On the Impossibility of Implementing Perpetual Failure Detectors in Partially Synchronous Systems[⋆]

Mikel Larrea[1] Antonio Fernández[2] Sergio Arévalo[2]

[1] Universidad Pública de Navarra, 31006 Pamplona, Spain
mikel.larrea@unavarra.es
[2] Universidad Rey Juan Carlos, 28933 Móstoles, Spain
afernandez@acm.org, s.arevalo@escet.urjc.es

Abstract. In this paper we study the implementability of different classes of failure detectors in several models of partial synchrony. We show that no failure detector with perpetual accuracy (namely, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$) can be implemented in any of the models of partial synchrony proposed in [2, 4] 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 class Θ.

1 Introduction

The Consensus problem is considered one of the fundamental problems in distributed computing. However, it was shown by Fischer et al. [5] that the Consensus problem cannot be solved deterministically in an asynchronous system in which processes can fail. This result 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 [3, 4].

An alternative and elegant approach to circumvent the unsolvability of Consensus in asynchronous systems was proposed by Chandra and Toueg [2]. 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 [5] and the possibility of solving Consensus using unreliable failure detectors [2], it can be derived 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 Our Results

In this paper we study the possibility of implementing several classes of failure detectors in partially synchronous systems. We start with the eight classes of failure detectors proposed in [2]. There are already algorithms that implement four of them ($\diamond\mathcal{P}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{W}$) in partially synchronous systems [2, 6]. (We call these classes *eventual* classes,

[⋆] Research partially supported by the Spanish Research Council, contracts TIC99-0280-C02-02, TEL99-0582, and TIC98-1032-C03-01, and the Madrid Regional Research Council, contract CAM-07T/00112/1998.

because they have eventual accuracy.) That leaves us with only four more classes to consider ($\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$), which we call *perpetual* classes. We show here 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 [2,4], which means that our results apply to their models of partial synchrony as well.

We complete 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. The class Θ of failure detectors was proposed by Aguilera et al. in [1], where it was shown that it is the weakest failure detector that solves *uniform reliable broadcast*. Since in [1] was also presented an algorithm implementing Θ in an asynchronous system with a majority of correct processes, our result identifies a necessary and sufficient condition for Θ failure detectors to be implemented in partially synchronous systems.

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 show that the perpetual failure detector classes cannot be implemented in our models of partial synchrony. Finally, in Section 4 we show that it is necessary a majority of correct processes to implement a Θ failure detector in our models of partial synchrony.

2 System Model

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 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 of the system we identify two complementary subsets of Π: the subset of processes that do not fail, denoted *correct*, and the subset of processes that do fail, denoted *crashed*. We use 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., $|crashed| \leq f < n$. We also assume that failures are symmetric, i.e., a priori *any* process in the system is allowed to crash.

2.1 Failure Detectors

As we said above, 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 [2], 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.

To be useful, a failure detector must also satisfy some accuracy, which restricts the *mistakes* that it can make. Chandra and Toueg consider the following four accuracy 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.

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.

– *Eventual Weak Accuracy.* There is a time after which some correct process is never suspected by any correct process.

Note that 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.

Combining one of the two completeness properties with one of the four accuracy properties we obtain a *class* of failure detectors. There are eight different classes, which are presented in Figure 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 [2] that any of the failure detectors of Figure 1 can be used to solve Consensus.

We now define the class Θ of failure detectors [1] in terms of completeness and accuracy properties. A failure detector of class Θ must satisfy the following properties. (We say that a process p trusts another process q at a given time t if p does not suspect q at time t.)

– 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.

As we said, Θ is the weakest class of failure detectors that solves *uniform reliable broadcast*, and Aguilera et al. proposed in [1] an algorithm implementing it in an asynchronous system with a majority of correct processes.

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. From the synchronous to the asynchronous models there is a whole range of possible models of synchrony. We call these *partially synchronous models.*

Dwork et al. [4] 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*). Messages sent before *GST* can get lost.

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.

Dwork et al. [4] showed that Consensus can be solved in both models M_1 and M_2 with a majority of correct processes, i.e., when $f < n/2$. They proposed Consensus algorithms for various fault models[2] that work correctly

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

[2] Model M_2 becomes interesting if channels are unreliable before *GST*. In such a case, algorithms must include techniques to mask the loss of messages.

regardless of the actual values of the bounds (in the case of model M_1), and the actual value of GST (in the case of model M_2).

In [2], Chandra and Toueg proposed a weaker model of partial synchrony M_3, which 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 also that every system that conforms to models M_1 or M_2 also conforms to model M_3.

Chandra and Toueg showed how to implement a failure detector of class $\diamond\mathcal{P}$ in a system that conforms to model M_3. This shows that the four classes of eventual failure detectors can be implemented in such a system model, since a $\diamond\mathcal{P}$ failure detector also belongs to classes $\diamond\mathcal{W}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{Q}$. 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 impossible to implement in models of partial synchrony.

In this paper, we prove the impossibility of implementing such perpetual classes of failure detectors in partially synchronous models of computation. 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.

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

There is a simple proof that any implementation in 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 M_3) requires $f < n/2$.

The proof, which uses contradiction, goes as follows. It is shown in [4] that the smallest number of processes for which an r-resilient Consensus protocol exists in the model of partial synchrony M_1 is $2r+1$. In other words, any protocol that solves Consensus in model M_1 requires a majority of correct processes.

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 [2], 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 Impossibility of Implementing Perpetual Failure Detectors

In this section, we show that none of the perpetual failure detector classes ($\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$, and $\mathcal{W}$) can be implemented in our models of partial synchrony. From the relationship between failure detector classes described in [2], it is 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 simplicity, we show the result for the failure detector class $\mathcal{S}$, and the result applies directly to $\mathcal{W}$ since both classes are equivalent [2]. The approach followed is assuming the existence of

[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 delay 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.

a failure detector satisfying the strong completeness property, and showing that the perpetual weak 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, perpetual weak accuracy is violated, and thus the failure detector does not implement any of the four perpetual failure detector classes defined in [2]. 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.

Following the previous approach, we show an admissible run of the system in which all the correct processes are erroneously suspected at least once, thus violating perpetual weak accuracy[5]. 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 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, $i = 1,\dots,n$ of Σ as follows. In run R_i only process p_i crashes, doing it at time t_{i-1}, defined in R_{i-1}, and for each process p_k, $k = 1,\dots,i-1$, 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, $k = 1,\dots,i-1$, all processes except p_k behave exactly like in run R_k up to time t_k. Clearly, for each process p_k, $k = 1,\dots,i-1$, 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, $i = 1,\dots,n$, all processes except p_i behave exactly like in run R_i up to time t_i.

Clearly, for each process p_i, $i = 1,\dots,n$, 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, $i = 1,\dots,n$ 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 1. *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 2. *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 1 and the fact that $\mathcal{S}$ and $\mathcal{W}$ are equivalent [2]. ■

[5] Note that an algorithm implementing any given class $\mathcal{D}$ of failure detectors must satisfy the properties that characterize $\mathcal{D}$ in *all admissible* runs.

Corollary 3. *There is no protocol that implements a failure detector of classes $\mathcal{P}$ or $\mathcal{Q}$ 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 $\mathcal{P}$ and $\mathcal{Q}$ are stronger than $\mathcal{W}$ [2]. ∎

4 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 [1], cannot be implemented in the models of partial synchrony M_1 and M_2 without a majority of correct processes.

Theorem 2. *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.*

Acknowledgments. We are grateful to André Schiper for his valuable comments on earlier drafts of this paper. We also thank Sam Toueg for suggesting the indirect proof of impossibility presented in Section 2.3.

References

1. M. Aguilera, S. Toueg, and B. Deianov. Revisiting the weakest failure detector for uniform reliable broadcast. In *Proceedings of the 13th International Symposium on DIstributed Computing (DISC, formerly WDAG)*, pages 19–33. LNCS, Springer-Verlag, September 1999.
2. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
3. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–98, January 1987.
4. C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
5. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
6. M. Larrea, S. Arévalo, and 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)*, pages 34–48. LNCS, Springer-Verlag, September 1999.