On the Weakest Failure Detector for Non-Blocking Atomic Commit *

Rachid Guerraoui Petr Kouznetsov
Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne

Abstract
This paper contributes to the analysis of the Non-Blocking Atomic Commit (NBAC) problem in an asynchronous model with failure detectors. In particular, we address the question of the weakest failure detector to solve NBAC in this model. We define the set $\mathcal{A}$ of *timeless* failure detectors which excludes failure detectors that provide information about global time but includes most known meaningful failure detectors such as $\diamond\mathcal{S}$, $\diamond\mathcal{P}$ and $\mathcal{P}$ [2]. We show that, within $\mathcal{A}$, the weakest failure detector for NBAC is $?\mathcal{P} + \diamond\mathcal{S}$. As a corollary of our results, we state out the relationship between NBAC and another famous agreement problem in distributed computing – Consensus.

1 Introduction

Problem. To ensure the atomicity of a distributed transaction, the processes must agree on a common outcome: *commit* or *abort*. Every process that does not crash during the execution of the algorithm (i.e., a correct process), should eventually decide on an outcome without waiting for crashed processes to recover.

More precisely, the *Non-Blocking Atomic Commit (NBAC)*problem [8] consists for a set of processes to reach a common *decision*, *commit* or *abort*, according to some initial votes of the processes, *yes* or *no*, such that the following properties are satisfied: *(1) Agreement:* no two processes decide differently; *(2) Termination:* every correct process eventually decides; *(3) A-Validity: abort* is the only possible decision if some process votes *no*; and *(4) C-Validity: commit* is the only possible decision if all processes are correct and vote *yes*. For brevity, we denote *yes* and *commit* by 1, *no* and *abort* by 0.

In this paper, we discuss the solvability of the problem in a *crash-stop asynchronous message-passing* model of distributed computing. Informally, the model is one in which processes exchange messages through reliable communication channels, processes can fail by crashing, and there are no bounds on message transmission time and relative processor speeds.

Background. It is well-known that many fundamental agreement problems in distributed computing, in particular, the well-known Consensus problem,

*This work is partially supported by the Swiss National Science Foundation (project number 510-207).

cannot be] solved deterministically in an asynchronous system that is subject to even a single crash failure [3]. In Consensus, the processes need to decide on one out of two values, 0 or 1, based on proposed values, 0 or 1, so that, in addition to the *Agreement* and *Termination* properties of NBAC, the following *Validity* property holds: A value decided must be a value proposed.

To circumvent the impossibility of Consensus, Chandra and Toueg [2] introduced the notion of *failure detector*. Informally, a failure detector is a distributed oracle that gives (possibly incorrect) hints about the crashes of processes. Each process has access to a local *failure detector module* that monitors other processes in the system. In [1], it is shown that a rather weak failure detector $\diamond\mathcal{S}$ is sufficient to solve Consensus in an asynchronous system with a majority of correct processes, and that any failure detector that solves Consensus can emulate $\diamond\mathcal{S}$: hence $\diamond\mathcal{S}$ is *the weakest* failure detector to solve the problem. In other words, $\diamond\mathcal{S}$ encapsulates the exact information about failures needed to solve Consensus in a system with a majority of correct processes. Informally, $\diamond\mathcal{S}$ guarantees that, in any execution, the processes eventually elect a single correct process.

Like Consensus, NBAC does not admit a deterministic solution in an asynchronous system even in the face of a single failure. In this paper we focus on the question of the weakest failure detector to solve NBAC.

Conjecture: $?\mathcal{P} + \diamond\mathcal{S}$. Guerraoui introduced in [6] the *anonymously perfect* failure detector $?\mathcal{P}$ and showed that $?\mathcal{P}$ is *necessary* to solve NBAC. Each module of $?\mathcal{P}$ at a given process outputs either the empty set or the identifier of the process. When the failure detector module of $?\mathcal{P}$ at a process p_i outputs p_i, we say that p_i *detects a crash*. $?\mathcal{P}$ satisfies the following properties: *Anonymous Completeness:* If some process crashes, then there is a time after which every correct process permanently detects a crash, and *Anonymous Accuracy:* No crash is detected unless some process crashes.

In other words, $?\mathcal{P}$ correctly detects that *some* process has crashed, but does not tell *which* process has actually crashed. Clearly, if at most one process can crash, then $?\mathcal{P}$ is equivalent to the perfect failure detector $\mathcal{P}$. An algorithm that transforms Consensus into NBAC using $?\mathcal{P}$ is presented in [6]. Since $\diamond\mathcal{S}$ is sufficient to solve Consensus in a system with a majority of correct processes, $?\mathcal{P} + \diamond\mathcal{S}$ is sufficient to solve NBAC in this environment. It is also shown in [6] that $?\mathcal{P} + \diamond\mathcal{S}$ is strictly weaker than the Perfect failure detector $\mathcal{P}$.

The conjecture we want to prove is that $?\mathcal{P} + \diamond\mathcal{S}$ is the weakest failure detector to solve NBAC (with a majority of correct processes). To show this, we need to prove that any algorithm that solves NBAC can be used to emulate $\diamond\mathcal{S}$.

Assumptions. If we consider the overall universe of failure detectors defined in [2], $\diamond\mathcal{S}$ is not necessary to solve NBAC (i.e., our conjecture is not true). Indeed, Guerraoui introduced in [6] a *stillborn* failure detector, denoted by $\mathcal{B}$, that solves NBAC and cannot be transformed into $?\mathcal{P} + \diamond\mathcal{S}$ [6]. More precisely, $\mathcal{B}$ ensures that every initial crash is immediately detected by every process p_i, so that p_i can safely decide *abort* without synchronizing with others processes.

More generally, a failure detector that tells *the time* when a failure occurred can solve NBAC without employing Consensus. Consider a failure detector $\mathcal{B}[\alpha]$, such that at each process p_i, $\mathcal{B}[\alpha]$ outputs a singleton $\perp$ until some time t_i. At time t_i, if some process is crashed at time α, the failure detector module outputs p_i. Otherwise, after t_i, $\mathcal{B}[\alpha]$ behaves like $\mathcal{P}$ (the perfect failure detector). It can be easily shown that $\mathcal{B}[\alpha]$ is not transformable into $\diamond\mathcal{S}$, although it solves NBAC as follows: each process p_i decides *abort* whenever its failure detector module outputs p_i instead of $\perp$, otherwise p_i runs the 3PC algorithm [8]. However, $\mathcal{B}[\alpha]$ reports the exact time when a failure occurred, which can be provided only through the global time source. It is interesting to know what happens if we rule out *time-based* failure detectors like $\mathcal{B}[\alpha]$: among the remaining failure detectors, is $?\mathcal{P} + \diamond\mathcal{S}$ indeed the weakest to solve NBAC?

The weakest in $\mathcal{A}$. This paper shows that the answer is "yes", i.e., our conjecture is true under the assumption that failure detectors are timeless.

We define a new class $\mathcal{A}$ of *timeless* failure detectors (restricting the original universe of failure detectors of [2]) that excludes time-based failure detectors like $\mathcal{B}[\alpha]$, but includes all known failure detectors like $\mathcal{P}$, $\diamond\mathcal{S}$ and $?\mathcal{P}$. Informally, a timeless failure detector module is not able to provide information about *when* exactly (in the sense of the global time) failures have occurred.

We show that in $\mathcal{A}$, $\diamond\mathcal{S}$ is *necessary* to solve NBAC. That is, any failure detector of $\mathcal{A}$ that solves NBAC can emulate $\diamond\mathcal{S}$. To show this, we extend the technique used in [1] to prove that $\diamond\mathcal{S}$ is necessary to solve Consensus [1]. This extension is not trivial. Given that no information about time can be provided by a timeless failure detector, for any execution scenario, we construct an *imaginary* run that helps eventually deduce valuable information about correct processes in the system and emulate $\diamond\mathcal{S}$.

$?\mathcal{P} + \diamond\mathcal{S}$ is shown to be the weakest failure detector in $\mathcal{A}$ to solve NBAC with a majority of correct processes. As a corollary of our result, we show that in a system equipped with timeless failure detectors, NBAC is strictly harder than Consensus. Roughly speaking, in the class $\mathcal{A}$ of timeless failure detectors, the difference between the problems is exactly captured by $?\mathcal{P}$.

Roadmap. Section 2 gives an intuition of our main result. Section 3 defines our system model. Section 4 presents the class $\mathcal{A}$ of timeless failure detectors. Section 5 gives a brief reminder of the technique of [1] and discusses its applicability to the NBAC problem. Section 6 proves formally that, within $\mathcal{A}$, $\diamond\mathcal{S}$ is necessary to solve NBAC. Section 7 presents the main result of the paper that $?\mathcal{P} + \diamond\mathcal{S}$ is the weakest failure detector to solve the problem. Section 8 concludes the paper by discussing some related work.

2 Intuition: two beer or not two beer

Assume that three guys (Andy, Bob and Clive) want to reach a decision whether to go or not to go to a bar and they can go only if none of them is bankrupt (they are collectivists). Assume also that they have no watches, each of them might reflect over every step arbitrarily long, and that they communicate through a reliable but arbitrarily slow e-mail service (asynchronous model). In taking their decisions, they obey the following rules:

(1) If someone does not have enough money, nobody decides to go.

(2) If nobody is bankrupt or *ill*, everybody decides to go.

(3) No two of them decide differently.

Assume that any of them can get ill. In this case, a nurse (failure detector) taking care of him, calls his friends to tell them that he cannot go. The nurse has bad memory with respect to names, she can only say that *someone* is ill. Intuitively, this corresponds to the use of failure detector $?\mathcal{P}$. Assume that everybody has enough money to go out and consider the following cases:

1. Suppose that the nurse possesses a gift of oracle: she can reliably identify whether someone is ill from the very beginning (intuitively, this corresponds to the use of the stillborn failure detector $\mathcal{B}$ of [6]). Assume that every guy, before doing anything, queries the nurse and waits for her response. If the response is "one of you is initially ill", the guy independently decides to skip going out tonight. Otherwise, they communicate with each other in order to learn everybody's intention.

   However, such a gifted nurse might be difficult to find. Usually, people are not that wise. In the practical sense, it would be more meaningful to consider the following case:

2. The nurse is very busy, so the others can be informed about their friend's illness arbitrarily late. Assume that Clive is ill, and the nurse decides to call his friends. It might happen that Andy received a call from the nurse, while Bob is still in ignorance and he thinks that everybody is able to go out.

   The problem is that in order to reach a common decision they need to synchronize their knowledge, but they never can do it without communication. A possible algorithm for any of them, say Bob, can be something like this:

   (1) Bob waits until he receives the wishes of the others or a call from the nurse that somebody is ill;

   (2) If everybody wants to go (no call from the nurse is received), Bob tries to agree with the others proposing to go. Otherwise, Bob tries to agree with the others proposing not to go.

   As we know from [1], to solve agreement in an asynchronous system, we need an external source that eventually informs every healthy guy that some particular one is currently in health (a failure detector that is convertible to $\Diamond\mathcal{S}$). This one will decide for all.

This simple example conveys the intuition that $?\mathcal{P} + \Diamond\mathcal{S}$ is necessary and sufficient to solve NBAC in a system equipped with timeless failure detectors. However, some not timeless (time-based) failure detectors allow to make do without agreement and, thus, without $?\mathcal{P} + \Diamond\mathcal{S}$.

3 Model

We consider in this paper a crash-prone asynchronous message passing model augmented with the failure detector abstraction. We recall here what in the model is needed to state and prove our results. More details on the model can be found in [2].

System. We assume the existence of a global clock to simplify the presentation. The processes do not have *direct* access to the clock (timing assumptions are captured within failure detectors). We take the range $\mathcal{T}$ of the clock output values to be the set of natural numbers and the integer 0, ($\{0\} \cup \mathbb{N}$). The system consists of a set of n processes $\Pi = \{p_1, .., p_n\} (n > 1)$. Every pair of processes is connected by a reliable communication channel. The systems is *asynchronous*: there is no time bound on message delay, clock drift, or the time necessary to execute a step [3].

Failures and failure patterns. Processes are subject to *crash* failures. A *failure pattern* F is a function from the global time range $\mathcal{T}$ to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed by time t. Once a process crashes, it does not recover, i.e., $\forall t < t' : F(t) \subseteq F(t')$. We define $correct(F) = \Pi - \cup_{t \in \mathcal{T}} F(t)$ to be the set of *correct* processes. A process $p_i \notin F(t)$ is said to be *up* at time t. A process $p_i \in F(t)$ is said to be *crashed* (or *incorrect*) at time t. We do not consider Byzantine failures: a process either correctly executes the algorithm assigned to it, or crashes and stops executing any action forever. An *environment* $\mathcal{E}$ is a set of failure patterns. By default, we consider an environment of the form $\mathcal{E}_f$ that includes all failure patterns in which up to f processes can fail. We assume that there is at least one correct process: $0 \leq f < n$. We denote by F_0 the failure-free failure pattern ($correct(F_0) = \Pi$).

Failure detectors. A *failure detector history H with range* $\mathcal{R}$ is a function from $\Pi \times \mathcal{T}$ to $\mathcal{R}$. $H(p_i, t)$ is the output of the failure detector module of process p_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 $\mathcal{D}(F)$ with range $\mathcal{R}_{\mathcal{D}}$.

Every process p_i has a failure detector module $\mathcal{D}_i$ that p_i queries to obtain information about the failures in the system. Typically, this information includes the set of processes that a process currently suspects to have crashed.[1] Among the failure detectors defined in [1, 2], we consider the following one:

Perfect ($\mathcal{P}$): the output of every $\mathcal{P}_i$ is a set of suspected processes satisfying *strong completeness* (i.e., every incorrect process is eventually suspected by *every* correct process) and *strong accuracy* (i.e., no process is suspected before it crashes);

Eventually strong ($\diamond\mathcal{S}$): the output of every $\diamond\mathcal{S}_i$ is a set of suspected processes satisfying strong completeness and *eventual weak accuracy* (i.e., there is a time after which one correct process is never suspected).

Eventual leader (Ω): the output of each failure detector module Ω_i is a single process p_j, that p_i currently *trusts*, i.e., that p_i considers to be correct ($\mathcal{R}_{\Omega} = \Pi$). For every failure pattern, there is a time after which all correct processes always trust the same correct process. Obviously, Ω

[1] In [1], failure detectors can output values from an arbitrary range. In determining the weakest failure detector for NBAC, we also do not make any assumption a priori on the range of a failure detector.

provides at least as much information as $\diamond\mathcal{S}$: if every process p_i always suspects $\Pi - \{\Omega_i\}$, the properties of $\diamond\mathcal{S}$ are guaranteed [1].

We consider also the anonymously perfect failure detector $?\mathcal{P}$ [6], such that each module of $?\mathcal{P}$ at a given process outputs either 0 or 1 ($\mathcal{R}_{?\mathcal{P}} = \{0,1\}$). When the failure detector module of $?\mathcal{P}$ at a process p_i outputs 1, we say that p_i *detects a crash*. $?\mathcal{P}$ satisfies the following properties: *anonymous completeness* (i.e., if some process crashes, then there is a time after which every correct process permanently detects a crash), and *anonymous accuracy* (i.e., no crash is detected unless some process crashes).

For any failure pattern F, $\mathcal{P}(F)$, $\diamond\mathcal{S}(F)$, $\Omega(F)$ and $?\mathcal{P}(F)$ denote the sets of *all* histories satisfying the corresponding properties. Recalling the notion of failure detector *classes* introduced in [2], every failure detector above denotes here the weakest element in the classes of failure detectors satisfying the corresponding properties.

Algorithms. We model the set of asynchronous communication channels as a message buffer which contains messages not yet received by their destinations. An algorithm A is a collection of n (possibly infinite state) deterministic automata, one for each of the processes. $A(p_i)$ denotes the automaton running on process p_i. In each step of A, process p_i performs atomically the following three actions: (receive phase) p_i chooses *non-deterministically* a single message addressed to p_i from the message buffer, or a null message, denoted λ; (failure detector query phase) p_i queries and receives a value from its failure detector module; (local state update phase) p_i changes its state; and (send phase) sends a message to all processes according to the automaton $A(p_i)$, based on its state at the beginning of the step, the message received in the receive action, and the value obtained by p_i from its failure detector module.[2]

Configurations, schedules and runs. A *configuration* defines the current state of each process in the system and the set of messages currently in the message buffer. Initially, the message buffer is empty. A step (p_i, m, d, A) of an algorithm A is uniquely determined by the identity of the process p_i that takes the step, the message m received by p_i during the step (m might be the null message λ), and the failure detector value d seen by p_i during the step. We say that a step $e = (p_i, m, d, A)$ *is applicable* to the current configuration if and only if $m = \lambda$ or m is a message from the current message buffer destined to p_i. $e(C)$ denotes the unique configuration that results when e is applied to C. A *schedule* S of algorithm A is a (finite or infinite) sequence of steps of A. $S_\perp$ denotes the empty schedule. We say that a schedule S *is applicable to a configuration* C if and only if (a) $S = S_\perp$, or (b) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$, etc. For a finite schedule S applicable to C, $S(C)$ denotes the unique configuration that results from applying S to C.

A *partial run of algorithm A in an environment $\mathcal{E}$ using a failure detector $\mathcal{D}$* is a tuple $R = \langle F, H_\mathcal{D}, I, S, T\rangle$, where $F \in \mathcal{E}$ is a failure pattern, $H_\mathcal{D} \in \mathcal{D}(F)$ is a failure detector history, I is an initial configuration of A, S is a *finite* schedule of A, and $T \subset \mathcal{T}$ is a *finite* list of increasing time values, such that $|S| = |T|$, S

[2] Our result also applies to weaker models where a step can atomically comprise at most one phase and where a process can atomically send at most one message to a single process per step.

is applicable to I, and for all $t \leq |S|$, if $S[t]$ is of the form (p_i, m, d, A) then: (1) p_i has not crashed by time $T[t]$, i.e., $p_i \notin F(T[t])$ and (2) d is the value of the failure detector module of p_i at time $T[t]$, i.e., $d = H_{\mathcal{D}}(p_i, T[t])$.

A *run of algorithm A in an environment $\mathcal{E}$ using a failure detector $\mathcal{D}$* is a tuple $R = \langle F, H_{\mathcal{D}}, I, S, T\rangle$, where S is an *infinite* schedule of A and $T \subseteq \mathcal{T}$ is an *infinite* list of increasing time values indicating when each step of S occurred. In addition to satisfying the properties (1) and (2) of a partial run, run R should guarantee that (3) every correct process in F takes an infinite number of steps in S and eventually receives every message sent to it (this conveys the reliability of the communication channels).

Weakest failure detector. A *problem* (e.g., NBAC or Consensus) is a set of runs (usually defined by a set of properties that these runs should satisfy). We say that a failure detector $\mathcal{D}$ *solves a problem M in an environment $\mathcal{E}$* if there is an algorithm A, such that all the runs of A in $\mathcal{E}$ using $\mathcal{D}$ are in M (i.e., they satisfy the properties of M).

Let $\mathcal{D}$ and $\mathcal{D}'$ be any two failure detectors and $\mathcal{E}$ be any environment. If there is an algorithm $T_{\mathcal{D}' \rightarrow \mathcal{D}}$ that emulates $\mathcal{D}$ with $\mathcal{D}'$ in $\mathcal{E}$ ($T_{\mathcal{D}' \rightarrow \mathcal{D}}$ is called a *reduction* algorithm), we say that *$\mathcal{D}$ is weaker than $\mathcal{D}'$ in $\mathcal{E}$*, or $\mathcal{D} \preceq_{\mathcal{E}} \mathcal{D}'$. If $\mathcal{D} \preceq_{\mathcal{E}} \mathcal{D}'$ but $\mathcal{D}' \not\preceq_{\mathcal{E}} \mathcal{D}$ we say that *$\mathcal{D}$ is strictly weaker than $\mathcal{D}'$ in $\mathcal{E}$*, or $\mathcal{D} \prec_{\mathcal{E}} \mathcal{D}'$.[3] Note that $T_{\mathcal{D}' \rightarrow \mathcal{D}}$ does not need to emulate *all* histories of $\mathcal{D}$; it is required that all the failure detector histories it emulates be histories of $\mathcal{D}$.

We say that a failure detector $\mathcal{D}$ is *the weakest failure detector to solve a problem M in an environment $\mathcal{E}$* if two conditions are satisfied: (1) Sufficiency: $\mathcal{D}$ solves M in $\mathcal{E}$, and (2) Necessity: if a failure detector $\mathcal{D}'$ solves M in $\mathcal{E}$ then $\mathcal{D} \preceq_{\mathcal{E}} \mathcal{D}'$.

We say that *problem M is harder than problem M' in environment $\mathcal{E}$*, if any failure detector $\mathcal{D}$ solving M in $\mathcal{E}$ solves also M' in $\mathcal{E}$. Respectively, *M is strictly harder than M' in $\mathcal{E}$*, if M is harder than M' in $\mathcal{E}$ and there exists a failure detector $\mathcal{D}'$ that solves M' (in $\mathcal{E}$) but not M.

4 Timeless failure detectors

This section introduces a new class $\mathcal{A}$ of *timeless* failure detectors. Intuitively, a timeless failure detector module is not able to provide information about *when* exactly (in the sense of the global time) failures have occurred.

We denote by F_0 the *failure-free* failure pattern: $\forall t \in \mathcal{T}, F_0(t) = \emptyset$. For any $F \in \mathcal{E}_f$ and $\delta \in \mathcal{T}$, we introduce the failure pattern F_δ, such that, for all $t \in \mathcal{T}$:

$$F_\delta(t) = \begin{cases} \emptyset & \text{if} \quad t < \delta \\ F(t - \delta) & \text{if} \quad t \geq \delta \end{cases}$$

Thus, for a failure that occurs at time t in F, the corresponding failure in F_δ occurs at $t + \delta$, and no failure occurs before time δ in F_δ. Note that $\forall \delta \in \mathcal{T}$: $\mathit{correct}(F) = \mathit{correct}(F_\delta)$. That is, a process is correct in F if and only if it is correct in F_δ. Thus, for any f, if $F \in \mathcal{E}_f$, then $F_\delta \in \mathcal{E}_f$.

[3]Later we omit $\mathcal{E}$ in $\prec_{\mathcal{E}}$ and $\preceq_{\mathcal{E}}$ when there is no ambiguity on the environment $\mathcal{E}$.

Formally, the class $\mathcal{A}$ consists of all (timeless) failure detectors $\mathcal{D}$, such that:

$$
\begin{array}{l}
\exists H_0 \in \mathcal{D}(F_0),\ \forall \delta \in \mathcal{T} \\
\forall F \in \mathcal{E}_f,\ \forall H \in \mathcal{D}(F), \\
\exists H_\delta \in \mathcal{D}(F_\delta) : \forall p_i \in \Pi,\ \forall t \in \mathcal{T}, \\
H_\delta(p_i, t) = \begin{cases} H_0(p_i, t) & \text{if} \quad t < \delta \\ H(p_i, t - \delta) & \text{if} \quad t \geq \delta \end{cases}
\end{array} \tag{1}
$$

It follows from (1) that, for any failure detector $\mathcal{D} \in \mathcal{A}$ and $\delta \in \mathcal{T}$, if a failure occurred at time t_0 and is reported by a module of $\mathcal{D}$ at time t_1, then a failure that occurred at $t_0 + \delta$ could be reported *in the same way* at $t_1 + \delta$: the process does not know when exactly the failure occurred. This captures our idea that failure detectors of class $\mathcal{A}$ provide no information about the time when failures occur.

From now on, we restrict our scope from the original universe of failure detectors [2] to $\mathcal{A}$. Note that $\mathcal{P}$, $\diamond\mathcal{S}$, Ω and $?\mathcal{P}$ are timeless. With $?\mathcal{P}$, for instance, a process can detect the very fact that some process has crashed, but there is no way to acquire the actual time at which the failure occurred.

Examples of histories output by a failure detector $\mathcal{D}$ from $\mathcal{A}$ are depicted in Figure 1. The system consists of two processes p_1 and p_2. At any time, $\mathcal{D}_i$ outputs a set of processes suspected by p_i ($i = 1, 2$). Assume that (a) p_2 crashes at time t_1 and p_1 detects the failure at time t_2. Then, by the definition of $\mathcal{A}$, if (b) p_2 crashes at time $t_1 + d$, then there exists a history of $\mathcal{D}$ in which p_1 detects the failure at time $t_2 + d$.

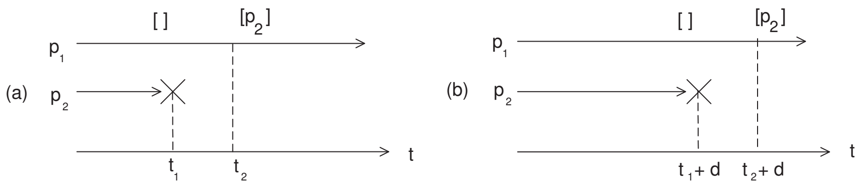

Figure 1: Examples of timeless failure detector histories.

Consider the failure detector $\mathcal{B}[\alpha]$ mentioned in the introduction: each module of $\mathcal{B}[\alpha]$ outputs $\perp$ or a subset of the processes in Π ($\mathcal{R}_{\mathcal{B}[\alpha]} = \perp \cup 2^{\Pi}$). Formally, $\mathcal{B}[\alpha]$ is defined as follows:

$$
\begin{array}{l}
\forall F \in \mathcal{E}_f, \forall H \in B[\alpha](F), \\
\exists H_{\mathcal{P}} \in \mathcal{P}(F), \\
\forall p_i \in \Pi, \exists t_i \in \mathcal{T}, \forall t \in \mathcal{T} : \\
H(p_i, t) = \begin{cases} \perp & \text{if } t < t_i \\ p_i & \text{if } t \geq t_i \ \wedge \ F(\alpha) \neq \emptyset \\ H_{\mathcal{P}}(p_i, t) & \text{if } t \geq t_i \ \wedge \ F(\alpha) = \emptyset \end{cases}
\end{array}
$$

(The Stillborn failure detector $\mathcal{B}$ is a particular case of $\mathcal{B}[\alpha]$ with $\alpha = 0$ and $t_i = 0, \forall p_i \in \Pi$.) Clearly, $\mathcal{B}[\alpha]$ does not belong to $\mathcal{A}$. Indeed, consider a failure pattern F in which only one process is crashed at time α. Take a corresponding history $H \in B[\alpha](F)$. For every process p_i, there is a time t_i, such that, for any $t \geq t_i$, $H(p_i, t) = p_i$. Now consider a failure pattern F_δ in which no process is crashed at time α (the condition holds for all $\delta > \alpha$). Since failure detector module of p_i behaves now like $\mathcal{P}_i$ (i.e., the perfect failure detector), its own identity p_i is never output. Thus, $\mathcal{B}[\alpha] \notin \mathcal{A}$.

5 Proof technique

Before proving that, in $\mathcal{A}$, Ω is necessary to solve NBAC, we briefly recall here the technique used in [1] to prove that Ω is necessary to solve Consensus and we discuss the applicablity of this technique to NBAC.

The weakest failure detector to solve Consensus. Let $\mathcal{E}$ be any environment, $\mathcal{D}$ be any failure detector that solves Consensus in $\mathcal{E}$, and $\mathsf{Cons}_{\mathcal{D}}$ be any Consensus algorithm that uses $\mathcal{D}$. The algorithm $T_{\mathcal{D}\to\Omega}$ that transforms $\mathcal{D}$ into Ω in $\mathcal{E}$ works as follows.

Fix an arbitrary run of $T_{\mathcal{D}\to\Omega}$ using $\mathcal{D}$, with failure pattern $F \in \mathcal{E}$ and failure detector history $H_{\mathcal{D}} \in \mathcal{D}(F)$. All processes periodically query their failure detector $\mathcal{D}$ and exchange information about the values of $H_{\mathcal{D}}$ that they see in this run. Using this information, the processes construct a directed acyclic graph (DAG) that represents a "sampling" of failure detector values in $H_{\mathcal{D}}$ and causal relationships between the values. By periodically sending its current version of the DAG to all processes, and incorporating all the DAGs that it receives into its own DAG, every correct process constructs ever increasing finite approximations of the same infinite limit DAG G.

The DAG G can be used to simulate runs of $\mathsf{Cons}_{\mathcal{D}}$ with failure pattern F and failure detector history $H_{\mathcal{D}}$. These runs *could have happened* if processes were running $\mathsf{Cons}_{\mathcal{D}}$ instead of $T_{\mathcal{D}\to\Omega}$. If we simulate all possible runs of $\mathsf{Cons}_{\mathcal{D}}$ applied to the DAG G with all possible initial configurations I, we obtain a *simulation forest*: a tree for each initial configuration.

Thus, the infinite DAG G induces an infinite simulation forest Υ of runs of $\mathsf{Cons}_{\mathcal{D}}$ with failure pattern F and failure detector history $H_{\mathcal{D}} \in \mathcal{D}(F)$. From the properties of the Consensus problem, it follows that Υ comprises schedules corresponding to the runs of Consensus in which every correct process decides 0 and runs in which every correct process decides 1. This allows to design a deterministic algorithm that identifies a process p^* that is correct in F, namely a process whose step defines which decision is going to be taken by the rest of correct processes in the descending schedules.

Although the simulation forest Υ is infinite and cannot be computed by any process, there exists a *finite subforest* of Υ that gives sufficient information to identify p^*. Thus, there is a time after which, every correct process running $T_{\mathcal{D}\to\Omega}$ obtains a reference p^*. In other words, $T_{\mathcal{D}\to\Omega}$ emulates Ω.

NBAC: a hard nut. As we discussed in the introduction, Ω is not necessary to solve NBAC [6]. Thus, the technique of [1] cannot be directly applied. For instance, if a module of the stillborn failure detector $\mathcal{B}$ outputs $\perp$, then there is an initial failure in the system and NBAC is trivially solved by deciding 0 at every correct process. There is no way to identify *correct* processes and, thus, no algorithm $T_{\mathcal{B}\to\Omega}$ is possible.

However, even if we exclude failure detectors like $\mathcal{B}[\alpha]$ by considering timeless failure detectors only (i.e., focusing on class $\mathcal{A}$), we are still not able to apply the technique of [1]. Indeed, let $\mathcal{D}$ be any failure detector that solves NBAC in an environment $\mathcal{E}$. Consider a run of an NBAC algorithm using $\mathcal{D}$ in a failure pattern F, such that $F(0) \neq \emptyset$ (some process is initially crashed). Clearly, no process can decide 1 (no matter which failure detector history $H_{\mathcal{D}}$ is output by $\mathcal{D}$). The only decision a correct process can take is 0 (otherwise, the A-Validity property of NBAC would be violated, since some $p \in F(0)$ could have voted 0). In this case, the corresponding simulation forest Υ does not bring any valuable information about failures to identify a correct process (no matter which failure detector history $H_{\mathcal{D}}$ is output by $\mathcal{D}$).

Fortunately, thanks to the very nature of timeless failure detectors, we can modify the original DAG G in order to fetch a valuable information about correct processes of F. The details are presented in Section 6.

6 Necessary condition

This section shows the necessity of Ω to solve NBAC using failure detectors in class $\mathcal{A}$. To this end, we present a reduction algorithm $T_{\mathcal{D}\rightarrow\Omega}$ transforming any failure detector $\mathcal{D} \in \mathcal{A}$ that solves NBAC into Ω. A corollary of our result is that $\Diamond\mathcal{S}$ (which is weaker than Ω) is necessary to solve NBAC (using $\mathcal{A}$), and hence $?\mathcal{P} + \Diamond\mathcal{S}$ is the weakest failure detector within $\mathcal{A}$ to solve NBAC.

Nice runs and nice DAGs. Let $\mathcal{D}$ be any failure detector in $\mathcal{A}$ and $NBAC_{\mathcal{D}}$ be any NBAC algorithm using $\mathcal{D}$. From now on, we denote by $e = (p, m, d)$ a step of process p executing $NBAC_{\mathcal{D}}$.

Let F_0 be the failure-free pattern and H_0 be the history from $\mathcal{D}(F_0)$, such that the condition (1) in Section 4 for $\mathcal{D}$ holds with H_0. Let I be any initial configuration in which all processes vote 1. Due to the properties of NBAC, there exists a partial run $R_0 = \langle F_0, H_0, I, S_0, T_0\rangle$ of $NBAC_{\mathcal{D}}$ comprising a finite number of steps in which every process decides 1.

Taking the *nice run* R_0 as a basis, we can now construct a *nice DAG* (directed acyclic graph) G_0 induced by the failure-free pattern F_0. For any step $e = (p_i, m, d)$ in S_0, we create a vertex $[p_i, d, k]$ of G_0, where $k - 1$ is the number of steps of p_i in S_0 preceding e. For any steps $e_1 = (p_i, m, d)$ and $e_2 = (p_j, m', d')$ in S_0, such that e_1 precedes e_2 in S_0, we create a corresponding edge in $[p_i, d, k] \rightarrow [p_j, d', k']$ in G_0. This means that p_j queried its failure detector for the k'-th time *after* p_i queried its failure detector for the k-th time.

Constructing a DAG. Let F be any failure pattern from $\mathcal{E}_f$ and $H \in \mathcal{D}(F)$ and assume that the same nice DAG G_0 is initially available to all processes. Consider a run R of $T_{\mathcal{D}\rightarrow\Omega}$. Processes periodically query their failure detector $\mathcal{D}$ and exchange information about the values of $H \in \mathcal{D}(F)$ that they see in the current run. Using this information, every process p_i constructs an *imaginary* DAG G_i, in which the real samples of H are assumed to be seen *after* all the values of H_0 presented in G_0. That is, every time a process p_i sees a failure detector value d, (1) a new vertex $[p_i, d, k]$ is added to G_i, such that $k = k_0 + k_R$, where k_0 is the number of steps of p_i in S_0 and k_R is the number of times p_i queried its failure detector module so far, and (2) a new edge from every vertex of G_i to $[p_i, d, k]$ is added. As a result, every correct process p maintains an ever growing graph $G_i(t)$, such that $G_i(t) \rightarrow_{t\rightarrow\infty} G$ for some infinite DAG G. Note that G contains a sampling of the failure detector history H corresponding to the *real* failure pattern F ($H \in \mathcal{D}(F)$) as well as of some *imaginary* history $H_0 \in \mathcal{D}(F_0)$, where F_0 is the failure-free pattern.

Let $\tilde{F}$ be any failure pattern and $\tilde{H}$ be any history in $\mathcal{D}(\tilde{F})$. Let $\tilde{G}$ be an infinite directed acyclic graph (DAG) defined by the set of vertices $V(\tilde{G})$ and a set of directed edges $E(\tilde{G})$ of the form $v \to v'$, where $v \in V(\tilde{G})$ and $v' \in V(\tilde{G})$, with the following properties:

(1) The vertices of $\tilde{G}$ are of the form $[p_i, d, k]$ where $p_i \in \Pi$, $d \in \mathcal{R}_{\mathcal{D}}$ and $k \in \mathbb{N}$. There is a mapping $f : V(\tilde{G}) \to \mathcal{T}$ that associates a time with each vertex of $\tilde{G}$, such that:

   (a) For any $v = [p_i, d, k] \in V(\tilde{G})$, $p_i \notin F(f(v))$ and $d = \tilde{H}(p_i, f(v))$.

   (b) For any edge $v \to v' \in E(\tilde{G})$, $f(v) < f(v')$.

(2) If $[p_i, d, k] \in V(\tilde{G})$,$[p_i, d', k'] \in V(\tilde{G})$ and $k < k'$ then $[p_r, d, k] \to [p_s, d', k'] \in E(\tilde{G})$.

(3) $\tilde{G}$ is transitively closed.

(4) Let $U \subseteq V(\tilde{G})$ be a finite set of vertices and p_i be any correct process in F. There is $d \in \mathcal{R}_{\mathcal{D}}$ and $k \in \mathbb{N}$, such that for every vertex $v \in V$, $v \to [p_i, d, k]$ is an edge of $\tilde{G}$.

Then we say that DAG $\tilde{G}$ *is a sampling DAG of history* $\tilde{H}$.

It can be easily seen that $G \backslash G_0$, a DAG that includes "real" vertices and edges only, is a sampling DAG of H. The following lemma precisely captures the relationship between the real failure pattern F and the infinite DAG G:

Lemma 1 *There exists $\delta \in \mathcal{T}$ and a failure detector history $H_\delta \in \mathcal{D}(F_\delta)$ such that G is a sampling DAG of H_δ.*

Proof: Assume that the last step of S_0 happened at time $t_0 \in \mathcal{T}$. Take any $\delta \in \mathcal{T}$, such that $\delta > t_0$. Let H_0 bt the history from $\mathcal{D}(F_0)$, such that the condition (1) in Section 4 for $\mathcal{D}$ holds with H_0. Construct H_δ as follows:

$$\forall t \in \mathcal{T},\ p \in \Pi,$$

$$H_\delta(p,t) = \begin{cases} H_0(p,t) & \text{if} \quad t < \delta \\ H(p, t-\delta) & \text{if} \quad t \geq \delta \end{cases} \tag{2}$$

Recall that $\mathcal{D} \in \mathcal{A}$. By the definition of $\mathcal{A}$, $H_\delta \in \mathcal{D}(F_\delta)$. Now we show that G is a sampling DAG of H_δ, i.e., that the properties 1-5 above are satisfied. Indeed:

1. Define $f : V(G) \to \mathcal{T}$ as follows. Take $v \in V(G)$. If $v \in V(G_0)$ represents a step $S_0[k]$, then $f(v) = T_0[k]$. If v is a vertex of $G \backslash G_0$, such that $v = [p_i, d, k]$ and $d = H(p_i, t)$, for some $t \in \mathcal{T}$, then $f(v) = T_0[k]$.

   (a) For any $v = [p_i, d, k] \in V(G)$, $p_i \notin F_\delta(f(v))$ and $d = \tilde{}H(p_i, f(v))$.

   (b) Take an edge $v \to v'$ of G, where $v = [p_i, d, k]$ and $v' = [p_j, d', k']$, such that $d = H_\delta(p_i, f(v))$ and $d' = H_\delta(p_j, f(v'))$. Four cases are possible:

      i. $f(v) < \delta$, $f(v') < \delta$: both vertices belong to G_0. Moreover, the vertices correspond to some steps e and e' of S_0, and e precedes e. Due to the definition of G_0, $f(v) < f(v')$.

ii. $f(v) \geq \delta$, $f(v') \geq \delta$: both vertices belong to $G \backslash G_0$, that is $d = H(p_i, f(v) - \delta)$ and $d' = H(p_j, f(v') - \delta)$. Since $G \backslash G_0$ is a sampling DAG of H, $f(v) < f(v')$.

iii. $f(v) < \delta$, $f(v') \geq \delta$: clearly, $f(v) < f(v')$;

iv. $f(v) \geq \delta$, $f(v') < \delta$: $[p_i, d, k] \in G$ and $[p_j, d', k'] \in G_0$. But by the construction of G, every vertex of G is seen *after* every vertex of G_0, that is $[p_j, d', k'] \to [p_i, d, k]$: a contradiction with the initial assumption.

Thus, $f(v) < f(v')$.

2. Let $v = [p_i, d, k]$ and $v' = [p_i, d', k']$ be any vertices of G and $k < k'$. Four cases are possible:

   (a) $v \in G_0$, $v' \in G_0$: by the definition of G_0, $v \to v'$ is an edge of G_0, and thus of $G_0 \cup G$;

   (b) $v \in G \backslash G_0$, $v' \in G \backslash G_0$: by the definition of a sampling DAG, $v \to v'$ is an edge of $G \backslash G_0$, and thus of G;

   (c) $v \in G_0$, $v' \in G \backslash G_0$: by the construction rule of G, $v \to v'$ is an edge of G;

   (d) $v \in G \backslash G_0$, $v' \in G_0$: contradicts the construction rule of G (k must be greater than k').

   Thus, $v \to v'$ is an edge of G.

3. By the construction rule of G and the fact that both G_0 and $G \backslash G_0$ are transitively closed, the resulting DAG G is transitively closed.

4. From the facts that G_0 is finite, that, for any $v \in V(G)$ and $v' \in V(G \backslash G_0)$, $v \to v'$ is an edge of G, and that $G \backslash G_0$ is a sampling DAG of H, it follows that, for any finite subset V of vertices of G and any correct process p_i, there is $d \in \mathcal{R}_{\mathcal{D}}$ and $k \in \mathbb{N}$, such that for every vertex $v \in V$, $v \to [p_i, d, k]$ is an edge of G.

Thus, for some $\delta \in \mathcal{T}$, there a failure detector history $H_\delta \in \mathcal{D}(F_\delta)$, such that G is a sampling DAG of H_δ. □

Thus, G represents a sample of a failure detector history H_δ that *could have been* seen if the failure pattern was F_δ. Note that even if a process p is initially crashed in F, G contains the samples of its failure detector module output. However, the number of vertices of the form $[p, \cdot, \cdot] \in G$ is finite, thus, p cannot be considered to be correct in F_δ. In other words, a crashed process in F cannot appear to be correct in F_δ.

Tags and decision gadgets. Lemma 1 allows us to use G to simulate some of the runs of $\mathsf{NBAC}_{\mathcal{D}}$ in the failure pattern F_δ. Take an initial configuration I of $\mathsf{NBAC}_{\mathcal{D}}$ in which every process votes 1. The set of simulated schedules of $\mathsf{NBAC}_{\mathcal{D}}$ that are compatible with some path of G and are applicable to I can be organized as a tree Υ: paths in this tree represent simulated schedules of $\mathsf{NBAC}_{\mathcal{D}}$ with initial configuration I. The fact that $G_0 \subset G$ guarantees that there exists a schedule in Υ in which every process decides 1.

Following [1], we assign a set of tags (*abort* or *commit*) to each vertex of the simulation tree Υ induced by G. Vertex S of tree Υ gets tag k if and only if it has a descendant S' (possibly $S = S'$) such that some correct process has decided k in $S'(I)$. A vertex of Υ is *monovalent* if it has only one tag, and *bivalent* if it has both tags (following the terminology of [3]).

Still following [1], we also introduce the notion of *decision gadgets* and *deciding processes* and show that any deciding process in Υ is correct. Informally, a decision gadget is a vertex S of Υ having exactly two monovalent leaves: one 0-valent and one 1-valent. In turn, a deciding process of S is a process whose step defines the decision taken by a descendant of S. The following lemma gives a condition of the existence of at least one decision gadget in Υ

Lemma 2 *If $correct(F) \neq \Pi$ (F is not failure-free), then Υ has a decision gadget.*

Proof: Let $p \notin correct(F)$. There exists a *finite* schedule E in Υ containing only steps of correct processes such that all correct processes have decided in in $E(I)$ (Lemma 10 of [1]). Since E contains no step of process p, no information is available about its initial vote, and the decision value must be 0 (otherwise the A-Validity property is violated). From the way the simulation tree is constructed, it follows that Υ contains a schedule in which 1 is decided. Thus the initial configuration of Υ is bivalent. By Lemma 18 of [1], Υ has at least one decision gadget (and hence a deciding process). $\square$

Reduction algorithm. The reduction algorithm $T_{\mathcal{D}\to\Omega}$ presented in Figure 2 works as follows. Every process p_i periodically updates and tags a simulation tree Υ_i induced by G_i with the initial configuration I in which all processes vote *yes*. If there exists a decision gadget in Υ_i, then $T_{\mathcal{D}\to\Omega}$ outputs the deciding process of the smallest decision gadget of Υ_i (since the set of vertices of Υ_i is countable, we can easily impose a rule to define the smallest decision gadget in it), otherwise $T_{\mathcal{D}\to\Omega}$ outputs p_1. Note that for any correct process p_i, $G = \lim_{t\to\infty} G_i(t)$ and thus $\Upsilon = \lim_{t\to\infty} \Upsilon_i(t)$.

Theorem 3 *There exists a process $p^* \in correct(F)$, such that, for every correct process p_i, there is a time after which $T_{\mathcal{D}\to\Omega}$ outputs p^* at p_i, forever.*

Proof: Consider $\delta \in \mathcal{T}$ and failure pattern F_δ, such that G is a sampling DAG for some $H_\delta \in \mathcal{D}(F_\delta)$. Note that $correct(F) = correct(F_\delta)$. Two cases are possible:

(1) F, and thus F_δ are failure-free. Then all vertices in Υ are monovalent and the reduction algorithm forever outputs $p_1 \in correct(F)$.

(2) F, and thus F_δ are not failure-free. By Lemma 2, Υ has a deciding process. Let p^* be the deciding process of the smallest decision gadget. Since ever growing simulation trees $\Upsilon_i(t)$ of all correct processes p_i tend to Υ, there exists t_0 such that $\forall t > t_0, \forall p \in correct(F)$, the deciding process of the smallest decision gadget is p^*. Thus, $\forall t > t_0$ all correct processes p_i have $output_i = p^*$. By Lemma 21 of [1], the deciding process is correct in F.

$G_i \leftarrow G_0$
$k \leftarrow 0$
while true **do**
  p_i receives m
  $d \leftarrow$ output of $\mathcal{D}$
  $k \leftarrow k + 1$
  if m is of the form (p_j, G_j, p_i) **then**
    $G_i \leftarrow G_i \cup G_j$
  add $[p_i, d, k]$ to G_i and edges from all other vertices of G_i to $[p_i, d, k]$
  $\Upsilon_i \leftarrow$ simulation tree induced by G_i and I
  if Υ_i has no decision gadgets **then**
    $output_i \leftarrow p_1$
  else
    $output_i \leftarrow$ deciding process of the smallest decision gadget of Υ_i
  p_i sends (p, G_i, q) to all $q \in \Pi$

Figure 2: Reduction algorithm $T_{\mathcal{D}\rightarrow\Omega}$ for process p_i.

Thus $T_{\mathcal{D}\rightarrow\Omega}$ eventually outputs the identity of the same correct process, at every correct process. □

Theorem 4 *For any environment $\mathcal{E}_f$, if a failure detector $\mathcal{D} \in \mathcal{A}$ can be used to solve NBAC in $\mathcal{E}_f$, then $\mathcal{D} \succeq_{\mathcal{E}_f} \Omega$.*

7 The weakest failure detector in $\mathcal{A}$ to solve NBAC

Consider failure detector $?\mathcal{P} + \Diamond\mathcal{S}$ ($\mathcal{R}_{?\mathcal{P}+\Diamond\mathcal{S}} = \mathcal{R}_{?\mathcal{P}} \times \mathcal{R}_{\Omega}$), such that, for any failure pattern F and for any $(H_{?\mathcal{P}}, H_{\Omega}) \in ?\mathcal{P} + \Diamond\mathcal{S}(F)$, we have $H_{?\mathcal{P}} \in ?\mathcal{P}(F)$ and $H_{\Omega} \in \Omega(F)$.

From the facts that $?\mathcal{P}$ is necessary to solve NBAC in any environment [6], Ω is weaker than Ω [1], and Theorem 4, the following result holds:

Theorem 5 *For any environment $\mathcal{E}_f$, if a failure detector $\mathcal{D} \in \mathcal{A}$ solves NBAC in $\mathcal{E}_f$, then $\mathcal{D} \succeq_{\mathcal{E}_f} ?\mathcal{P} + \Omega$.*

From the theorem above and [6], we have:

Corollary 6 *$?\mathcal{P} + \Diamond\mathcal{S}$ is the weakest timeless failure detector to solve NBAC in any environment $\mathcal{E}_f$ with $f < \lceil \frac{n}{2} \rceil$.*

Corollary 7 *For any environment $\mathcal{E}_f$ with $0 < f < \lceil \frac{n}{2} \rceil$ in a system augmented with timeless failure detectors, NBAC is strictly harder than Consensus.*

Proof: Since Ω is timeless and it is the weakest to solve Consensus (in $\mathcal{E}_f$) in the whole universe of failure detectors [2], it is also the weakest failure detector to solve Consensus in $\mathcal{A}$. In turn, $?\mathcal{P} + \Omega$ is the weakest from $\mathcal{A}$ to solve NBAC in $\mathcal{E}$. Clearly, $\Omega \preceq_{\mathcal{E}_f} ?\mathcal{P} + \Omega$. However, Ω cannot be transformed into $?\mathcal{P} + \Omega$ ($?\mathcal{P}$ does not make mistakes while Ω is allowed to do so [6]). Hence, $\Omega \prec_{\mathcal{E}_f} ?\mathcal{P} + \Omega$.

Thus, in $\mathcal{E}_f$, any algorithm that solves NBAC using $\mathcal{A}$ can be transformed into an algorithm that solves Consensus, while the converse transformation is not possible. In other words, NBAC is strictly harder than Consensus in $\mathcal{E}_f$ with a majority of correct processes in a system augmented with timeless failure detectors. □

8 Concluding remarks

Sabel and Marzullo showed in [7] that $\mathcal{P}$ is the weakest failure detector to solve the Leader Election problem within a specific class of failure detectors. They focus on failure detectors that output sets of suspected processes and satisfy the following symmetry property: if a process detects a failure erroneously, then any process can detect a failure erroneously an arbitrary number of times. The requirement is rather strong: for instance, it excludes all failure detectors that make a finite number of mistakes. The approach is somewhat similar to ours. We also defined a subset $\mathcal{A}$ of the overall universe of failure detectors [2] in which $?\mathcal{P} + \Omega \prec \mathcal{P}$ is shown to be the weakest to solve our NBAC problem. The class of *symmetric* failure detectors of [7] and our class $\mathcal{A}$ of timeless failure detectors are however incomparable.

Fromentin, Raynal and Tronel stated in [4] that $\mathcal{P}$ is the weakest failure detector to solve NBAC. Guerraoui [6] pointed out that [4] assumes NBAC to be solved among any subset of the processes in the system and showed that $\mathcal{P}$ is not the weakest failure detector to solve NBAC without that assumption. In this paper, we make a step further showing that a failure detector $?\mathcal{P} + \Omega \prec \mathcal{P}$ is the weakest to solve NBAC in a wide class $\mathcal{A}$ of timeless failure detectors (provided an environment with a majority of correct processes). Thus, in this environment, NBAC is strictly harder than Consensus (which is not true in general [5, 6]). The question of the weakest failure detector to solve NBAC without assuming a majority of correct processes is open for future research.

References

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

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

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(3):374–382, April 1985.

[4] E. Fromentin, M. Raynal, and F. Tronel. On classes of problems in asynchronous distributed systems with process crashes. In *Proceedings of the IEEE International Conference on Distributed Systems (ICDCS)*, pages 470–477, 1999.

[5] R. Guerraoui. On the hardness of failure-sensitive agreement problems. *Information Processing Letters*, 79(2):99–104, June 2001.

[6] R. Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15:17–25, January 2002.

[7] L. S. Sabel and K. Marzullo. Election vs. consensus in asynchronous systems. Technical report, Cornell University, Ithaca, NY, TR95-1488, 1995.

[8] D. Skeen. NonBlocking commit protocols. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 133–142. ACM Press, May 1981.