A Realistic Look At Failure Detectors

C. Delporte-Gallet †, H. Fauconnier †, R. Guerraoui ‡
†Laboratoire d'Informatique Algorithmique: Fondements et Applications,
Université Paris VII - Denis Diderot
‡Distributed Programming Laboratory,
Swiss Federal Institute of Technology in Lausanne

Abstract

This paper shows that, in an environment where we do not bound the number of faulty processes, the class $\mathcal{P}$ of Perfect failure detectors is the weakest (among realistic *failure detectors) to solve fundamental agreement problems like uniform consensus, atomic broadcast, and terminating reliable broadcast (also called Byzantine Generals).*

Roughly speaking, in this environment, we collapse the Chandra-Toueg failure detector hierarchy, by showing that $\mathcal{P}$ ends up being the only class to solve those agreement problems. This contributes in explaining why most reliable distributed systems we know of do rely on some group membership service that precisely aims at emulating $\mathcal{P}$.

As an interesting side effect of our work, we show that, in our general environment, uniform consensus is strictly harder than consensus, and we revisit the view that uniform consensus and atomic broadcast are strictly weaker than terminating reliable broadcast.

1 Introduction

1.1 Motivation

It is well known that agreement is at the heart of reliable distributed computing, but is rather difficult to achieve in a failure-prone environment. In particular, without any *synchrony assumptions* (i.e., assumptions on process relative speeds and communication delays), agreement is impossible even among a set of distributed processes that communicate through reliable channels and at most one of them might fail (and it can do so only by crashing) [5]. In fact, synchrony assumptions are needed to provide processes with *information about failures*, and this information is the key to reaching agreement in the presence of failures. The motivation of our work is to determine the *exact* information about failures needed to achieve agreement in an environment where we do not bound the number of failures (we focus here on process crash-failures). Roughly speaking, determining that information comes down to providing an abstract metric that helps measure whether a set of synchrony assumptions are necessary and sufficient to reach agreement [3].

We consider in this paper two fundamental agreement problems: *consensus* and *terminating reliable broadcast.* In the *consensus* problem, processes need to decide on a common value among one of the proposed values.[1] Solving this problem is known to be equivalent to solving the *atomic broadcast* problem [1], in any system where only a finite number of messages can be lost, e.g., with reliable channels. This problem consists in delivering messages to processes in a reliable and totally ordered manner. Solving this problem is a key to building highly available and consistent replicated services. *Terminating reliable broadcast* is a strong and convenient form of reliable broadcast: the processes should deliver the same sequence of messages, just like with reliable broadcast but, in addition, should deliver a specific *nil* value for every message that was broadcast by a faulty process and not delivered by any correct process [11]. This problem is a rephrasing, in the crash-stop model, of the famous Byzantine Generals problem [13].

[1] By default, we consider the *uniform* variant of the consensus problem, which precludes any disagreement among two processes, even if one of them ends up being faulty [10]. In Section 6, we discuss the impact of going back to the *correct-restricted* variant of consensus, which is solely of theoretical interest.

1.2 Background: the failure detector hierarchy

In a seminal paper [1], Chandra and Toueg proposed a precise way to measure the information about failures needed to solve agreement problems within the abstraction of a *failure detector*. A failure detector is represented by a distributed oracle that provides processes with hints about process failures, and this oracle can be viewed as an abstraction hiding lower level synchrony assumptions such as message communication delays and process relative speeds, i.e., assumptions that underly any useful form of information about process failures.

Chandra and Toueg established a hierarchy of failure detector classes. Basically, a class gathers a set of failure detectors that capture the same information about failures. In short, a class A is *stronger* than a class B in the hierarchy if the information about failures captured by A encompasses the information about failures captured by B. In other words, the synchrony assumptions underlying A are stronger than those underlying B. In particular, three interesting classes were identified: the class $\diamond\mathcal{S}$, of *Eventually Strong* failure detectors, the class $\mathcal{S}$ of *Strong* failure detectors, and the class $\mathcal{P}$ of *Perfect* failure detectors. Among these classes, $\mathcal{P}$ is the strongest whereas $\diamond\mathcal{S}$ is the weakest. The following results were proved [1]:

- Any failure detector of class $\diamond\mathcal{S}$ solves consensus (and hence atomic broadcast) if a majority of processes are correct.
- Any failure detector of class $\mathcal{S}$ solves consensus (and hence atomic broadcast) even if the number of faulty processes is not bounded. [2]
- Any failure detector of class $\mathcal{P}$ solves terminating reliable broadcast even if the number of faulty processes is not bounded.

Interestingly, each of these results conveys the fact that a certain information about failures is *sufficient* to solve some agreement problem (possibly under a specific assumption on the maximum number of faulty processes [1]). A natural question follows: is that information about failures also *necessary*? Together with Hadzilacos, Chandra and Toueg addressed the question for the case of consensus with a majority of correct processes. They proved that $\diamond\mathcal{S}$ is actually the weakest for consensus if a majority of processes are correct [2]. In a precise sense, this goes through proving that there is an algorithm A that transforms any failure detector $\mathcal{D}$ that solves consensus into a failure detector of class $\diamond\mathcal{S}$. In short, the very existence of A means that $\mathcal{D}$ provides at least as much information about failures as $\diamond\mathcal{S}$: this information is hence minimal. Chandra and Toueg also pointed out the very fact that if we do not bound the number of failures, $\diamond\mathcal{S}$ is neither sufficient for consensus nor for terminating reliable broadcast.

1.3 Contributions

In an environment where we do not restrict the number of possible failures, determining the weakest failure detector classes for problems like consensus (thus for atomic broadcast) and terminating reliable broadcast have been open for almost a decade now. We show here that there is one answer to these questions: $\mathcal{P}$. More precisely, if any number of processes can fail, $\mathcal{P}$ is the weakest failure detector class to solve consensus (hence atomic broadcast) and terminating reliable broadcast.

We state and prove our result using simple algorithm reductions (as in [2]) and following the original failure detector formalism of [1], with one exception however. We exclude from the original failure detector space of [1], failure detectors that can *guess the future* (these cannot be implemented even in a perfectly synchronous system), and we focus only on realistic failures detectors that indeed encapsulate synchrony assumptions.

At first glance, our result seems to introduce a contradiction. As we recalled above, it was shown in [1] that $\mathcal{S}$ solves consensus even if we do not restrict the number of faulty processes, and $\mathcal{S}$ is strictly weaker than $\mathcal{P}$. How can $\mathcal{P}$ be the weakest? Interestingly, and as we show in the paper, within the space of realistic failure detectors, $\mathcal{S}$ and $\mathcal{P}$ have the same computational power. As observed in [12], this means that the difference between these classes is rather artificial in our general environment.

To summarise, our paper shows that, in an environment where we do not bound the number of faulty processes, the exact information about failures needed to solve consensus (hence atomic broadcast) and terminating reliable broadcast is captured by $\mathcal{P}$. In short, we collapse the failure detector hierarchy: $\mathcal{P}$ ends up being the only useful class in the hierarchy to solve agreement problems. A posteriori, this is not that surprising and our results might explain why developers of reliable distributed systems have been considering, as a basic building block [14], a group membership

[2] In fact, the actual definition of $\mathcal{S}$ assumes that at least one process does not crash but the definition can easily be adapted to the more general case where any number of processes can crash.

service, which precisely aims at emulating a *Perfect* failure detector, i.e., when a process is suspected, i.e., timed-out, it is excluded from the group: every suspicion hence turns out to be accurate [4, 6, 16].
As a side effect of our work, we point out two interesting results in our general environment. First, if we consider the correct-restricted variant of consensus (i.e., non-uniform consensus), $\mathcal{P}$ is clearly not the weakest. A simple corollary of this observation is that (uniform) consensus is strictly harder than the correct-restricted variant of consensus. Second, we also revisit the view that consensus and atomic broadcast are strictly weaker problems than terminating reliable broadcast.

2 System model

Our model of asynchronous computation with failure detection is the FLP model [5] augmented with the failure detector abstraction [1, 2]. A discrete global clock is assumed, and Φ, the range of the clock's ticks, is the set of natural numbers. The global clock is used for presentation simplicity and is not accessible to the processes. We sketch here the fundamentals of the model. The reader interested in specific details about the model should consult [2].

2.1 Failure patterns and environments

We consider a distributed system composed of a finite set of n processes $\Omega = \{p_1, p_2, \dots, p_n\}$ ($|\Omega| = n > 3$). A process p_i is said to *crash at time* t if p_i does not perform any *action* after time t (the notion of *action* is recalled below). Failures are permanent, i.e., no process *recovers* after a crash. A *correct* process is a process that does not crash. A *failure pattern* is a function F from Φ to 2^Ω, where $F(t)$ denotes the set of processes that have crashed through time t. The set of correct processes in a failure pattern F is noted $correct(F)$.
An *environment* E is a set of failure patterns. Environments describe the crashes that can occur in a system. In this paper, we consider the environment that contains all possible failure patterns. That is, we do not bound the number of processes that can crash.

2.2 Failure detectors

Roughly speaking, a failure detector $\mathcal{D}$ is a distributed oracle which gives hints about failure patterns. Each process p_i has a local failure detector module of $\mathcal{D}$, denoted by $\mathcal{D}_i$. Associated with each failure detector $\mathcal{D}$ is a range $R_\mathcal{D}$[3] of values output by the failure detector. A *failure detector history* H with range R is a function H from $\Omega \times \Phi$ to R. For every process $p_i \in \Omega$, for every time $t \in \Phi$, $H(p_i, t)$ denotes the value of the failure detector module of process p_i at time t, i.e., $H(p_i, t)$ denotes the value output by $\mathcal{D}_i$ at time t. A *failure detector* $\mathcal{D}$ is a function that maps each failure pattern F to a set of failure detector histories with range $R_\mathcal{D}$. $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted for the failure pattern F, i.e., each history represents a possible behaviour of $\mathcal{D}$ for the failure pattern F. The failure detectors introduced in [1] do all have a range $R = 2^\Omega$. For any such failure detector $\mathcal{D}$, any failure pattern F and any history H in $\mathcal{D}(F)$, $H(p_i, t)$ is the set of processes *suspected* by process p_i at time t.

2.3 Algorithms

An *algorithm* using a failure detector $\mathcal{D}$ is a collection A of n deterministic automata A_i (one per process p_i). Computation proceeds in steps of the algorithm. In each step of an algorithm A, a process p_i atomically performs the following three actions: (1) p_i receives a message from some process p_j, or a "null" message λ; (2) p_i queries and receives a value d from its failure detector module $\mathcal{D}_i$ ($d \in R_\mathcal{D}$ is said to be *seen* by p_i); (3) p_i changes its state and sends a message (possibly null) to some process. This third action is performed according to (a) the automaton A_i, (b) the state of p_i at the beginning of the step, (c) the message received in action 1, and (d) the value d seen by p_i in action 2. The message received by a process is chosen non-deterministically among the messages in the message buffer destined to p_i, and the null message λ. A *configuration* is a pair (I, M) where I is a function mapping each process p_i to its local state, and M is a set of messages currently in the message buffer. A configuration (I, M) is an initial configuration if $M = \emptyset$ (no message is initially in the buffer): in this case, the states to which I maps the processes are called *initial states*. A *step* of an algorithm A is a tuple $e = (p_i, m, d, A)$, uniquely defined by the algorithm A, the identity of the process p_i that takes the step, the message m received by p_i, and the failure detector value d seen by p_i during the step. A step $e = (p_i, m, d, A)$ is *applicable to a configuration* (I, M) if and only if $m \in M \cup \{\lambda\}$. The *unique* configuration that results from applying e to configuration $C = (I, M)$ is noted $e(C)$.

[3]When the context is clear we omit the subscript.

2.4 Schedules and runs

A *schedule* of an algorithm A is a (possibly infinite) sequence $S = S[1]; S[2]; \ldots S[k]; \ldots$ of steps of A. A schedule S is applicable to a configuration C if (1) S is the empty schedule, or (2) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$ (the configuration obtained from applying $S[1]$ to C), etc.
Let A be any algorithm and $\mathcal{D}$ any failure detector. A *partial run of* A *using* $\mathcal{D}$ is a tuple $R =< F, H, C, S, T >$ where H is a failure detector history such that $H \in \mathcal{D}(F)$, C is an initial configuration of A, T is a finite sequence of increasing time values, and S is a finite schedule of A such that, (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $k \leq |S|$ where $S[k] = (p_i, m, d, A)$, we have $p_i \notin F(T[k])$ and $d = H(p_i, T[k])$.
A *run of* of A *using* $\mathcal{D}$ is a tuple $R =< F, H, C, S, T >$ where H is a failure detector history and $H \in \mathcal{D}(F)$, C is an initial configuration of A, S is an infinite schedule of A, T is an infinite sequence of increasing time values, and in addition to the conditions above of a partial run ((1), (2) and (3)), the two following conditions are satisfied: (4) every correct process takes an infinite number of steps, and (5) every message sent to a correct process p_j is eventually received by p_j.

2.5 Solvability

An algorithm A *solves* a problem B using a failure detector $\mathcal{D}$ if every run of A using $\mathcal{D}$ satisfies the specification of B. We say that $\mathcal{D}$ solves B if there is an algorithm that solves B using $\mathcal{D}$. We say that a failure detector $\mathcal{D}1$ is *stronger* than a failure detector $\mathcal{D}2$ ($\mathcal{D}2 \preceq \mathcal{D}1$) if there is an algorithm that transforms $\mathcal{D}1$ into $\mathcal{D}2$, i.e., that can *emulate* $\mathcal{D}2$ with $\mathcal{D}1$ [1]. The algorithm does not need to emulate all histories of $\mathcal{D}2$. It is required however that for every run $R =< F, H, C, S, T >$ where $H \in \mathcal{D}1(F)$, the output of the algorithm with R is a history of $\mathcal{D}2(F)$. We say that $\mathcal{D}1$ is *strictly stronger* than $\mathcal{D}2$ ($\mathcal{D}2 \prec \mathcal{D}1$) if $\mathcal{D}2 \preceq \mathcal{D}1$ and $\mathcal{D}1 \not\preceq \mathcal{D}2$. Finally, we say that a failure detector $\mathcal{D}$ is the weakest to solve a problem B if (a) $\mathcal{D}$ solves B and (b) any failure detector that solves B is stronger than $\mathcal{D}$.

3 Realistic failure detectors

Stating that failure detector class $\mathcal{D}$ is the weakest to solve a problem X hides an implicit assumption: the set of failure detectors among which $\mathcal{D}$ is the weakest. Without precisely defining that set, the statement is simply meaningless. In [2], $\diamond\mathcal{S}$ is shown to be the weakest class for consensus (with a majority of correct processes) among all possible failure detectors that comply with the original definition of a failure detector in [1]. According to that definition (recalled in Section 2.2), a failure detector is precisely defined as a function of the failure pattern. Any function of the failure pattern is a failure detector, including a function that provides information about *future* failures. Such a failure detector does not really factor out synchrony assumptions of the system: it cannot be implemented even in a perfectly synchronous system - remember that the motivation of introducing failure detectors was basically to factor out synchrony assumptions within an abstract formalism.
In this paper, we restrict our space to failure detectors as functions of the "past" failure pattern. In the following, we first define the class $\mathcal{R}$ of *realistic* failure detectors (those that cannot guess the future), which include among others, *Eventually Perfect* and *Strong* failure detectors. In other words, class $\mathcal{R}$ intersects with both classes $\mathcal{S}$ and $\diamond\mathcal{P}$. We then illustrate this notion through two simple examples.

3.1 Definition

Roughly speaking, we say that a failure detector is *realistic* if it cannot guess the future. In other words, there is no time t and no failure pattern F at which the failure detector can provide exact information about crashes that will hold after t in F. More precisely, we define the class of realistic failure detector $\mathcal{R}$, as the set of failure detectors $\mathcal{D}$ that satisfy the following property:

- $\forall (F, F') \in E \ \forall t \in \Phi$ s.t. $\forall t_1 \leq t, F(t_1) = F'(t_1)$, we have:
  - $\forall H \in \mathcal{D}(F), \exists H' \in \mathcal{D}(F')$ s.t.: $\forall t_1 \leq t, \forall p_i \in \Omega : H(p_i, t_1) = H'(p_i, t_1)$.

Basically, a failure detector $\mathcal{D}$ is *realistic* if for any pair of failure patterns F and F' that are similar up to a given time t, whenever $\mathcal{D}$ outputs some information at a time $t - k$ in F, $\mathcal{D}$ could output the very same information at $t - k$ in F'. In other words, a realistic failure detector cannot distinguish two failure patterns according to what will happen in the future.
Note that if a failure detector $\mathcal{D}$ is realistic, then, for any failure pattern F, the output of $\mathcal{D}$ at time t is a *function* of F up to time t.

3.2 Examples

We illustrate below our notion through two failure detector examples: a realistic and a non-realistic one.

3.2.1 The Scribe

We describe here the *Scribe* failure detector $\mathcal{C}$. This failure detector outputs a list of processes. In short, failure detector $\mathcal{C}$ sees what happens at all processes at real time and takes notes of what it sees. More precisely, in any failure pattern F, failure detector $\mathcal{C}$ outputs, at any time t, the list of values of F up to time t: we denote this list by $F[t]$. (Remember that a failure pattern is a function that associates to every positive integer, representing time, a subset of the processes in the system Ω). More precisely, for each failure pattern F, $\mathcal{C}(F)$ is the singleton that contains the failure detector history H such that:

- $\forall t \in \Phi, \forall p_i \in \Omega,\ H(p_i, t) = F[t]$.

It is obvious to see that failure detector $\mathcal{C}$ is realistic: it actually belongs to $\mathcal{P}$.

3.2.2 The Marabout

Consider failure detector $\mathcal{M}$ (*Marabout*), defined in [9]. This failure detector outputs a list of processes. For any failure pattern F and at any process p_i, the output of the failure detector $\mathcal{M}$ is constant: it is the list of faulty processes in F, i.e., $\mathcal{M}$ outputs the list of processes that *have* crashed or *will* crash in F. Failure detector $\mathcal{M}$ belongs both to class $\diamond\mathcal{P}$ and $\mathcal{S}$ of [1].
Clearly, $\mathcal{M}$ is not realistic. To see why, consider failure patterns F and F' such that:

1. In F_1, all processes are correct, except p_1 which crashes at time 10.
2. In F_2, all processes are correct.

Consider H_2, any history in $\mathcal{M}(F_2)$. By the definition of $\mathcal{M}$, the output at any process and any time of H_2 is $\emptyset$. Consider time $T = 9$. Up to this time, F_1 and F_2 are the same. If $\mathcal{M}$ was realistic, $\mathcal{M}$ would have had a failure detector history H_1 in $\mathcal{M}(F_1)$ such that H_2 and H_1 are the same (at any process) up to time 9. This is clearly impossible since for any history $H_1 \in \mathcal{M}(F_1)$, for any process p_i, and any time $t \in \Phi$, $H_1(p_i, t) = \{p_1\}$. As observed in [9], the class $\mathcal{M}$ and the class $\mathcal{P}$ are incomparable. In short, $\mathcal{M}$ is accurate about the future whereas $\mathcal{P}$ is accurate about the past.

In the following, we restrict ourselves to algorithms using *realistic failure detectors*. We shall come back to this in Section 6.

4 The weakest failure detector for consensus

In the consensus problem, the processes propose an initial value and must agree on one of these values. The following properties must be satisfied: *1. termination.* every correct process eventually decides; *2. agreement.* no two processes decide differently; *3. validity.* the value decided must have been proposed by some process.
We show here that if we do not restrict the number of faulty processes, the *weakest* failure detector class (among realistic ones) to solve consensus is $\mathcal{P}$. Obviously, any failure detector of class $\mathcal{P}$ solves consensus no matter how many processes may fail. In other words, we show here that if we do not restrict the number of faulty processes, any realistic failure detector that solves consensus can be transformed into a failure detector of class $\mathcal{P}$. We first give an intuition of this lower bound proof and then we give the proof itself.

4.1 Intuitions

We prove our lower bound result in two steps: we show that (a) any consensus algorithm is *total*: the causal chain of any decision event contains a message from every process that has not crashed at the time of the decision; and then (b) if a failure detector $\mathcal{D}$ implements a total consensus algorithm, then $\mathcal{D}$ can be transformed into a *Perfect* failure detector.

- (a) The first part of the proof (i.e., in the first lemma below) uses the fact that we do not restrict the number of faulty processes. Intuitively, we show here that no process can reach a consensus decision without having "consulted" every correct process. This is to prevent the case where, after the decision, all processes crash except the process that was not "consulted" and this process decides later differently. If all processes that have not crashed are consulted before every decision, we say that the algorithm is *total*. (Our notion of *total* is a generalisation of the notion of *total* initially defined in [15] for the failure-free case.)[4]
- (b) In the second part of the proof (i.e., in the second lemma below), we use the fact that $\mathcal{D}$

[4] Typically, algorithms like the consensus algorithm of [1] based on $\diamond\mathcal{S}$ is not total because only a majority needs to be consulted, even if all processes are correct. On the contrary, the $\mathcal{S}$-based consensus algorithm of [1] would be total with a realistic failure detector.

is realistic. We show that if a realistic failure detector $\mathcal{D}$ implements a total consensus algorithm, then $\mathcal{D}$ can be transformed into a *Perfect* failure detector. Roughly speaking, we use the fact that the algorithm is total, and hence no decision is taken without "consulting" every correct process, to accurately track process failures. We run a sequence of consensus instances and we suspect a process to have crashed if and only if a decision is reached and the process was not consulted in the decision.

4.2 Total consensus

Let A be any algorithm that solves consensus. We call *decision events* in A, the events by which processes decide a consensus value. We say that A is total if any decision event in A at time t contains, in its causal chain [11], a message sent by every process that has not crashed by time t.

Lemma 4.1 *Consider the environment where we do not bound the number of processes that can crash. Every consensus algorithm using a realistic failure detector in this environment is total.*

Proof (sketch): Assume by contradiction that there is a consensus algorithm A that is not total. This means that there is a run R_0 of A such that, in R_0, some process p_i has a decision event e executed at some time t (e is the event by which p_i decides some value v), and a process p_j that has not crashed by t, such that no message from p_j is in the causal chain of e. Assume without loss of generality that the decision v is 0. As there is no message from p_j in the causal past of e, we can assume that the value proposed by p_j is 1. Now consider the following runs:

- R_1: R_1 is similar to R_0, except that p_j does not receive any message from any other process before time t, i.e., we delay in R_1 the reception of all messages by p_j. Moreover, no process p_k, $k \neq i, j$, takes any step after its last step in the causal past of e, until time t. Since p_j does not participate in the decision e of R_0, then p_i executes event e in R_1 and also decides 0 at some time t (as in R_0).
- R_2: in R_2, the failure pattern is the same as in R_1 until time t, and all processes crash at time t, except p_j which is correct. Moreover, no process take steps until time t. By the termination property of consensus, p_j decides at some time t' in R_2, and by the validity property of consensus, p_j actually decides 1 at t'.
- R_3: the failure pattern of R_3 is exactly the same as in R_2, but, until time t, all processes are scheduled exactly as in R_1 and all messages between are sent and received as in R_1. Moreover, p_j is scheduled as in R_2 and all messages from and to p_j are delayed after time t_2.

As the failure detector is realistic, it can behave in R_3 as in R_1 until time t. In this way, p_i behaves in R_3 as in R_1 and decides 0. But, p_j behaves in R_3 as in R_2 and p_j decides 1: contradicting the agreement property of consensus. □

4.3 Reduction

Let A be any total consensus algorithm using $\mathcal{D}$. We build a transformation algorithm $T_{\mathcal{D}\Rightarrow\mathcal{P}}$, that emulates the behaviour of a *Perfect* failure detector within a variable denoted by $output(\mathcal{P})$. This variable is distributed and every process p_i has a copy of this variable denoted by $output(\mathcal{P})_i$. Algorithm $T_{\mathcal{D}\Rightarrow\mathcal{P}}$ consists in an infinite sequence of executions of A (i.e., a sequence of total consensus instances) plus the following additions:

1. Whenever p_i sends a message m, p_i attaches to m the information [p_i is alive].
2. Whenever a process p_j receives a message m from a process p_i, p_j extracts from m any information [p_k is alive] and attaches that information to every event executed as a consequence of the reception of m.
3. Whenever a process p_j executes a decision event e (i.e., p_j decides some value), p_j adds to $output(\mathcal{P})_j$ every process p_i such that [p_i is alive] is not attached to e.

Lemma 4.2 *The algorithm $T_{\mathcal{D}\Rightarrow\mathcal{P}}$ emulates in $output(\mathcal{P})$ the behaviour of a Perfect failure detector.*

Proof: We prove that the failure detector emulated in $output(\mathcal{P})$ ensures strong completeness and strong accuracy. Consider first completeness. Let p_i be any process that crashes and p_j a correct process. There is a time after which p_i does not send any message. By the termination property of consensus, p_j eventually decides in that execution, i.e., by executing some decision event e. Given that the information [p_i is alive] is not attached to e, p_j adds to $output(\mathcal{P})_j$ p_i and never removes it, i.e., p_j permanently suspects p_i. Strong completeness is thus ensured. Consider now accuracy. Assume that p_j suspects p_i, i.e., p_j adds p_i to *output*$(\mathcal{P})_j$. This can only happen if p_j executes a decision event e in some execution of A, and the information [p_i is alive] is not attached to e. Given that A is a total algorithm, this can only happen if p_i has crashed. Strong accuracy is hence ensured too. □

Proposition 4.3 *Consider the environment where we do not bound the number of processes that can crash. In this environment, among realistic failure detectors, the weakest class for consensus is $\mathcal{P}$.*

PROOF: (*1. Sufficient condition.*) In [1], Chandra and Toueg presented a $\mathcal{S}$-based algorithm that solves consensus. Obviously, the algorithm works with any failure detector in both $\mathcal{R}$ and $\mathcal{P}$ even if we do not bound the number of faulty processes. (*2. Necessary condition.*) By Lemma 4.1 and Lemma 4.2, any failure detector in $\mathcal{R}$ that solves consensus can be transformed into a failure detector in $\mathcal{P}$. □

5 The weakest failure detector for terminating reliable broadcast

We consider here a strong form of reliable broadcast called *terminating reliable broadcast* [11]. In this problem, processes must deliver a specific value *nil* if the sender process has crashed [11]: otherwise, the processes must deliver the message m broadcast by *sender*(m). We actually consider a general variant of the problem where every process is a potential initiator of the broadcast. We denote by (i, k) the k'th instance of the problem where the initiator of the broadcast is p_i. Instance $(i, *)$ is defined with the following properties: (1) *validity* if a correct process p_i broadcasts a message m, then p_i eventually delivers m, (2) *agreement* if a process delivers a message m, then every correct process delivers m; and (3) *integrity* if a process delivers a message m and p_i is correct, then *sender*(m) $= p_i$.
We state and show here that if we do not restrict the number of faulty processes and consider realistic failure detectors, the *weakest* failure detector class to solve terminating reliable broadcast is $\mathcal{P}$. This result was already stated in [7] and proved implicitly assuming realistic failure detectors. The proof below makes that assumption explicit.

Proposition 5.1 *Consider the environment where we do not bound the number of processes that can crash. In this environment, among realistic failure detectors, the weakest class for for terminating reliable broadcast is $\mathcal{P}$.*

PROOF (SKETCH): (*1. Sufficient condition.*) It is easy to see that any *Perfect* failure detector, including realistic failure detectors, solves the terminating reliable broadcast problem. When executing instance (k, k') of the problem, each process waits until it receives the value from p_k or it suspects p_k. In the first case it proposes this value to a consensus else it proposes *nil*. The value delivered is the consensus value. (*2. Necessary condition.*) Let A be any terminating reliable broadcast algorithm using $\mathcal{D}$. It is easy to see how we can emulate the output of $\mathcal{D}$ a failure detector of class $\mathcal{P}$ in a distributed variable *output*$(\mathcal{P})$. Whenever a process p_j delivers *nil* for an instance $(i, *)$ of the problem, p_j adds p_i to *output*$(\mathcal{P})_j$. Any process that crashes will eventually be permanently added to *output*$(\mathcal{P})$ at every correct process: strong completeness will hence be ensured. Let p_i be any process that is added to *output*$(\mathcal{P})_j$ at some time t. This can only be possible if p_i is faulty. Since we assume here that $\mathcal{D}$ is realistic, then p_i must have crashed by time t. □

6 Concluding Remarks

6.1 The impact of realism

As we pointed out in the introduction, our lower bound results do not hold if we also consider failure detectors that can guess the future. To see why, consider the class $\mathcal{M}$ of *Marabout* failure detectors introduced in [9], and recalled in Section 3.
There is an obvious algorithm A that solves consensus using $\mathcal{M}$ even if we do not restrict the number of faulty processes. Every process p_i consults its failure detector and selects the process p_j such that (a) p_j is not suspected and (b) there is no process p_k such that (b.1) $k < j$ and (b.2) p_k is not suspected. If $i = j$, then p_j sends its value to all and decides it. Otherwise, p_j waits for p_j's proposed value and decides that value. Similarly, there is an obvious algorithm that solves terminating reliable broadcast using $\mathcal{M}$ [9].

6.2 Consensus vs uniform consensus

We considered in this paper the uniform variant of the consensus problem. In this variant, no two processes should decide differently, whether they are correct or not [10]. Does our result apply to the *correct-restricted* variant of consensus where agreement is only required among correct processes? Addressing this question is appealing from a theoretical point of view (even if this form of consensus is rather meaningless in practice). The answer is "no".

Even if we consider only realistic failure detectors and we do not bound the number of failures, the weakest failure detector class for consensus is not $\mathcal{P}$. There is an algorithm given in [8] that solves correct-restricted consensus with a strictly weaker class, which we denote by $\mathcal{P}_<$ (the class of *Partially Perfect* failure detectors). Class $\mathcal{P}_<$ is defined through the strong accuracy property of $\mathcal{P}$ and the following *partial* completeness property: If a process p_i crashes, then eventually every correct process p_j such that $j > i$ permanently suspects p_i. It is easy to see that if we do not restrict the number of faulty processes, $\mathcal{P}_<$ is strictly weaker than $\mathcal{P}$: roughly speaking, this is because a process p_i has no knowledge about any process p_j such that $j > i$. Interestingly, this actually means that uniform consensus is strictly harder than consensus.

6.3 Strength vs perfection

It was shown in [1] that the class $\mathcal{S}$ of *Strong* failure detectors solves (uniform) consensus even if we do not restrict the number of faulty processes. This might seem to contradict our result because $\mathcal{S}$ is strictly weaker than $\mathcal{P}$ in the original model of [1]. In fact, if we consider only realistic failure detectors, the classes $\mathcal{S}$ and $\mathcal{P}$ collapse. That is, $\mathcal{S} \cap \mathcal{R} \subset \mathcal{P}$. To see why, assume by contradiction that $\mathcal{D}$ is a realistic failure detector that is *Strong* but not *Perfect*. This means that $\mathcal{D}$ violates strong accuracy: some process p_i is falsely suspected. Because $\mathcal{D}$ is realistic, it cannot guess the future and hence it can very much be the case that all processes but p_i crashes. Weak accuracy would also be violated: a contradiction.

It is important to notice that the observation above has already been made by Halpern and Ricciardi in [12]. In fact, they have expressed our notion of *realism* as a desirable property, among other desirable properties, of a failure detector model using knowledge theory and they proved that in this "new" model, *Strong* failure detectors turn out to be *Perfect*. Our definition of realism can be viewed as a simpler rephrasing of a similar concept introduced in [12]. (It is simpler because we do not introduce any knowledge theory construct and we stick to the original formalism of [1].)

Acknowledgements

We are very grateful to Michel Raynal for his comments on earlier versions of this paper.

References

[1] T. Chandra and S. Toueg. *Unreliable Failure Detectors for Reliable Distributed Systems.* Journal of the ACM, 43(2), March 1996.

[2] T. Chandra, V. Hadzilacos and S. Toueg. *The Weakest Failure Detector for Solving Consensus.* Journal of the ACM, 43(4), July 1996.

[3] C. Dwork, N. Lynch and L. Stockmeyer. *Consensus in the presence of partial synchrony.* Journal of the ACM, 35 (2), 1988.

[4] C. Fetzer and F. Cristian. *Fail-aware failure detectors.* Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems, Niagara-on-the-Lake, Canada, Oct 1996.

[5] M. Fischer, N. Lynch and M. Paterson. *Impossibility of distributed consensus with one faulty process.* Journal of the ACM, 32(2), 1985.

[6] F. Greve, M. Hurfin, M. Raynal and F. Tronel. *Primary Component Asynchronous Group Membership as an Instance of a Generic Agreement Framework.* Proceedings of the IEEE International Symposium on Autonomous Decentralized Systems (ISADS), 2001.

[7] E. Fromentin, M. Raynal, and F. Tronel. *About Classes of Problems in Asynchronous Distributed Systems with Process Crashes.* Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), 1999.

[8] R. Guerraoui. *Revisiting the relationship between the atomic commitment and consensus problems.* Proceedings of the Workshop on Distributed Algorithms, Springer Verlag (LNCS 972), 1995.

[9] R. Guerraoui. *On the Hardness of Failure Sensitive Agreement Problems.* Information Processing Letters, 79, 2001.

[10] V. Hadzilacos. *On the relationship between the atomic commitment and consensus problems.* Proceedings of the Workshop on Fault-Tolerant Distributed Computing, Springer Verlag (LNCS 448), 1986.

[11] V. Hadzilacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems.* Cornell University, Technical Report (TR 94-1425), 1994. Also in *Distributed Systems*, S. Mullender (ed), Addison-Wesley, 1993.

[12] J. Halpern and A. Ricciardi. *A Knowledge-Theoretic Analysis of Uniform Distributed Coordination and Failure Detectors.* Proceedings of the ACM Symposium on Principles of Distributed Computing, 1999.

[13] L. Lamport, M. Pease and R. Shostak. *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems 4 (3), July 1982.

[14] D. Powell (editor). *Special issue on Group Communications.* Communications of the ACM, 39 (4), 1996.

[15] G. Tel. *Topics in Distributed Algorithms.* Cambridge International Series, 1991.

[16] P. Verissimo, A. Casimiro and C. Fetzer. *The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness.* Proceedings of the IEEE International Symposium on Dependable Systems and Networks (DSN), 2000.