, this later assumption does not restrict our results in any way. Any impossibility result shown with this model also applies to the model with messages losses.

In [6], Chandra and Toueg proposed a weaker model of partial synchrony M_3, which somehow generalizes the two previous models M_1 and M_2:

- M_3: bounds Δ and Φ exist, but they are not known *and* they hold only after some unknown (but finite) time GST.

A system that conforms to model M_3 can be seen as asynchronous up to GST, and as a system conforming to model M_1 after GST. Note that every system that conforms to models M_1 or M_2 also conforms to model M_3.

In their initial definition, Chandra and Toueg considered a model M_3 in which no message can get lost (even before GST). They later considered the possibility of message losses before GST. For simplicity, in this work, we consider a model M_3 without message losses. It is not hard to modify the algorithms presented to deal with losses, but that would imply an increase on the number of messages exchanged.

2.3 Implementation of Failure Detectors

A *distributed failure detector* can be viewed as a set of n failure detection modules, each one attached to a different process in the system. These modules cooperate to satisfy the required properties of the failure detector. Each module maintains a list of the processes it suspects to have crashed. These lists can differ from one module to another at a given time.

In this paper, we only describe the behavior of the failure detection modules in order to implement a failure detector, but not the behavior of the processes they are attached to. For this reason, in the rest of the paper, we will mostly use the term *process* instead of *failure detection module*. It will be clear from the context if we are referring to the failure detection module or the process attached to it. We consider that a process cannot crash independently of its attached failure detection module.

3 Impossibility Results of Implementing Failure Detectors

In this section, we show various impossibility results, the main one being the impossibility of implementing perpetual failure detectors. When proving impossibility results, it is convenient to consider the strongest model of partial synchrony because the impossibility applies directly to the weaker ones. Hence, we will consider in our proofs of impossibility models M_1 and M_2. Furthermore, when considering model M_1, we will assume that the bound on the relative speeds of processes Φ is known, while only the bound on the transmission delay of messages Δ is unknown.[3] Clearly, this model is stronger than M_1 and M_3, and any negative result will apply to these models as well. We will also assume that communication channels are completely reliable under both models. As we will see, the impossibility proofs are the same for both models, with minor variations, which will be pointed out.

3.1 Any Implementation of a Perpetual Failure Detector in Model M_1 Requires a Majority of Correct Processes

There is a simple proof that any implementation in model M_1 of a perpetual failure detector requires a majority of correct processes. The proof basically shows that any implementation of a failure detector of class $\mathcal{W}$ in the model of partial synchrony M_1 (and, thus, in model M_3) requires $f < n/2$.

The proof, which uses contradiction, is as follows: It is shown in [10] that the smallest number of processes for which a Consensus protocol that tolerates f failures exists in the model of partial synchrony M_1 is $2f + 1$. In other words, any protocol that solves Consensus in model M_1 requires a majority of correct processes, i.e., $f < n/2$.

Let us assume now that we have an algorithm A that implements a failure detector of class $\mathcal{W}$ in model M_1 with $f \geq n/2$. In [6], Chandra and Toueg propose a Consensus protocol based on $\mathcal{W}$[4] that tolerates up to $n-1$ faulty processes in asynchronous systems with n processes. In other words, Chandra-Toueg's protocol does not require a majority of correct processes. Clearly, one could run this protocol on top of A and solve Consensus in model M_1 with $f \geq n/2$, which is a contradiction.

Note that this argument shows that $\mathcal{W}$ cannot be implemented in the model of partial synchrony M_1 without a majority of correct processes, but it says nothing about the possibility of implementing $\mathcal{W}$ with a majority of correct processes, i.e., when $f < n/2$. In the following section, we show the impossibility even when there is only one faulty process. Furthermore, the above proof only applies to the models M_1 and M_3 of partial synchrony, while the results of the following section also apply to model M_2.

3.2 Impossibility of Implementing Perpetual Failure Detectors

In this section, we show that none of the perpetual failure detector classes (i.e., $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$) can be implemented in the models of partial synchrony M_1 and M_2.

3. Actually, the results hold if at least one of the two bounds is unknown. Intuitively, any slowness of the relative speeds of processes can always be attributed to the slowness of the transmission of messages and vice versa.

4. Actually, their protocol is based on $\mathcal{S}$, but they also show that failure detector classes $\mathcal{W}$ and $\mathcal{S}$ are equivalent.

3.2.1 An Outline of the Result

From the relationship between failure detector classes described in [6], it would be sufficient to show the impossibility result for the failure detector class $\mathcal{W}$ since $\mathcal{W}$ is the *weakest* of the four classes of failure detectors satisfying perpetual accuracy. For the sake of the presentation we prefer to start showing the result for the failure detector classes satisfying perpetual *strong* accuracy ($\mathcal{P}$ and $\mathcal{Q}$) and then show it for those satisfying perpetual *weak* accuracy ($\mathcal{S}$ and $\mathcal{W}$). In both cases, the approach followed is assuming the existence of a failure detector satisfying a completeness property, and showing that the perpetual accuracy property is violated.

The principle used to prove the impossibility is to consider different runs of the system—with and without crashes—such that they look identical for some correct processes up to certain time t. Hence, these processes can take the same actions in both kinds of runs up to t, in particular in what concerns the suspicion of other processes. We show that, by doing this, the required perpetual accuracy is violated and, thus, the failure detector does not implement any of the four perpetual failure detector classes defined in [6]. To construct a run without a crash that looks identical up to time t to one with a crash, we assume that the appropriate messages are delayed beyond t. This can happen if the value of the parameter Δ or GST (depending on the synchrony model) is larger than t. This is a valid assumption since the values of these parameters are unknown and can be chosen freely for a given run if required.

We first show the impossibility result for failure detector classes $\mathcal{P}$ and $\mathcal{Q}$. For that, one single incorrect suspicion of a correct process by another correct process is sufficient since this violates the perpetual strong accuracy property. Then, we extend the result to failure detector classes $\mathcal{S}$ and $\mathcal{W}$, by showing a run of the system in which all the correct processes are erroneously suspected at least once, thus violating perpetual weak accuracy.[5]

3.2.2 Impossibility for $\mathcal{P}$ and $\mathcal{Q}$ (Perpetual Strong Accuracy)

In this section, we show the impossibility result for failure detector classes $\mathcal{P}$ and $\mathcal{Q}$. Let Σ be a partially synchronous distributed system that conforms to model M_1 or model M_2, made up of $n > 1$ processes, such that at least one of them is correct, i.e., at most $f < n$ of them may crash.

Theorem 1. *Let FD_Σ be a failure detector, implemented on the system Σ, that satisfies the weak completeness property. Then, FD_Σ cannot satisfy the strong accuracy property.*

Proof. Let us consider a run R of Σ in which some process p crashes at time 0. Since FD_Σ satisfies the weak completeness property, there is a time t after which some correct process q permanently suspects p.

Let us consider now a run R' in which no process crashes, but:

5. Note that an algorithm implementing any given class $\mathcal{D}$ of failure detectors must satisfy the properties that characterize $\mathcal{D}$ in any run.

- All messages sent by p are received after time t. This can happen if we assume that $\Delta > t$, if Σ conforms to M_1, or $GST > t$, if Σ conforms to M_2.
- All processes except p behave exactly like in run R up to time t.

Clearly, process q cannot distinguish run R from run R' up to time t as defined in R. Hence, at time t, q will suspect p in R', as it did in R, and the strong accuracy property is not satisfied. □

Corollary 1. *There is no protocol that implements a failure detector of class $\mathcal{Q}$ (hence, of class $\mathcal{P}$) in a partially synchronous distributed system that conforms to model M_1 or model M_2.*

3.2.3 Impossibility for $\mathcal{S}$ and $\mathcal{W}$ (Perpetual Weak Accuracy)

In this section, we show the impossibility result for failure detector classes $\mathcal{S}$ and $\mathcal{W}$. We first give a more intuitive preliminary result, which assumes runs in which all processes except one crash. Then, we generalize the result for any number of crashes.

Let Σ be a partially synchronous distributed system that conforms to model M_1 or model M_2, made up of $n > 1$ processes, such that at least one of them is correct.

3.2.4 Impossibility for $f = n - 1$

Theorem 2. *Let FD_Σ be a failure detector, implemented on the system Σ, that satisfies the strong completeness property. Then, if $f = n - 1$, FD_Σ cannot satisfy the weak accuracy property.*

Proof. Let us consider n runs R_i, $i = 1, \ldots, n$, of Σ such that, in run R_i all processes except p_i crash at time 0. Since FD_Σ satisfies the strong completeness property, there is some time t_i at which p_i suspects all other processes. Let us define $t = \max_i\{t_i\}$.

Let us consider now a run R in which no process crashes, but:

- All messages sent are received after time t. This can happen if we assume that $\Delta > t$, if Σ conforms to M_1, or $GST > t$, if Σ conforms to M_2.
- Each process p_i, $i = 1, \ldots, n$, behaves exactly like in run R_i up to time t.

Clearly, a process p_i, $i = 1, \ldots, n$, cannot distinguish run R from run R_i up to time t. Hence, at time $t_i \leq t$ it will suspect the rest of processes in R, as it did in R_i. Since this is true for every process in the system, in run R all correct processes are suspected at some time by the rest of correct processes, and the weak accuracy property is not satisfied. □

3.2.5 Impossibility for any $f < n$

Theorem 3. *Let FD_Σ be a failure detector, implemented on the system Σ, that satisfies the strong completeness property. Then, FD_Σ cannot satisfy the weak accuracy property.*

Proof. Let us consider a run R_1 of Σ in which only process p_1 crashes, doing it at time $t_0 = 0$. Since FD_Σ satisfies the strong completeness property, there is some time t_1 at which all other processes permanently suspect p_1 in R_1.

Let us consider now a run R_2 of Σ in which only process p_2 crashes, doing it at the time t_1 defined in R_1, and all messages sent by p_1 are received after t_1 (this can happen if we assume that $\Delta > t_1$, if Σ conforms to M_1, or $GST > t_1$, if Σ conforms to M_2). Also, in R_2, all processes except p_1 behave exactly like in run R_1 up to time t_1. Clearly, all correct processes, except p_1, cannot distinguish run R_2 from run R_1 up to time t_1. Hence, at time t_1 they will suspect p_1 in R_2, as they did in R_1. Finally, since FD_Σ satisfies the strong completeness property, there is some time $t_2 \geq t_1$ at which all other processes permanently suspect p_2 in R_2.

Generalizing this reasoning, we obtain n runs R_i, $i = 1, \ldots, n$ of Σ as follows. In run R_i, only process p_i crashes, at time t_{i-1}, defined in R_{i-1}, and for each process p_k, $k = 1, \ldots, i-1$, all messages sent by p_k after t_{k-1} are received after t_k, with $t_0 = 0$ (this can happen if we assume that $\Delta > t_{i-1}$, if Σ conforms to M_1, or $GST > t_{i-1}$, if Σ conforms to M_2). Also, in R_i, for each process p_k, $k = 1, \ldots, i-1$, all processes except p_k behave exactly like in run R_k up to time t_k. Clearly, for each process p_k, $k = 1, \ldots, i-1$, all correct processes except p_k cannot distinguish run R_i from run R_k up to time t_k. Hence, at time t_k they will suspect p_k in R_i, as they did in R_k. Finally, since FD_Σ satisfies the strong completeness property, there is some time $t_i \geq t_{i-1}$ at which all other processes permanently suspect p_i in R_i.

Let us now consider a run R of Σ in which no process crashes, but:

- All messages sent by p_n after time t_{n-1} as defined in R_{n-1} are received after time t_n as defined in R_n. This can happen if we assume that $\Delta > t_n$, if Σ conforms to M_1, or $GST > t_n$, if Σ conforms to M_2.
- For each process p_i, $i = 1, \ldots, n$, all processes except p_i behave exactly like in run R_i up to time t_i.

Clearly, for each process p_i, $i = 1, \ldots, n$, all processes except p_i cannot distinguish run R from run R_i up to time t_i. Hence, at time t_i they will suspect p_i in R, as they did in R_i. Since this is true for every process p_i, $i = 1, \ldots, n$ in the system, in run R all correct processes are suspected at some time by the rest of correct processes, and the weak accuracy property is not satisfied. □

Corollary 2. *There is no protocol that implements a failure detector of class $\mathcal{S}$ in a partially synchronous distributed system that conforms to model M_1 or model M_2.*

Corollary 3. *There is no protocol that implements a failure detector of class $\mathcal{W}$ in a partially synchronous distributed system that conforms to model M_1 or model M_2.*

Proof. Follows from Corollary 2 and the fact that $\mathcal{S}$ and $\mathcal{W}$ are equivalent [6]. □

3.3 Impossibility of Implementing Θ without a Majority of Correct Processes

In this section, we show that the failure detector Θ, proposed by Aguilera et al. in [2], cannot be implemented in the models of partial synchrony M_1 and M_2 without a majority of correct processes.

Theorem 4. *Let Σ be a partially synchronous distributed system that conforms to model M_1 or model M_2, made up of $n > 1$ processes. Let FD_Σ be a failure detector, implemented on the system Σ, that satisfies the Θ-completeness property. Then, if $f \geq \lceil n/2 \rceil$, FD_Σ cannot satisfy the Θ-accuracy property.*

Proof. Let us consider a run R of Σ in which processes $p_1, p_2, \ldots, p_{\lceil n/2 \rceil}$ crash at time 0. Since FD_Σ satisfies the Θ-completeness property, there is some time t at which p_n permanently suspects all these processes.

Let us now consider a run R' in which processes $p_{\lceil n/2 \rceil+1}, \ldots, p_n$ crash at time $t+1$, and:

- All messages sent by processes $p_1, p_2, \ldots, p_{\lceil n/2 \rceil}$ are received after time t. This can happen if we assume that $\Delta > t$, if Σ conforms to M_1, or $GST > t$, if Σ conforms to M_2.
- Each process $p_{\lceil n/2 \rceil+1}, \ldots, p_n$ behaves exactly like in run R up to time t.

Clearly, process p_n cannot distinguish run R' from run R up to time t. Hence, at time t, it will suspect all the processes $p_1, p_2, \ldots, p_{\lceil n/2 \rceil}$ which are the correct processes of the run R'. Hence, in run R', all correct processes are suspected at time t by process p_n, and the Θ-accuracy property is not satisfied since there is a time at which some process does not trust any correct process. □

Corollary 4. *There is no protocol that implements a failure detector of class Θ in a partially synchronous distributed system that conforms to model M_1 or model M_2 without a majority of correct processes.*

4 Implementing Eventual Failure Detectors

In this section, we present a family of algorithms that implement the four classes of unreliable failure detectors satisfying eventual accuracy, namely, $\diamond\mathcal{W}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{P}$. The algorithms are based on a logical ring arrangement of the processes, which defines the monitoring and failure information propagation pattern. As a consequence, the resulting algorithms periodically exchange at most a linear number of messages.

The algorithms have been designed and are presented as a sequence of refinements. First, we present a basic algorithm providing *only* weak completeness (i.e., it does not provide any accuracy). Next, we present two extensions to this basic algorithm that provide the two eventual accuracy properties, respectively, and a third extension that transforms weak completeness into strong completeness while preserving accuracy. Hence, the basic algorithm combined with the extensions implement the four classes of failure detectors.

4.1 Definitions

In the algorithms presented in this section, we consider the processes $p_1, \ldots, p_n$ arranged in a logical ring. This arrangement is known by all the processes. Without loss of generality, process p_i is followed in the ring by process $p_{(i \bmod n)+1}$. In general, we use $succ(p)$ to denote the process that follows process p in the ring, and $pred(p)$ to denote the process that precedes process p in the ring. Finally, for a given run r, we use $corr\_succ_r(p)$ and $corr\_pred_r(p)$ to denote the closest correct (i.e., belonging to the subset $correct_r$) successor and predecessor of p in the ring in r, respectively.

Contrary to the case of proving impossibility results, when designing a distributed algorithm it is convenient to consider the weakest model of partial synchrony because the algorithm remains correct in the stronger models. Hence, in all our algorithms, we will consider a partially synchronous system conforming to the model M_3. In this model, we shall use GST_r to denote the ending instant of the stabilization interval in the run r of interest. We will also denote by Δ^r_{msg} the maximum time, after GST_r, between the sending of a message and the delivery and processing by its destination process (assuming that both the sender and the destination have not crashed) in run r. Clearly, Δ^r_{msg} depends on the existing bounds on processor relative speeds and on message transmission time. Note that the exact value of Δ^r_{msg} exists, but it is unknown.

We denote by L_p the list of suspected processes of the failure detection module attached to process p. Clearly, the contents of the list L_p can be different at different times. We use $L_p(t)$ to denote the contents of L_p at time t.

The algorithms use a polling monitoring model. To monitor process q, a process p sends an ARE-YOU-ALIVE? message to q and waits for an I-AM-ALIVE message from it. As soon as q receives the ARE-YOU-ALIVE? message, it sends the I-AM-ALIVE message to p. We will denote by $\Delta^r_{rtt} = 2\Delta^r_{msg}$ the maximum monitoring round-trip time after stabilization, i.e., the maximum time, after GST_r, elapsed between the sending of an ARE-YOU-ALIVE? message to a correct process, and the reception and processing of the corresponding I-AM-ALIVE reply message.

Since a monitoring process p does not know Δ^r_{rtt}, it has to use an estimated value (timeout) that tells how much time it has to wait for the reply from the monitored process q. This time value is denoted by $\Delta_{p,q}$. Then, if after $\Delta_{p,q}$ time p did not receive the reply from q, it suspects that q has crashed. We need to allow these time values to vary over time in the algorithms. We use $\Delta_{p,q}(t)$ to denote the value of $\Delta_{p,q}$ at time t.

4.2 A Basic Algorithm that Provides Weak Completeness

We present here an algorithm that will be used as a framework for all the failure detector implementations presented in this section. This first algorithm satisfies the weak completeness property. In the following sections, we will extend the algorithm to satisfy also eventual weak accuracy, eventual strong accuracy, and strong completeness. This algorithm is presented here for the sake of clarity but is not very useful by itself since it does not satisfy any of the accuracy properties previously defined.

The algorithm executes as follows: Initially, every process starts monitoring its successor in the ring. If a process p does not receive the reply from the process q it is monitoring, then p suspects that q has crashed, and starts monitoring the successor of q in the ring. This monitoring scheme is repeated, so that p always suspects all processes in the ring between itself and the process it is monitoring (not included). If, later on, p receives a message from a suspected process q while it is monitoring another process r,

```
Every process p executes:

$target_p \leftarrow succ(p)$
$L_p \leftarrow \emptyset$
$\forall q \in \Pi : \Delta_{p,q} \leftarrow$ default timeout

cobegin

  || Task 1:
    loop
      if $target_p \neq p$ then
        send ARE-YOU-ALIVE? to $target_p$
        $t_{out} \leftarrow \Delta_{p,target_p}$
        $received \leftarrow false$
        delay $t_{out}$
        if not $received$ then
          $L_p \leftarrow L_p \cup \{target_p\}$
          $target_p \leftarrow succ(target_p)$
        end if
      end if
    end loop

  || Task 2:
    loop
      receive message $m$ from a process $q$
      if $q = target_p$ then
        $received \leftarrow true$
      else if $q \in L_p$ then
        $L_p \leftarrow L_p - \{q, \ldots, pred(target_p)\}$
        $target_p \leftarrow q$
        $received \leftarrow true$
      end if
      if $m =$ ARE-YOU-ALIVE? then
        send I-AM-ALIVE to $q$
      end if
    end loop

coend
```

Fig. 2. Algorithm that provides weak completeness.

then p stops suspecting q and all the processes between q and r in the ring, and starts monitoring q again.

Fig. 2 presents the algorithm in detail. Each process p has a variable $target_p$ which holds the process being monitored by p at a given time. As we said above, all processes between p and $target_p$ in the ring (and only them) are suspected by p, and these are the only processes included in the list L_p of suspected processes of p. (Initially, no process is suspected, i.e., $\forall p : L_p(0) = \emptyset$.)

We now show that weak completeness holds with this algorithm. For every run r, given an incorrect process p, the following theorem states that it will be permanently suspected by $corr\_pred_r(p)$ (the first correct process preceding p in the ring in r).

Theorem 5. *$\forall r : \exists t_0 : \forall p \in crashed_r,$ p has failed at time t_0 and $\forall t \geq t_0, p \in L_{corr\_pred_r(p)}(t)$.*

Proof. Let p be a process that crashes, doing it at time $Tcrash_p^r$. We claim that p will be permanently included in $L_{corr\_pred_r(p)}$. The proof uses strong induction on the distance from $corr\_pred_r(p)$ to p. Let's first consider that such distance is 1, i.e., $corr\_pred_r(p) = pred(p)$. Before p fails, $corr\_pred_r(p)$ and p exchange ARE-YOU-ALIVE? and I-AM-ALIVE messages (see Fig. 2). Eventually, p crashes, and there is an ARE-YOU-ALIVE? message sent by $corr\_pred_r(p)$ that reaches p after $Tcrash_p^r$. Since p has already crashed by then, it will never reply to that message. If such a message was sent at time t', then $\Delta_{corr\_pred_r(p),p}(t')$ time later, $corr\_pred_r(p)$ will include p in $L_{corr\_pred_r(p)}$. Since no message will ever be received by $corr\_pred_r(p)$ from p after that, it will never be removed from $L_{corr\_pred_r(p)}$.

We will now prove that, if the claim holds for any distance $1 \leq d \leq i-1$, it also holds for distance i. Let us assume the distance from $corr\_pred_r(p)$ to p be $i > 1$. Then, for any process $q \in \{succ(corr\_pred_r(p)), \ldots, pred(p)\}$, it can be easily seen that $corr\_pred_r(q) = corr\_pred_r(p)$ and the distance d from $corr\_pred_r(p)$ to q verifies $1 \leq d \leq i-1$. Hence, from the induction hypothesis, all processes in $\{succ(corr\_pred_r(p)), \ldots, pred(p)\}$ will eventually be permanently in $L_{corr\_pred_r(p)}$. After that, they will never be monitored again by $corr\_pred_r(p)$. The situation then is similar to the distance-1 case considered above and, by a similar argument, p will eventually be permanently included in $L_{corr\_pred_r(p)}$. □

Corollary 5. *The algorithm of Fig. 2 provides weak completeness.*

Observe that a process p will never suspect any process beyond $target_p$. If p never suspects $succ(p)$ (i.e., $target_p = succ(p)$ at all times) in some run r, and some other process q (different from p and $succ(p)$) fails, p will never suspect q either.

Observation 1. *The algorithm of Fig. 2 does not provide strong completeness.*

4.3 Extending the Basic Algorithm to Provide Eventual Weak Accuracy

The algorithm presented in the previous section does not satisfy any of the accuracy properties defined in Section 1.2. It does not prevent the erroneous suspicion of any correct process, and these incorrect suspicions, although not permanent (if the suspected process is correct, the reply message will eventually be received), can happen infinitely often. This is due to the fact that the message delivery time could be greater than the fixed *default timeout* (see Fig. 2). In order to provide some useful accuracy, the timeout values must be augmented when processes are aware of having erroneously suspected a correct process. In this section, we present an extension to the basic algorithm of Fig. 2, based on augmenting the timeout values, which satisfies the eventual weak accuracy property.

```
Every process p executes:

$initial\_cand_p \leftarrow$ pre-agreed process, e.g., $p_1$
$target_p \leftarrow succ(p)$
$L_p \leftarrow \emptyset$
$\forall q \in \Pi : \Delta_{p,q} \leftarrow$ default timeout

cobegin

 || Task 1:                                                      || Task 2:
   loop
     if $target_p \neq p$ then                                       { Same as in Fig. 2 }
       send ARE-YOU-ALIVE? to $target_p$
       $t_{out} \leftarrow \Delta_{p,target_p}$
       $received \leftarrow false$
       delay $t_{out}$
       if not $received$ then
         if $initial\_cand_p \in \{succ(p), \ldots, target_p\}$ then
           $\Delta_{p,target_p} \leftarrow \Delta_{p,target_p} + 1$
         end if
         $L_p \leftarrow L_p \cup \{target_p\}$
         $target_p \leftarrow succ(target_p)$
       end if
     end if
   end loop

coend
```

Fig. 3. Extension to the algorithm of Fig. 2 to provide eventual weak accuracy.

Eventual weak accuracy requires that, eventually, some correct process is never suspected by any correct process. In order to provide it, it is enough that this is satisfied for only one correct process. We propose an extension to the basic algorithm that guarantees the existence of such a process, for each run r, which we denote $leader_r$. Clearly, if we knew beforehand the identity of a correct process, then eventual weak accuracy could be obtained by making all processes augment their timeout value with respect to this process each time they suspect it. This correct process would be $leader_r$. But, since we cannot know in advance the correctness of any process, we need to devise another way to eventually have a correct and not-suspected process.

In the extension of the algorithm of Fig. 2 that we propose, processes behave as follows: Initially, every process will consider a preagreed upon process (e.g. p_1) as an *initial candidate* to be $leader_r$. When a process p that monitors this candidate suspects it, p considers the successor of the candidate in the ring—$succ(candidate)$—as new candidate and monitors it. This scheme is repeated every time the current candidate is suspected. (Note that a process not monitoring a candidate cannot suspect it.) Finally, each time p suspects a candidate, the timeout value of p with respect to this candidate is increased. This way, for a given run r, $leader_r$ will be the first correct process in the ring starting from the *initial candidate* (inclusive). All processes monitoring it will eventually stop suspecting it, and processes that do not monitor it will never suspect it. This gives us the eventually weak accuracy property. Fig. 3 presents the extended algorithm in detail.

We now show that eventual weak accuracy holds with this algorithm, i.e., eventually some correct process is never suspected by any correct process.

Lemma 1. *For every run r, after GST_r, any correct process p will suspect $leader_r$ for no more than Δ^r_{rtt} time, each time it does.*

Proof. Remember that, after GST_r, Δ^r_{rtt} is a bound on the monitoring round-trip time. A correct process p suspects $leader_r$ after sending an ARE-YOU-ALIVE? message to it at time t and not receiving an I-AM-ALIVE reply message in $\Delta_{p,leader_r}(t)$ time. Since, by definition, $leader_r$ is a correct process, the I-AM-ALIVE message will arrive at most at time $t + \Delta^r_{rtt}$ ($GST_r + \Delta^r_{rtt}$ if $t < GST_r$). At this moment, $leader_r$ is removed from L_p, the list of suspected processes of p. □

Lemma 2. *$\forall r$, any correct process p will suspect $leader_r$ a finite number of times.*

Proof. The proof is by contradiction. Let p be some correct process. Let us assume that p suspects $leader_r$ an infinite number of times. From the algorithm, each time p suspects $leader_r$, the value of $\Delta_{p,leader_r}$ is incremented by one. Since GST_r and Δ^r_{rtt} are finite (although unknown), after suspecting $leader_r$ a finite number of times, $\Delta_{p,leader_r}$ will be greater than Δ^r_{rtt}. After this moment, p never suspects $leader_r$ anymore and, thus, we reach a contradiction. □

Theorem 6.

$$\forall r : \exists t_1 : \forall p \in correct_r, \forall t > t_1, leader_r \notin L_p(t).$$

Proof. Let t^p_1 be the instant at which a correct process p stops suspecting $leader_r$ for the last time. (If p never suspects $leader_r$, $t^p_1 = 0$.) Such an instant exists from Lemma 1 and Lemma 2. Then, after instant $t_1 = \max_{p \in correct_r} \{t^p_1\}$ no correct process p has $leader_r$ in its list L_p. □

```
Every process p executes:

$target_p \leftarrow succ(p)$
$L_p \leftarrow \emptyset$
$\forall q \in \Pi : \Delta_{p,q} \leftarrow$ default timeout

cobegin

  || Task 1:                                                    || Task 2:
    loop                                                          { Same as in Fig. 2 }
      if $target_p \neq p$ then
        send ARE-YOU-ALIVE? to $target_p$
        $t_{out} \leftarrow \Delta_{p,target_p}$
        $received \leftarrow false$
        delay $t_{out}$
        if not $received$ then
          $\Delta_{p,target_p} \leftarrow \Delta_{p,target_p} + 1$
          $L_p \leftarrow L_p \cup \{target_p\}$
          $target_p \leftarrow succ(target_p)$
        end if
      end if
    end loop

coend
```

Fig. 4. Extension to the algorithm of Fig. 2 to provide eventual strong accuracy.

Corollary 6. *The algorithm of Fig. 3 provides eventual weak accuracy.*

Observation 2. *The only difference between the algorithm of Fig. 3 and the algorithm of Fig. 2 is that in the former the values of $\Delta_{p,q}$ can change. Clearly, this does not affect the proof of Theorem 5. Hence, Corollary 5 also applies to this algorithm.*

Corollary 7. *The algorithm of Fig. 3 implements a failure detector of class $\diamond\mathcal{W}$.*

Proof. Follows from Corollary 6, Observation 2, and Corollary 5. □

Since Observation 1 still applies to this algorithm, we have that the failure detector it implements does not satisfy strong completeness and, hence, is not in $\diamond\mathcal{S}$. On the other hand, in some run r, we may have two processes p and $succ(p)$ beyond $leader_r$, such that both are correct but p keeps suspecting $succ(p)$ over and over. This can happen if the default timeout is smaller than Δ^r_{rtt}. Then, the eventual strong accuracy is not satisfied in r and, hence, the failure detector implemented is not in $\diamond\mathcal{Q}$.

4.4 Extending the Basic Algorithm to Provide Eventual Strong Accuracy

Eventual strong accuracy requires that, eventually, no correct process is ever suspected by any correct process. In this section, we propose another extension to the basic algorithm of Fig. 2 which provides this property. Broadly, the extension consists in each process augmenting its timeout values with respect to all processes it incorrectly suspects. This way, every process will augment the timeout value with respect to its closest correct successor in the ring, and will thus eventually stop suspecting it (and, hence, any other correct process). This gives us the eventually strong accuracy property. Fig. 4 presents the extended algorithm in detail.

We now show that eventual strong accuracy holds with the algorithm in Fig. 4. We start with two lemmas, whose proofs are very similar to those of Lemma 1 and Lemma 2, respectively, and are omitted.

Lemma 3. *For every run r, after GST_r, any correct process p will suspect $corr\_succ_r(p)$ for no more than Δ^r_{rtt} time, each time it does.*

Lemma 4. *For every run r, any correct process p will suspect $corr\_succ_r(p)$ a finite number of times.*

Theorem 7.

$$\forall r : \exists t_2 : \forall p \in correct_r, \forall q \in correct_r, \forall t > t_2, q \notin L_p(t).$$

Proof. Let t_2^p be the instant at which a correct process p stops suspecting $corr\_succ_r(p)$ for the last time. (If p never suspects $corr\_succ_r(p)$, $t_2^p = 0$.) Such an instant exists from Lemma 3 and Lemma 4. Then, after instant

$$t_2 = \max_{p \in correct_r} \{t_2^p\},$$

no correct process p has $corr\_succ_r(p)$ in its list L_p. Then, after t_2, each correct process p only suspects processes in $succ(p), \ldots, pred(corr\_succ_r(p))$, which are not correct by the definition of $corr\_succ_r(p)$. Therefore, no correct process q is in L_p after t_2. □

Corollary 8. *The algorithm of Fig. 4 provides eventual strong accuracy.*

Note that Observation 2 still applies to the algorithm of Fig. 4. Hence, the following corollary, that follows from Corollary 8, Observation 2, and Corollary 2.

```
Every process $p$ executes:

{if the algorithm needs it:
$initial\_cand_p \leftarrow$ pre-agreed process, e.g., $p_1$}
$target_p \leftarrow succ(p)$
$L_p \leftarrow \emptyset$
$G_p \leftarrow \emptyset$
$\forall q \in \Pi : \Delta_{p,q} \leftarrow$ default timeout

cobegin

 || Task 1:
   loop
     if $target_p \neq p$ then
       send ARE-YOU-ALIVE?
             —with $G_p$— to $target_p$
       $t_{out} \leftarrow \Delta_{p,target_p}$
       $received \leftarrow false$
       delay $t_{out}$
       if not $received$ then
         {Update $\Delta_{p,target_p}$ if required}
         $G_p \leftarrow G_p \cup \{target_p\}$
         $L_p \leftarrow L_p \cup \{target_p\}$
         $target_p \leftarrow succ(target_p)$
       end if
     end if
   end loop

 || Task 2:
   loop
     receive message $m$ from a process $q$
     if $q = target_p$ then
       $received \leftarrow true$
     else if $q \in L_p$ then
       $L_p \leftarrow L_p - \{q, \ldots, pred(target_p)\}$
       $G_p \leftarrow G_p - \{q\}$
       $target_p \leftarrow q$
       $received \leftarrow true$
     end if
     if $m$ = ARE-YOU-ALIVE? —with $G_q$— then
       send I-AM-ALIVE to $q$
       $G_p \leftarrow G_q \cup L_p - \{p, q\}$
     end if
   end loop

coend
```

Fig. 5. Extension to the previous algorithms to provide strong completeness.

Corollary 9. *The algorithm of Fig. 4 implements a failure detector of class $\diamond\mathcal{Q}$.*

Observation 1 still applies to this algorithm and, hence, it does not implement a $\diamond\mathcal{P}$ failure detector.

4.5 Extending the Previous Algorithms to Provide Strong Completeness

In this section, we present an extension to the previous algorithms to provide strong completeness, while preserving accuracy. By combining this extension with the algorithms that implement failure detectors of classes $\diamond\mathcal{W}$ and $\diamond\mathcal{Q}$ of Fig. 3 and Fig. 4, we obtain implementations of failure detectors of classes $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$, respectively.

Strong completeness requires that, eventually, every process that crashes is permanently suspected by every correct process. In [6], Chandra and Toueg presented a distributed algorithm that transforms weak completeness into strong completeness. Broadly, in their algorithm, every process periodically *broadcasts* (sends to every other process) its *local* list of suspected processes. Upon reception of these lists, each process builds a *global* list of suspected processes, which provides strong completeness. Clearly, in this algorithm, each correct process periodically sends n messages, with the total number of messages exchanged being at least $n\mathcal{C}_r$ (with $\mathcal{C}_r$ being the number of correct processes for a given run r).

We propose an extension that follows a similar approach. Besides its local list L_p of suspected processes, each process p has a global list G_p of suspected processes. While L_p only holds the suspected processes between p and the process p is monitoring ($target_p$), G_p holds all the processes that are being suspected in the system. Now, the global lists are the ones providing strong completeness.

In order to correctly build the global lists, processes need to propagate their local lists. However, instead of periodically broadcasting its local list, every process will only send its global list (which contains the local list) to the process it is monitoring. This process, upon reception of that list, updates its global list and further propagates it. Note that, since we use the ring arrangement of processes, each process, at most, sends and receives one message periodically, and the total number of messages exchanged in a period is $O(n)$ in the worst case, which eventually becomes $O(\mathcal{C}_r)$. Furthermore, instead of using specific messages to send the global lists, we can piggyback the global lists in the ARE-YOU-ALIVE? messages inherent to the monitoring mechanism. This way, there is no increment in message exchanges from the previous algorithms. Fig. 5 presents the extended algorithm in detail. We now show that strong completeness holds, while accuracy is preserved, with this algorithm.

Observation 3. *The only difference between this algorithm and the previous ones is the handling of the global lists of suspected processes G_p, while the local lists L_p are handled as before. Hence, Theorem 5 and whichever corresponds to Theorem 6 and Theorem 7 are still applicable to this algorithm.*

Observation 4. $\forall p \in \Pi, \forall r, \forall t, L_p(t) \subseteq G_p(t)$.

Observation 5. *$\forall r, \forall p \in correct_r, \forall t$, p will eventually receive ARE-YOU-ALIVE? messages after t (unless it is the only correct process).*

Lemma 5.

$$\forall r : \exists t_3 : \forall q \in crashed_r, \forall p \in correct_r, \forall t \geq t_3, q \in G_p(t).$$

Proof. Let us assume we are at least at instant t_0 as defined in Theorem 5. We know that at this instant any process $q \in crashed_r$ has already failed and has been permanently included in $L_{corr\_pred_r(q)}$.

Let us assume now that we have a process $q \in crashed_r$ and a process $p \in correct_r$. We claim that q will eventually be permanently included in G_p. We use strong induction on the number of correct processes in the set

$$\{corr\_pred_r(q), \ldots, p\}.$$

For the base case, we assume there is only one correct process in the set, i.e., $p = corr\_pred_r(q)$. Hence, from Theorem 5, q is permanently in L_p and, from Observation 4, q will be permanently in G_p in this case.

We will now prove that, if the claim holds for any number $1 \leq c \leq i-1$ of correct processes in the set $\{corr\_pred_r(q), \ldots, p\}$, it also holds when the number of correct processes in the set is i. To do so, we show first that there is a time t' after which p receives ARE-YOU-ALIVE? messages and all of them carry global lists containing q. From that, it is immediate to see in the algorithm that, after receiving the first such ARE-YOU-ALIVE? message, q will be permanently included in G_p. Let us assume the number of correct processes in the set $\{corr\_pred_r(q), \ldots, p\}$ be $i > 1$. By induction hypothesis, there is a time t'' at which any correct process $r \in \{corr\_pred_r(q), \ldots, corr\_pred_r(p)\}$ permanently contains q in its global list G_r. Also, there is a time $t' = \max(t'', GST_r) + \Delta^r_{msg}$ at which all the ARE-YOU-ALIVE? messages sent to p before t'' have been received. From Observation 5, process p will receive new ARE-YOU-ALIVE? messages after t'. Let be an ARE-YOU-ALIVE? message received by p from a process s at a time $t > t'$. There are two cases to consider:

- $s \in \{corr\_pred_r(q), \ldots, corr\_pred_r(p)\}$. In this case, from the induction hypothesis and the definition of t', we know that the global list G_s carried by the ARE-YOU-ALIVE? message contains q.
- $s \in \{p, \ldots, corr\_pred_r(corr\_pred_r(q))\}$. In this case, it can be seen from the algorithm that, if p receives an ARE-YOU-ALIVE? message from s, then necessarily, at the time of sending the message, $p = target_s$ and L_s contained q. Therefore, from Observation 4, the G_s carried by the ARE-YOU-ALIVE? message contains q. □

The following lemma states that the algorithm of Fig. 5 preserves eventual accuracy.

Lemma 6. *Let p be any correct process for a given run r. If there is a time after which no correct process q contains p in L_q in r, then there is a time after which no correct process q contains p in G_q in r.*

Proof. Let us assume we are at least at instant t_0 as defined in Theorem 5. We know that, at this instant, any process in $crashed_r$ has already failed. Let p be a correct process and $t''' \geq t_0$ be an instant such that

$$\forall t \geq t''', \forall q \in correct_r, p \notin L_q.$$

Let us assume now that we have a process $q \in correct_r$. We claim that there is a time after which p is never in G_q. We use strong induction on the number of correct processes in the set $\{p, \ldots, q\}$. For the base case, we assume there is only one correct process in the set, i.e., $p = q$. It is easy to observe from the algorithm that p will never include itself in G_p.

We will now prove that, if the claim holds for any number $1 \leq c \leq i-1$ of correct processes in the set $\{p, \ldots, q\}$, it also holds when the number of correct processes in the set is i. To do so, we show first that there is a time t' after which q receives ARE-YOU-ALIVE? messages and all of them carry global lists not containing p. From that, it is immediate to see in the algorithm that, after receiving the first such ARE-YOU-ALIVE? message, p will be removed (if needed) and never included again in G_q. Let us assume the number of correct processes in the set $\{p, \ldots, q\}$ be $i > 1$. By induction hypothesis, there is a time t'' after which any correct process

$$r \in \{p, \ldots, corr\_pred_r(q)\}$$

does not contain p in its global list G_r. Also, there is a time $t' = \max(t'', GST_r) + \Delta^r_{msg}$ at which all the ARE-YOU-ALIVE? messages sent to q before t'' have been received. From Observation 5, process q will receive new ARE-YOU-ALIVE? messages after t'. Let an ARE-YOU-ALIVE? message be received by q from a process s at a time $t > t'$. There are two cases to consider:

- $s \in \{p, \ldots, corr\_pred_r(q)\}$. In this case, from the induction hypothesis and the definition of t', we know that the global list G_s carried by the ARE-YOU-ALIVE? message does not contain p.
- $s \in \{q, \ldots, corr\_pred_r(p)\}$. This case cannot happen because it would imply that p is in the local list L_s. □

Combining both lemmas it is immediate to derive the following theorem:

Theorem 8. *The algorithm of Fig. 5 provides strong completeness while preserving accuracy.*

Corollary 10. *The algorithm of Fig. 5, combined with the algorithm of Fig. 3, implements a failure detector of class $\diamond\mathcal{S}$.*

Corollary 11. *The algorithm of Fig. 5, combined with the algorithm of Fig. 4, implements a failure detector of class $\diamond\mathcal{P}$.*

4.6 Performance Analysis

In this section, we will evaluate the performance of the presented algorithms in terms of the number and size of the exchanged messages. Observe that failure detection is an ongoing activity that inherently requires an unbounded number of messages. Furthermore, the pattern of message exchange between processes can vary over time (and need not be periodic), and different algorithms can have completely different patterns. For these reasons, we have to make some assumptions in order to use the number of messages as a meaningful performance measure. We will first assume that the algorithms execute in a periodic[6] fashion, so that we can count the number of messages exchanged in a period. Second, to be able to compare the number of messages exchanged by different algorithms, we must assume that their respective periods have the same length.

Under the above assumptions, in our algorithms each correct process periodically polls only one other process. Each polling involves two messages. Thus, a total of no more than $2n$ messages would be periodically exchanged. Eventually, in a run r, this amount becomes $2\mathcal{C}_r$ since there will be only $\mathcal{C}_r$ correct processes remaining in the system. This compares favorably with Chandra and Toueg's algorithm, which requires a periodic exchange of at least $n\mathcal{C}_r$ messages.

Concerning the size of the messages, our algorithms implementing failure detectors with weak completeness, i.e., $\diamond\mathcal{W}$ and $\diamond\mathcal{Q}$, require messages of $\Theta(\log n)$ bits (to identify the sender). On the other hand, the algorithms implementing failure detectors with strong completeness, i.e., $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$, require messages of $\Theta(n)$ bits since we can code the global list G_p of suspected processes in n bits (one bit per process).

Chandra and Toueg's algorithm, which implements $\diamond\mathcal{P}$, requires messages of $\Theta(\log n)$ bits. This size is smaller than the size needed by our algorithm implementing $\diamond\mathcal{P}$. However, the total amount of information periodically exchanged in our algorithms is $\Theta(n^2)$ bits, while in Chandra-Toueg's it is $\Theta(n^2 \log n)$ bits. Furthermore, each message that is sent involves a fixed overhead. In this sense, our algorithms present an edge since they involve less messages.

A drawback of our algorithms satisfying strong completeness is that the use of the ring to propagate the information about crashes in order to obtain strong completeness delays the detection of faulty processes (relative to the use of direct message exchanges between all pairs of processes). Intuitively, this time increases linearly with the number of processes in the system, while in the case of the algorithm of Chandra and Toueg remains practically constant. Clearly, there is a trade off between the number of *extra* messages sent and the latency of failure information propagation. For example, we could propagate the list of suspects following the two directions of the ring. This would reduce the latency approximately to the half, at the price of increasing the number of messages.

5 Conclusions

In this paper, we have first shown the impossibility of implementing several classes of unreliable failure detectors in partially synchronous systems. The models of partially synchronous systems we consider are at least as strong as those proposed in [6], [10] and, hence, our results apply to those as well. We show that no perpetual failure detector from those proposed by Chandra and Toueg in [6] can be implemented, and that to implement a failure detector of class Θ a majority of correct processes is required.

Then, we have proposed several algorithms to implement failure detectors of classes $\diamond\mathcal{W}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{P}$. These algorithms are efficient alternatives to the algorithm implementing $\diamond\mathcal{P}$ proposed by Chandra and Toueg [6].

One line of work derived from this paper is the investigation of more efficient implementations of the eventual failure detectors than those presented here. In fact, after the algorithms presented here appeared in conference paper form [16], more efficient algorithms for the classes $\diamond\mathcal{S}$ [17] and $\diamond\mathcal{P}$ [1], [18] have been proposed.

6. For algorithms based on the heartbeat model, we assume that all processes send I-AM-ALIVE messages with the same period. For algorithms based on the polling model, we assume that there are no incorrect suspicions, and that all the timeout values are identical. Thus, the timeout value can be viewed as the periodicity of the algorithm.

Acknowledgments

The authors are grateful to André Schiper for his valuable comments. The authors also thank Sam Toueg for suggesting the indirect proof of impossibility presented in Section 3.1. The research was partially supported by the Basque Government, under grant BFI96.043, the Spanish Research Council (MCyT), under grants TIC99-0280-C02-02, TEL99-0582, TIC98-1032-C03-01, and TIC2001-1586-C03-01, the Madrid Regional Research Council (CAM), under grants CAM-07T/00112/1998 and CAM-07T/0022/2003, and the Universidad Rey Juan Carlos, under grants PPRC-2001/10 and PPR-2003/37.

References

[1] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Stable Leader Election," *Proc. 15th Int'l Symp. DIstributed Computing (DISC '2001)*, pp. 108-122, Oct. 2001.

[2] M. Aguilera, S. Toueg, and B. Deianov, "Revisiting the Weakest Failure Detector for Uniform Reliable Broadcast," *Proc. 13th Int'l Symp. DIstributed Computing (DISC '99)*, pp. 19-33, Sept. 1999.

[3] E. Anceaume, A. Fernández, A. Mostefaoui, and M. Raynal, "A Necessary and Sufficient Condition for Transforming Limited Accuracy Failure Detectors," *Submitted to journal publication*, 2001.

[4] M. Bertier, O. Marin, and P. Sens, "Implementation and Performance Evaluation of an Adaptable Failure Detector," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN '02)*, pp. 354-363, June 2002.

[5] T.D. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *J. ACM*, vol. 43, no, 4, pp. 685-722, July 1996.

[6] T.D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, pp. 225-267, Mar. 1996.

[7] W. Chen, S. Toueg, and M.K. Aguilera, "On the Quality of Service of Failure Detectors," *IEEE Trans. Computers*, vol. 51, no. 5, pp. 561-580, 2002.

[8] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, "A Realistic Look at Failure Detectors," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN '2002)*, pp. 345-352, June 2002.

[9] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *J. ACM*, vol. 34, no. 1, pp. 77-98, Jan. 1987.

[10] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, pp. 288-323, Apr. 1988.

[11] C. Fetzer, M. Raynal, and F. Tronel, "An Adaptive Failure Detection Protocol," *Proc. Eighth IEEE Pacific Rim Int'l Symp. Dependable Computing (PRDC '2001)*, pp. 146-153, Dec. 2001.

[12] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, pp. 374-382, Apr. 1985.

[13] R. Guerraoui, M. Larrea, and A. Schiper, "Non-Blocking Atomic Commitment with an Unreliable Failure Detector," *Proc. 14th Symp. Reliable Distributed Systems (SRDS '95)*, pp. 41-51, Sept. 1995.

[14] R. Guerraoui and A. Schiper, "Γ-Accurate Failure Detectors," *Proc. 10th Int'l Workshop Distributed Algorithms (WDAG '96)*, pp. 269-286, Oct. 1996.
[15] I. Gupta, T.D. Chandra, and G. Goldszmidt, "On Scalable and Efficient Distributed Failure Detectors," *Proc. 20th Ann. ACM Symp. Principles of Distributed Computing (PODC '2001)*, pp. 170-179, Aug. 2001.
[16] M. Larrea, S. Arévalo, and A. Fernández, "Efficient Algorithms to Implement Unreliable Failure Detectors in Partially Synchronous Systems," *Proc. 13th Int'l Symp. DIstributed Computing (DISC '99)*, pp. 34-48, Sept. 1999.
[17] M. Larrea, A. Fernández, and S. Arévalo, "Optimal Implementation of the Weakest Failure Detector for Solving Consensus," *Proc. 19th IEEE Symp. Reliable Distributed Systems (SRDS '2000)*, pp. 52-59, Oct. 2000.
[18] M. Larrea, A. Fernández, and S. Arévalo, "Eventually Consistent Failure Detectors," *Proc. 10th Euromicro Workshop Parallel, Distributed, and Network-Based Processing (PDP '02)*, pp. 91-98, Jan. 2002. Also in *Brief Announcements of the 14th Int'l Symp. DIstributed Computing (DISC 2000)*, Oct. 2000.
[19] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous Implementation of Failure Detectors," Technical Report 1484, Institut de Recherche en Informatique et Systémes Aléatoires (IRISA), Sept. 2002.
[20] A. Mostefaoui and M. Raynal, "Unreliable Failure Detectors with Limited Scope Accuracy and an Application to Consensus," *Proc. 19th Int'l Conf. Foundations of Software Technology and Theoretical Computer Science (FST&TCS '99)*, pp. 329-340, Dec. 1999.
[21] A. Mostefaoui and M. Raynal, "*k*-Set Agreement and Limited Accuracy Failure Detectors," *Proc. 19th Ann. ACM Symp. Principles of Distributed Computing (PODC '00)*, pp. 143-152, July 2000.
[22] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *J. ACM*, vol. 27, no. 2, pp. 228-234, Apr. 1980.
[23] M. Raynal and F. Tronel, "Group Membership Failure Detection: A Simple Protocol and Its Probabilistic Analysis," *Distributed Systems Eng. J.*, vol. 6, no. 3, pp. 95-102, 1999.
[24] M. Raynal and F. Tronel, "Restricted Failure Detectors: Definition and Reduction Protocols," *Information Processing Letters*, vol. 72, pp. 91-97, 1999.
[25] J. Yang, G. Neiger, and E. Gafni, "Structured Derivations of Consensus Algorithms for Failure Detectors," *Proc. 17th Ann. ACM Symp. Principles of Distributed Computing (PODC '98)*, pp. 297-308, July 1998.

Mikel Larrea graduated in computer science from the Swiss Federal Institute of Technology in Lausanne, Switzerland, in 1995. He received the PhD degree in computer science from the Universidad del País Vasco in 2000. He is an assistant professor at the Universidad del País Vasco in San Sebastián, Spain, since 2001. Previously, he was on the faculty of the Universidad Pública de Navarra. His research interests include distributed algorithms and systems, fault tolerance, and group communication systems.

Antonio Fernández graduated in computer science from the Universidad Politécnica de Madrid in 1991. He received the PhD degree in computer science from the University of Southwestern Louisiana in 1994 and was a postdoc at the Massachusetts Institute of Technology from 1995 to 1997. He has been an associate professor at the Universidad Rey Juan Carlos in Madrid, Spain, since 1998. Previously, he was on the faculty of the Universidad Politécnica de Madrid. His research interests include data communications, computer networks, parallel and distributed processing, algorithms, and discrete and applied mathematics. He is a senior member of the IEEE.

Sergio Arévalo graduated in computer science from the Universidad Politécnica de Madrid in 1983. He received the PhD degree in computer science from the Universidad Politécnica de Madrid in 1988. He has been a professor at the Universidad Rey Juan Carlos in Madrid, Spain, since 2002. Since 1998, he has been an associate professor at this university. Previously, he was on the faculty of the Universidad Politécnica de Madrid. He was an invited researcher at AT&T Bell Laboratories, Murray Hill, New Jersey, in 1997 and 1988. He was a postdoctoral research fellow in 1989 at the European Space Agency, Nordwijk, The Netherlands. His research interests include distributed languages and systems, fault tolerance, and operating systems. He has been a member of the ACM since 1983.