Anonymous Asynchronous Systems: The Case of Failure Detectors

François Bonnet, Michel Raynal

▶ To cite this version:

François Bonnet, Michel Raynal. Anonymous Asynchronous Systems: The Case of Failure Detectors. [Research Report] PI 1945, 2010, pp.16. inria-00452799

HAL Id: inria-00452799

https://inria.hal.science/inria-00452799

Submitted on 3 Feb 2010

Anonymous Asynchronous Systems:
The Case of Failure Detectors

François Bonnet[*] , Michel Raynal[**]
francois.bonnet@irisa.fr, raynal@irisa.fr

Abstract: Due the multiplicity of loci of control, a main issue distributed systems have to cope with lies in the uncertainty on the system state created by the adversaries that are asynchrony, failures, dynamicity, mobility, etc. Considering message-passing systems, this paper considers the uncertainty created by the net effect of three of these adversaries, namely, asynchrony, failures, and anonymity. This means that, in addition to be asynchronous and crash-prone, the processes have no identity.

Trivially, agreement problems (e.g., consensus) that cannot be solved in presence of asynchrony and failures cannot be solved either when adding anonymity. The paper consequently proposes anonymous failure detectors to circumvent these impossibilities. It has several contributions. First it presents three classes of failure detectors (denoted AP, $A\Omega$ and $A\Sigma$) and show that they are the anonymous counterparts of the classes of perfect failure detectors, eventual leader failure detectors and quorum failure detectors, respectively. The class $A\Sigma$ is new and showing it is the anonymous counterpart of the class Σ is not trivial. Then, the paper presents and proves correct a genuinely anonymous consensus algorithm based on the pair of anonymous failure detector classes $(A\Omega, A\Sigma)$ ("genuinely" means that, not only processes have no identity, but no process is aware of the total number of processes). This new algorithm is not a "straightforward extension" of an algorithm designed for non-anonymous systems. To benefit from $A\Sigma$, it uses a novel message exchange pattern where each phase of every round is made up of sub-rounds in which appropriate control information is exchanged. Finally, the paper discusses the notions of failure detector class hierarchy and weakest failure detector class for a given problem in the context of anonymous systems.

Key-words: Anonymous system, Asynchronous system, Distributed computability, Failure detector class, Fault-tolerance, Message passing system, Process crash.

Systèmes Asynchrones Anonymes:
L'Étude des Détecteurs de Fautes

Résumé : *Cet article s'intéresse aux systèmes asynchrones anonymes, et plus particulièrement aux détecteurs de fautes qui peuvent être définis et utilisés dans ce contexte. Trois classes de détecteurs de faute sont présentées: AP, AΩ, et AΣ.*

Mots clés : *Calcul réparti, Classe de détecteurs de fautes, Panne de processus, Système anonyme, Système asynchrone, Système à passage de messages, Tolérance aux pannes.*

---

[*] Projet ASAP : équipe commune avec l'INRIA, l'INSA et l'Université de Rennes 1
[**] Projet ASAP : équipe commune avec l'INRIA, l'INSA et l'Université de Rennes 1

1   Introduction

Anonymous systems   One of the main issue faced by distributed computing lies in mastering uncertainty created by the adversaries that are asynchrony and failures. As a simple example, the net effect of these adversaries makes impossible for a process to know if another process has crashed or is only very slow. Recently, new facets of uncertainty (e.g., dynamicity, mobility) have appeared and made distributed computing even more challenging.

Among the many facets of uncertainty that distributed computing has to cope with, *anonymity* is particularly important. It occurs when the computing entities (processes, agents, sensors, etc.) have no name, and consequently cannot distinguish the ones from the others. It is worth noticing that, from a practical point of view, anonymity is a first class property as soon as one is interested in guaranteeing privacy. As an example, some peer-to-peer file-sharing systems assume the peers are anonymous [12]. In the same vein, not all the sensor networks assume that each sensor has a proper identity [2, 18].

One of the very first works (to our knowledge) that addressed anonymous systems is the work of D. Angluin [1]. In that paper, considering message passing systems, Angluin was mainly interested in computability issues, namely answering the question "which functions can be computed in presence of asynchrony and anonymity?" The leader election problem is a simple example of a problem that is unsolvable in such a setting (intuitively, this because symmetry cannot be broken in presence of asynchrony and anonymity). Other works have then addressed anonymity in particular settings such as ring networks [3], or networks with a regular structure [23]. Failure-free message passing anonymous systems have also been investigated in [29, 30] where is given a characterization of problems solvable in this context according to which amount on information about network attributes are initially known by the processes.

Enriching a system with failure detectors   The failure detector-based approach [10] is one of the most popular approaches to circumvent impossibility results in non-anonymous failure-prone asynchronous systems. Roughly speaking, a failure detector is a device that provides each process with failure-related information. According to the quality of this information, several failure detector classes have been defined. As an example, let us consider the consensus problem. This problem cannot be solved in a pure asynchronous message-passing system prone to even a single process crash [19]. It is defined as follows. Each processes proposes a value, and every process that does not crash has to decide a value (termination), such that a decided value is a proposed value (validity), and no two processes decide different values (agreement). It has been shown that the failure detector class denoted Ω is the weakest failure detector class that allows consensus to be solved in message-passing asynchronous systems where a majority of processes never crash [9]. It has also been shown that the pair (Σ, Ω) is the weakest failure detector class when any number of processes may crash [14, 15]. (These failure detector classes are precisely defined later in the paper.)

Anonymous failure detectors   Recently a few failure detector classes have been proposed that, while used in non-anonymous systems, are anonymous failure detectors. The failure detector class $\mathcal{L}$ has been introduced in [16], where it is shown to be the weakest failure detector class for the $(n-1)$-set agreement problem in n-process asynchronous message-passing systems prone to any number of crashes. (The k-set agreement problem is a weakened form of consensus where up to k values can be decided [11].) A failure detector of the class $\mathcal{L}$ outputs a boolean value at each process, such that (while they can change) these values collectively satisfy some properties.

This failure detector class has been generalized in [5], where is defined the family $\{\mathcal{L}(k)\}_{1 \leq k < n}$ of failure detector classes ($\mathcal{L}(n-1)$ is $\mathcal{L}$). An $\mathcal{L}(k)$-based k-set agreement algorithm suited to anonymous systems is presented in [5].

A class of anonymously perfect failure detectors is introduced in [6] (this class is called "anonymously perfect" because it is equivalent to the class of perfect failure detectors when considering a non-anonymous system). This class is a simple adaptation to the anonymous context of a failure detector class introduced in [25, 26]. It is shown in [6] that anonymity has a price in an asynchronous anonymous system prone to up to t process crashes enriched with such a failure detector, namely, consensus requires $2t + 1$ rounds (in a non-anonymous system enriched with a perfect failure detector, the lower bound on round numbers is $t + 1$).

A consensus algorithm for anonymous systems is presented in [17]. Instead of enriching the anonymous system with a failure detector, this paper assumes a round-based system that satisfies partial synchrony assumptions (an anonymous variant of the GIRAF framework [22]).

Content of the paper   This paper is on failure detectors for anonymous asynchronous message-passing systems prone to any number of process crashes. It has several contributions.

- It first introduces the class of anonymous failure detector class $A\Sigma$ and shows that it is the anonymous counterpart of the class Σ of quorum failure detectors.

  While Σ provides each process with a set of process identities that satisfies some intersection property, the main issue encountered in defining its anonymous counterpart $A\Sigma$ lies in capturing properties on set cardinals from which an

intersection property can be extracted. This capture is not trivial, as demonstrated by the construction of Σ from $A\Sigma$ in a non-anonymous system which is particularly subtle. (Contrarily, the construction of $A\Sigma$ from Σ is simple.)

The paper also presents the anonymous failure detector classes AP and $A\Omega$ which are the anonymous counterparts of the classes P (perfect failure detectors) and Ω (eventual leader failure detectors), respectively. While the classes have already been presented in the literature, a novel bounded quiescent construction of a failure detector of the class P from AP in a non-anonymous system is presented.

- The paper presents then an algorithm, based on the pair of failure detector classes $A\Sigma$ and $A\Omega$, that solves the consensus problem in an anonymous system. This algorithm is "genuinely anonymous" in the sense that, not only the processes have no identity, but none of them know the total number n of processes.

  This algorithm adopts the structure of the non-anonymous consensus algorithm presented in [28]: processes execute asynchronous rounds and each round is made up of three communication phases. While in a non-anonymous system, the use of quorums [27] allows a process to broadcast a message and then wait for messages from a given set of processes, this is no longer possible in an anonymous system. To solve this problem, the proposed algorithm is based on a novel message exchange pattern in which each phase of a round is composed of a finite number of sub-rounds.

  The AP-based consensus algorithm presented in [6] and the $\mathcal{L}(1)$-based consensus algorithm presented in [5] are not genuine. Moreover, while the pair $(A\Sigma, A\Omega)$ is equivalent to the pair (Σ, Ω) in a non-anonymous system (this pair has been shown to be the weakest failure detector class for solving consensus despite any number of process crashes [14, 15]), $\mathcal{L}(1)$ is strictly stronger than the pair (Σ, Ω) in a non-anonymous system [5, 7].

- The paper finally discusses the notion of hierarchy for classes of anonymous failure detectors. As an example, while the class P is strictly stronger than the class Ω in non-anonymous systems, it appears that their anonymous counterparts AP and $A\Omega$ are incomparable. The paper addresses also the issue of the "weakest failure detector class" for the consensus problem in an anonymous context.

Roadmap   The paper is made up of 8 sections. The anonymous distributed computation model is presented Section 2. Section 3 presents the three classes of anonymous failure detectors AP, $A\Omega$ and $A\Sigma$. Section 4 shows that $A\Sigma$ and Σ are equivalent in non-anonymous systems. Section 5 discusses the notion of failure detector class hierarchy for anonymous systems. Section 6 presents a genuinely anonymous consensus algorithm based on the pair of failure detector classes $A\Sigma$ and $A\Omega$. Section 7 discusses the "weakest failure detector class" issue for anonymous consensus Finally, Section 8 concludes the paper.

2   Base computation model: anonymous message passing system

Process model   The system is made up of a fixed number n of processes, denoted $p_1, \ldots, p_n$. A process p_i does not know its index i, which means that indexes are only used for a presentation purpose. $\Pi = \{1, \ldots, n\}$ denote the set of process indexes. Processes are anonymous in the sense that they have no name (they do not know their index), and execute the same algorithm. They are asynchronous in the sense that there is no assumption on their respective speeds.

The underlying time model is the set of positive integers (denoted $\mathbb{N}$). Time instants are denoted τ, τ', etc. This time notion is not accessible to the processes. It can only be used from an external observer point of view to state or prove properties.

Failure model   A process executes correctly its algorithm until it possibly crashes. A crash is a premature stop; after it has crashed, a process executes no step. A process that does not crash in a run is *correct* in that run. Otherwise, it is *faulty* in that run. Until it crashes (if ever it does), a process is *alive*. Given a run, *Correct* denotes the the set of processes that are correct in that run.

An environment is a set of failure patterns, where a *failure pattern* is a function $F()$ such that $F(\tau)$ denotes the set of processes that have crashed by time τ. We consider here failure patterns in which all (but one) processes may crash in a run. This set of failure patterns is called *wait-free environment*.

Communication   The processes communicate by exchanging messages through reliable channels. These channels are asynchronous, which means that there is no assumption on message transit delays, except that they are positive and finite (every message eventually arrives).

The processes are provided with a broadcast() communication primitive that allows the invoking process to send the same message to all the processes (including itself). The broadcast() primitive is not reliable in the sense that, if a process p_i crashes while broadcasting a message, that message can be received by an arbitrary subset of processes. When it receives messages, a process cannot determine which are their senders.

Notation The previous computation model is denoted $\mathcal{AAS}[\emptyset]$. $\mathcal{AAS}$ is an acronym for *Anonymous Asynchronous System*; $\emptyset$ means that there is no additional assumption.

$\mathcal{AS}[\emptyset]$ is used to denote the non-anonymous counterpart of $\mathcal{AAS}[\emptyset]$, i.e., an asynchronous message passing system prone to any number of crash failures and where each process has a distinct name and each process knows all process names [4, 24].

Let A and B be two failure detectors classes. In the following $\mathcal{AAS}[A, B]$ denotes the system $\mathcal{AAS}[\emptyset]$ enriched with a failure of the class A and a failure detector of the class B. This means that any process can additionally read the local variables provided by these failure detectors. $\mathcal{AAS}[A]$ denotes a system where only the failure detector A can be accessed.

3 Anonymous failure detector classes

3.1 The class AP of anonymous perfect failure detectors

The class P of perfect failure detectors This class of failure detectors (introduced in [10]) assumes that the processes have distinct identities, and those are known by all processes. A failure detector of the class P provides each process p_i with a read-only variable, denoted $suspected_i$, that is a set that never contains the identity of an alive process, and eventually contains the identities of all the faulty processes.

The class AP of anonymous perfect failure detectors This failure detector class is a variant of a class introduced in [25, 26]. Let f^τ denote the number of processes that have crashed up to time τ, and f denote the actual number of processes that crash. A failure detector of the class AP provides each process p_i with a read-only integer variable denoted $aa\ell_i$ (*a*pproximate number of *a*live processes) that satisfies the following properties ($aa\ell_i^\tau$ denotes the value of $aa\ell_i$ at time τ).

- Safety: $\forall \tau \in \mathbb{N} : \forall i \in \Pi \setminus F(\tau) : aa\ell_i^\tau \geq n - f^\tau$.

- Liveness: $\forall i \in Correct : \exists \tau \in \mathbb{N} : \forall \tau' \geq \tau : aa\ell_i^{\tau'} = n - f$.

The safety property states that $aa\ell_i$ is always an over-estimate of the number of processes that are still alive, while the liveness property states that it eventually converges to its exact value.

3.2 P and AP are equivalent in non-anonymous systems

This section shows that the classes AP and P are equivalent when we consider a non-anonymous system. Let us remember that, in such a system, each process has its own identity, and the identities are knows by all processes. Without loss of generality we assume that the identity of p_i is its index i.

In the following, $\mathcal{AS}_{n,t}[A]$ denotes a non-anonymous asynchronous system where up to $t < n$ processes may crash. Taking $t = n - 1$ provides us with the wait-free environment (failure patterns in which all but one processes may crash).

Building AP in $\mathcal{AS}_{n,t}[P]$ The transformation in that direction is trivial. The reader can easily check that, whatever the value of t, taking the current value $n - |suspected_i|$ to define the current value of $aa\ell_i$, constructs a failure detector of the class AP.

```
Init: k_i ← 0; ans_i ← [0, · · · , 0]; suspected_i ← ∅.

(1)  T1: repeat wait until (n − aaℓ_i > k_i);
(2)              k_i ← n − aaℓ_i; brodcast INQUIRY(k_i)
(3)         until (k_i = t) end repeat.

(4)  T2: when INQUIRY(k) is received from p_j: send ALIVE(k) to p_j.

(5)  T3: when ALIVE(k) is received from p_j: ans_i[j] ← max(ans_i[j], k).

(6)  T4: repeat m ← k_i; % m is local to T4, while k_i is not %
(7)              X ← {x  such that ans_i[x] ≥ m};
(8)              if (|X| = n − m) then suspected_i ← {1, . . . , n} \ X end if
(9)         until (|suspected_i| = t) end repeat.
```

Figure 1: Building P in $\mathcal{AS}_{n,t}[AP]$: a bounded transformation (code for p_i)

Building P in $\mathcal{AS}_{n,t}[AP]$ A transformation that builds a failure detector of the class P in $\mathcal{AS}[AP]$ is described in Figure 1. Interestingly, this transformation is bounded (be the execution finite or infinite, the local memory of each process requires only a bounded number of bits). Moreover, (1) the transformation is quiescent (i.e., there is a finite time after which no more messages are exchanged), and (2) the algorithm terminates in the runs where t processes crash.

In order to compute the value of $suspected_i$ (that is initialized to $\emptyset$), each process p_i manages two local variables:

- An integer k_i, initialized to 0, that represents its current knowledge on the number of processes that have crashed.

- An array $ans_i[1..n]$, initialized to $[0, \cdots, 0]$, such that $ans_i[j] = k$ means that k is the greatest inquiry number for which p_i has received the corresponding answer ALIVE (k).

The behavior of p_i is defined by four tasks. First, when p_i discovers there are more than k_i processes that have crashed, it updates accordingly k_i, and broadcasts an inquiry message INQUIRY (k_i) to all the processes. Let us notice that this task can stop when $k_i = t$ as, due to the model definition, no more crash can occur. Let us also observe that the messages INQUIRY(k_i) are sent by p_i with increasing values, and due to the strong accuracy property of $aa\ell_i$, p_i knows that there are at most $n - k_i$ alive processes.

When p_i receives an INQUIRY (k) message from a process p_j it sends back to p_j an ALIVE (k) message to indicate that it is still alive. When it receives an answer ALIVE (k) from a process p_j, p_i learns that p_j has answered up to its k-th inquiry, and consequently updates $ans_i[j]$.

The core of the transformation is the task $T4$ that gives its current value to $suspected_i$. It is made up of a **repeat** statement that is executed until t processes are locally suspected. (When t processes have crashed, no more processes can crash and the task can terminate. If less than t processes crash, the task becomes quiescent -no more messages are sent- but does not terminate.)

The body of the **repeat** statement is as follows. First, p_i sets a local variable m to k_i (the number of processes that, to the best of its knowledge, have crashed). Then, p_i computes the set X made up of the processes that have answered its m-th inquiry or a more recent one. If the predicate $|X| = n - m$ is true, p_i can safely conclude that the $n - m$ processes that have answered its m-th inquiry were alive when they answered, which means that the m processes that have not answered have crashed and are exactly the ones in the set $\Pi \setminus X$ (let us recall that, while the tasks $T1$ and $T4$ proceed asynchronously, p_i broadcasts INQUIRY (m) only after it knows that m processes have crashed).

Theorem 1 *The algorithm described in Figure 1 is a bounded quiescent construction of a failure detector of the class P in $\mathcal{AS}_{n,t}[AP]$.*

Proof   Proof of the completeness property of P. Let us assume that p_i is a non-faulty process. We have to show that if a process p_j crashes, after some finite time, j permanently belongs to $suspected_i$. Let $f = |Faulty(F)|$.

There is a finite time τ, after which the f faulty processes have crashed and we have permanently $aa\ell_i = n - f$, which means that, after some finite time, p_i broadcasts a message INQUIRY(f). Due to the strong accuracy property of AP, this message is sent after the f processes have crashed. Consequently, no crashed process can answer this inquiry message. It follows that, when task $T4$ executes with $m = k_i = f$, the set X can only contain the $n - f$ non-faulty processes, and we have then $|X| = n - f = n - m$. Hence, $suspected_i$ is set to $\{1, \ldots, n\} \setminus X$, i.e., contains exactly the f faulty processes, which concludes the proof of the completeness property.

Proof of the strong accuracy property of P. Let p_i be any process. We have to show that no process is added to $suspected_i$ before crashing. Let $i_1, \ldots, i_m$ be the m process identities that are placed in $suspected_i$ during an iteration of task $T4$. It follows from the query/response mechanism (implemented by the INQUIRY/ALIVE messages) used when $k_i = m$, and the strong accuracy property of AP, that each of the $n - m$ other processes has answered after these m processes have crashed. Consequently, none of these $n - m$ processes can be part of the m crashed processes. Hence, the set of processes that defines the value of $suspected_i$ contains only crashed processes.

The fact that the construction is bounded and quiescent follows directly from the text of the algorithm: a process broadcast at most one INQUIRY(k) message for every value of k, and k can take a bounded number of values. $\square_{Theorem\ 1}$

An algorithm solving the consensus problem in $\mathcal{AAS}[AP]$: is described in [6], where it is proved that $(2t + 1)$ rounds is a lower bound on the number of rounds for solving consensus in such a system model.

3.3   The class $A\Omega$ of anonymous eventual leader failure detectors

The class Ω of eventual leader failure detectors   The class of (non-anonymous) eventual leader failure detectors Ω has been introduced in [9]. It provides each process p_i with a local variable $leader_i$ that contains a process identity and is such that, after an arbitrary but finite time, the variables $leader_i$ of the non-faulty processes contain forever the same identity, and this identity is the one of a non-faulty process.

The class $A\Omega$ of anonymous eventual leader failure detectors   It is easy to define an anonymous counterpart of Ω. This class of failure detectors, denoted $A\Omega$, provides every process p_i with a boolean variable a_leader_i such that, after an arbitrary but finite time, there is one non-faulty process (say p_ℓ) whose boolean variable remains forever true, and the boolean variables of the other non-faulty processes remain forever false. Let us notice that there is an arbitrary long anarchy period during which the local variables a_leader_i can take arbitrary values (e.g., it is possible that they all are equal to *false*).

3.4 Ω and $A\Omega$ are equivalent in non-anonymous systems

Ω and $A\Omega$ are equivalent in $\mathcal{AS}[\emptyset]$. The two directions of the equivalence are explained below. The proofs are straightforward and left to the reader.

Building $A\Omega$ in $\mathcal{AS}[\Omega]$ For any process p_i, the current value of the boolean variable a_leader_i of $A\Omega$ is computed by the test $leader_i = i$ where $leader_i$ is the output of Ω.

Building Ω in $\mathcal{AS}[A\Omega]$ The reduction consists in two tasks executed by all processes: (1) Each process p_i checks periodically its boolean a_leader_i and if its value is true, it broadcasts a message LEADER(i) (note that this reduction is done in the non-anonymous model, hence p_i knows its identity). (2) When p_i receives a message LEADER(k), it updates its $leader_i$ to k.

3.5   The class $A\Sigma$ of anonymous quorum failure detectors

The class Σ of non-anonymous quorum failure detectors   The notion of quorum has been introduced in [20] (and explicitly used to solved consensus in [27]). The *quorum failure detector* class has been introduced and investigated in [14]. Each process p_i is provided with a local variable (denoted $sigma_i$) that it can only read. At any time, this variable contains a set of process identities (quorum). Let $sigma_i^\tau$ be the value of $sigma_i$ at time τ. By definition, $sigma_i^\tau = \Pi$ when $i \in F(\tau)$. The class Σ contains all the failure detectors that satisfy the following properties.

- Safety property. $\forall i, j \in \Pi$: $\forall \tau, \tau'\mathbb{N}$: $sigma_i^\tau \cap sigma_j^{\tau'} \neq \emptyset$.

- Liveness property. $\forall i \in Correct$: $\exists\tau$: $\forall \tau' \geq \tau$: $sigma_i^{\tau'} \subseteq Correct$.

The first property states that the values of any two quorums taken at any times do intersect. This property prevent partitioning and is consequently used to maintain consistency. The second property states that a quorum cannot block the process that uses it. (Because two majorities always intersect, it is easy to see that $sigma$ can be implemented - despite asynchrony- in the environments where less than $n/2$ processes may crash. Differently, it cannot be implemented in environments where $n/2$ or more processes can crash.)

It is shown in [14] that Σ is the weakest class of failure detectors to implement a register in an asynchronous message-passing system prone to any number of process crashes. A simple proof of this result appears in [8].

The class $A\Sigma$ of anonymous quorum failure detectors   The class of anonymous quorum failure detectors is denoted $A\Sigma$. Any failure detector of that class provides each process with a read-only local variable a_sigma_i that contains pairs. Each pair is composed of a label x and an integer y. Without loss of generality, the set of labels is assumed to be a subset of the set $\mathbb{N}$. The intuition is the following. If $(x, y) \in a\_sigma_i$, $A\Sigma$ has informed process p_i of (1) the existence of label x and (2) the fact that y processes are assumed to know it. As, we will see, a quorum is a set of processes that know the same label.

Formally, the behavior of the local variables $\{a\_sigma_i\}_{1 \leq i \leq n}$ is defined by the following properties. The first two properties (validity and monotonicity) are well-formedness properties, while the last two properties (safety an liveness) are behavioral properties.

Formal definition   The formal definition of $A\Sigma$ is as follows.

- Validity. $\forall i \in \Pi$: $\forall \tau \in \mathbb{N}$: $a\_sigma_i^\tau = \{(x_1, y_1), \ldots, (x_p, y_p)\}$ where $\forall 1 \leq a, b \leq p : (x_a, y_b \in \mathbb{N}) \wedge \big((a \neq b) \Rightarrow (x_a \neq x_b)\big)$.

- Monotonicity. $\forall i \in \Pi$: $\forall\tau \in \mathbb{N}$: $(x, y) \in a\_sigma_i^\tau \Rightarrow \big(\forall\tau' \geq \tau : (x, y') \in a\_sigma_i^{\tau'}$ with $y' \leq y\big)$.

Definition 1 $S(x) = \{i \mid \exists\tau \in \mathbb{N} : (x, -) \in a\_sigma_i^\tau\}$.

- Liveness. $\forall i \in Correct : \exists(x, y)$: $\exists \tau : \forall \tau' \geq \tau$: $\big((x, y) \in a\_sigma_i^{\tau'}\big) \wedge \big(|S(x) \cap Correct| \geq y\big)$.

- Safety. $\forall i_1, i_2 \in \Pi$: $\forall \tau_1, \tau_2 \in \mathbb{N}$: $\forall (x_1, y_1) \in a\_sigma_{i_1}^{\tau_1}$: $\forall (x_2, y_2) \in a\_sigma_{i_2}^{\tau_2}$:
  $$\forall T_1 \subseteq S(x_1): \forall T_2 \subseteq S(x_2): \quad \big((|T_1| = y_1) \wedge (|T_2| = y_2)\big) \Rightarrow (T_1 \cap T_2 \neq \emptyset).$$

Interpretation The validity property expresses the fact that, at any time, a_sigma_i is a non-empty set of pairs (x, y) where x is a label and y a number of processes associated with this label (those are processes assumed to know the label x). A label can appear at most once in a_sigma_i, but the number of distinct labels that can appear in a_sigma_i is arbitrary.

The monotonicity property states that the number y of processes associated with a label x, as known by p_i, can only decrease. This requirement is not necessary but makes things simpler. Not considering this monotonicity property will not change our results but would make them more difficult to understand and proofs more technical. Hence, this property has to be seen as a "comfort" property, and not as a "computability" property.

$S(x)$ is the set of all processes that know the label x. While a process p_i knows it belongs to $S(x)$, it does not know the value of $S(x)$.

The next property is called liveness because it is used to prove liveness of $A\Sigma$-based algorithms (and similarly for the safety property). It captures the fact that, after some time, a quorum contains only correct processes, thereby preventing a correct process from blocking forever if it uses that quorum. To that end, this property states that, for any correct process p_i, there is eventually a label x such that its associated number y of processes remains always smaller or equal to the number of correct processes in $S(x)$. (The underlying intuition is that a label of any correct process will be forever associated with correct processes only.)

The safety property is a little bit more involved. It captures the intersection property associated with quorums. Let x_1 and x_2 be two labels known by p_{i_1} and p_{i_2} respectively, T_1 any subset of $S(x_1)$, T_2 any subset of $S(x_2)$ (let us remember that $S(x)$ is the set of all the processes that know label x). The safety property states the following: if $|T_1| = y_1$ and $|T_2| = y_2$, where $(x_1, y_1) \in a\_sigma_{i_1}$ and $(x_2, y_2) \in a\_sigma_{i_2}$, then $T_1 \cap T_2 \neq \emptyset$. (Let us remember that y_1 is the number of processes associated with label x_1 as know by p_{i_1} (and similarly for y_2). The intuition is that the y_1 processes that know label x_1 and the set of y_2 processes that know label x_2 do intersect.

Remark By its very definition, $A\Sigma$ is indeed anonymous (process indexes appear only in quantifiers). Moreover, it is possible to have (1) $(x, y) \in a\_sigma_i$ and $(x, y') \in a\_sigma_{i'}$ with $i \neq i'$ and $y \neq y'$, and (2) $(x, y) \in a\_sigma_i$ with $|S(x)| < y$.

4 Σ and $A\Sigma$ are equivalent in non-anonymous systems

This section shows that, in a non-anonymous system, Σ and $A\Sigma$ are equivalent whatever the failure pattern (wait-free environment). In this section, a label is a quorum name, hence "quorum name" and "label" are used as synonym.

4.1 Building $A\Sigma$ in $\mathcal{AS}[\Sigma]$

Preliminary definitions As $\mathcal{AS}[\Sigma]$ is not anonymous, it is possible for the processes to (statically) build all the possible subsets Q, such that $Q \neq \emptyset$ and $Q \subseteq \{1, \ldots, n\}$. There are $2^n - 1$ such subsets. Moreover, the processes are provided with a deterministic function denoted name(). That function associates a label with each set Q, and satisfies the following properties: (a) $\forall Q$: name$(Q) \in [1..2^n - 1]$, and (b) $(Q \neq Q') \Rightarrow (\text{name}(Q) \neq \text{name}(Q'))$.

The construction The algorithm building a failure detector of the class $A\Sigma$ in $\mathcal{AS}[\Sigma]$ is described in Figure 2. Interestingly, this construction is bounded and quiescent. It builds, at each process p_i, a set a_sigma_i that contains pairs of integers, and ensures that these sets of pairs satisfy the properties defining the class $A\Sigma$. The algorithm is made up of two tasks that are executed at each process p_i.

- Task $T1$ repeatedly broadcasts the local output $sigma_i$ of the underlying failure detector Σ. In order to ensure the boundedness and quiescence properties, a local set variable $sent_i$ (a set of sets) is used to prevent the same quorum to be sent several times.

- Task $T2$ is associated with the reception of messages QUORUM($quorum$). When such a message is received, p_i adds the pair (name($quorum$), $|quorum|$) to a_sigma_i if $i \in quorum$. Otherwise, p_i discards the message.

Theorem 2 *The algorithm described in Figure 2 is a bounded quiescent construction in $\mathcal{AS}[\Sigma]$ of a failure detector of the class $A\Sigma$.*

Proof The validity property follows immediately from the definition of the function name(). Moreover since name() is a one-to-one function, there is at most one pair $(x, -)$ associated with a given x. The monotonicity property is then obvious.

Liveness property. We have to show that $\forall i \in Correct : \exists (x, y): \exists \tau : \forall \tau' \geq \tau: (x, y) \in a\_sigma_i^{\tau'} \land |S(x) \cap Correct| \geq y$. To that end, let us consider a time instant τ_0 after which all faulty processes have crashed and their messages have been

```
Init: a_sigma_i ← {(name(Π), |Π|)}; sent = {Π};
(1) T1: repeat forever
(2)         quorum_i ← sigma_i;
(3)         if (quorum_i ∉ sent_i) then broadcast QUORUM(quorum_i); sent_i ← sent ∪ {quorum_i} end if
(4)     end repeat.
(5) T2: when QUORUM(quorum) is received:
(6)         if (i ∈ quorum) then a_sigma_i ← a_sigma_i ∪ {(name(quorum), |quorum|)} end if.
```

Figure 2: Building $A\Sigma$ in $\mathcal{AS}[\Sigma]$: a bounded quiescent transformation (code for p_i)

received and processed. Moreover, let τ_1 be a time instant, such that, $\forall \tau_1' \geq \tau_1$, $sigma_i^{\tau_1'}$ contains only correct processes (due to the liveness property of Σ, τ_1 does exist). Finally, let $\tau \geq \max(\tau_0, \tau_1)$.

All the processes that execute after τ are correct. Let $i \in Correct$. Let $quorum$ be any set obtained by p_i after τ. As $\tau \geq \tau_1$, $quorum$ contains only correct processes, i.e., $quorum \subseteq Correct$ (Observation O1).

As p_i is correct, any correct process p_j receives QUORUM($quorum$) and processes it (if not yet done). From observation O1, all processes of $quorum$ receive QUORUM($quorum$). Let us consider a process p_j such that $j \in quorum$. If not yet done, each such p_j adds $(\mathsf{name}(quorum), |quorum|)$ to a_sigma_j. It follows that $j \in S(x)$. Moreover, it follows from the text of the algorithm that no process p_k outside the set $quorum$ adds the pair $(\mathsf{name}(quorum), |quorum|)$ to a_sigma_k. Consequently we have $S(\mathsf{name}(quorum)) = quorum$ (Observation O2). It follows from observations O1 and O2 that, for any correct process p_i, there is a pair $(x, y) = (\mathsf{name}(quorum), |quorum|)$ such that $|S(x) \cap Correct| = |quorum| = y$, which completes the proof of the liveness property.

Safety property. Let $quorum_{i_1}$ and $quorum_{i_2}$ be two quorums obtained (at line 2) by two processes p_{i_1} and p_{i_2} at the time instants τ_{i_1} and τ_{i_2}, respectively.

Let $(x_1, y_1) = (\mathsf{name}(quorum_{i_1}), |quorum_{i_1}|)$. As previously stated, only processes that can belong to $S(x_1)$ (by adding the pair (x_1, y_1) to their set a_sigma) are processes of $quorum_{i_1}$, which means that $S(x_1) \subseteq quorum_{i_1}$. As defined in the safety property of $A\Sigma$, let $T_1 \subseteq S(x_1)$ such that $|T_1| = y_1$. Let us notice that, if such a set T_1 exists, we have $T_1 = S(x_1) = quorum_{i_1}$. With similar definitions and observations for $quorum_{i_2}$, if a set T_2 exists, we have $T_2 = S(x_2) = quorum_{i_2}$.

Thus we have $T_1 = quorum_{i_1} = sigma_{i_1}^{\tau_{i_1}}$, and $T_2 = quorum_{i_2} = sigma_{i_2}^{\tau_{i_2}}$. It follows from the safety property of Σ that $sigma_{i_1}^{\tau_{i_1}} \cap sigma_{i_2}^{\tau_{i_2}} \neq \emptyset$. Consequently, $T_1 \cap T_2 \neq \emptyset$, which concludes the proof of the safety property.

Boundedness and quiescence. As there is a bounded number of processes, the number of possible quorums output by a failure detector Σ is clearly bounded. It follows that both all the sets a_sigma and $sent$ are bounded. Moreover, since each process broadcasts a quorum at most once, not only the content but also the number of messages is bounded, from which follows the quiescence property. $\square_{Theorem\ 2}$

Remark The $\mathcal{AS}[\Sigma]$ model assumes that the communication channels are reliable in the sense that there is no loss, no duplication, and no creation of messages. Actually, the reader can check that the previous construction is not only bounded and quiescent, but remains correct when messages are finitely duplicated.

4.2   Building Σ in $\mathcal{AS}[A\Sigma]$

The construction The algorithm that builds a failure detector of the class Σ in $\mathcal{AS}[A\Sigma]$ is described in Figure 3. It relies on two main data structures at each process p_i.

- $alive_i$ is a queue, always containing all the process indexes, that is managed as follows. When p_i receives a message from p_j, it reorders j and places it at the head of that queue. In that way, the processes that are alive (i.e., those that send messages) appear at the head of $alive_i$, while the processes that have crashed are progressively moved at its tail.

- $queue_i$ is an array of queues, such that $queue_i[x]$ contains the indexes of the processes that, from p_i's point of view, know the quorum whose name is x. The quorum names x are obtained from the local output a_sigma_i supplied by the underlying failure detector of the class $A\Sigma$.

According to these data structures, the behavior of a process p_i is made up of three tasks.

- Task $T1$ is an infinite loop in which p_i repeatedly broadcasts a message ALIVE(i, $labels_i$) that contains the names of the quorums it knows (i.e., those that appear in a_sigma_i).

- Task $T2$ is the matching task of $T1$. When p_i receives ALIVE($j, labels$), it first updates $alive_i$ accordingly (line 6). Then, for each quorum name it knows (line 7), updates its current view of the processes that know x (i.e., the processes p_j that have $(x, -)$ in their a_sigma_j, lines 8-9)).

- Task $T3$ is the core of the construction. It is an infinite loop whose aim is to define the current value of $sigma_i$ (the local output of Σ). Process p_i first computes a set *candidates* that contains the pairs $(x, y) \in a\_sigma_i$ such that $|queue_i[x]| \geq y$ (line 12). Those are the pairs such that p_i has received a message ALIVE($-, \{\cdots, x, \cdots\}$) from at least y distinct processes (i.e., y processes know the label x). If the set *candidates* is empty, p_i cannot compute a non-trivial value for $sigma_i$. It consequently sets $sigma_i$ to Π (line 14). Otherwise, p_i computes a non-trivial value for $sigma_i$ from the set *candidates* (lines 15-18). To that end, rank(ℓ) is defined as the position of the index ℓ in the queue $alive_i$ (line 16).

The aim is to assign to $sigma_i$ the y processes that are at the head of $queue_i[x]$ (line 18), where the corresponding pair $(x, y) \in candidates$ is determined as follows. Using an array-like notation, the indexes in the prefix $queue_i[x][1..y]$ "globally appear in $alive_i$ before" the indexes in the other prefixes $queue_i[x'][1..y']$. "Globally appear before" means that there is an index in $queue_i[x'][1..y']$ whose rank in $alive_i$ is after the rank of any index in $queue_i[x][1..y]$. (This is formally expressed by lines 15-17.) Let us notice that several prefixes $queue_i[x][1..y]$ can globally appear as being the "first" in $alive_i$. If it is the case, any of them can be selected.

To fix the idea, let us consider the following simple example. $alive_i = [7, 1, 3, 9, 4, 8, 2, 5, 6]$, $a\_sigma_i = \{(5, 4), (7, 3), (2, 5)\}$, $queue_i[5] = [1, 3, 4, 2, 5]$, $queue_i[7] = [1, 8, 5]$, $queue_i[2] = [1, 5]$. Considering only $queue_i[5]$, $queue_i[7]$ and $queue_i[2]$, we have $candidates = \{(5, 4), (7, 3)\}$. As $queue_i[5][4] = 2$, and $queue_i[7][3] = 5$, we have $r_min = $ rank($queue_i[5][4]$) = rank(2) = 7 < rank($queue_i[7][3]$) = rank(5) = 8. Hence, $(x, y) = (5, 4)$ defines the queue prefix whose indexes are "first" in $alive_i$. Consequently $sigma_i$ is set to $queue_i[5][1..4] = \{1, 3, 4, 2\}$.

Init: $alive_i \leftarrow$ all process indexes in arbitrary order;
      for each x **do** $queue_i[x] \leftarrow$ empty queue **end for**.

(1) $T1$: **repeat forever**
(2) $labels_i \leftarrow \{x \mid (x, -) \in a\_sigma_i\}$;
(3)   broadcast ALIVE($i, labels_i$)
(4)   **end repeat**.

(5) $T2$: **when** ALIVE($j, labels$) **is received**:
(6)   suppress j from $alive_i$; enqueue j at the head of $alive_i$;
(7)   **for each** $x \in labels$ **such that** $\big((x, -) \in a\_sigma_i\big)$ **do**
(8)   **if** ($j \in queue_i[x]$) **then** suppress j from $queue_i[x]$ **end if**;
(9)   enqueue j at the head of $queue_i[x]$
(10)   **end for**.

(11) $T3$: **repeat forever**
(12)   **let** $candidates = \{ (x, y) \mid (x, y) \in a\_sigma_i \land |queue_i[x]| \geq y \}$;
(13)   **if** ($candidates = \emptyset$)
(14)   **then** $sigma_i \leftarrow \{1, \ldots, n\}$
(15)   **else** **let** $r\_min = \min_{(x,y) \in candidates} \big(\text{rank}(queue_i[x][y])\big)$
(16)   **where** rank(ℓ) = rank of ℓ in the queue $alive_i$;
(17)   **let** $(x, y) \in candidates$ **such that** rank($queue_i[x][y]$) = r_min;
(18) $sigma_i \leftarrow$ the first y elements of $queue_i[x]$
(19)   **end if**
(20)   **end repeat**.

Figure 3: Building Σ in $\mathcal{AS}[A\Sigma]$ (code for p_i)

Theorem 3 *The algorithm described in Figure 3 builds a failure detector of the class Σ in $\mathcal{AS}[A\Sigma]$.*

Proof *Safety property.* We have to show that $\forall\, i, j \in \Pi$: $\forall\, \tau, \tau' \mathbb{N}$: $sigma_i^\tau \cap sigma_j^{\tau'} \neq \emptyset$.

Let us first observe that a set assigned to $sigma_i$ is never empty (lines 14 and 18). When the set Π is the value of $sigma_i$ (line 14, the safety property is trivially satisfied. Hence, let us consider two processes p_{i_1} and p_{i_2}, such that the values of $sigma_{i_1}$ and $sigma_{i_2}$ have been computed at any time instants τ_1 and τ_2, respectively (at lines lines 15-18).

We have then the following. The value of $sigma_{i_1}^{\tau_1}$, obtained from some pair (x_1, y_1), is the value of some set $T_1 = queue_i[x_1][1..y_1]$. It follows from the definition of $S(x_1)$, line 2, and lines 7-9 that $T_1 = queue_i[x_1][1..y_1] \subseteq S(x_1)$. Similarly, the value of $sigma_{i_2}^{\tau_2}$ is obtained from some pair (x_2, y_2), is the value of some set $T_2 = queue_i[x_2][1..y_2]$, and it follows from

the definition of $S(x_2)$, line 2, and lines 7-9 that $T_2 = queue_i[x_2][1..y_2] \subseteq S(x_2)$. It then follows directly from the safety property of $A\Sigma$ that $T_1 \cap T_2 \neq \emptyset$, and consequently, $sigma_{i_1}^{\tau_1} \cap sigma_{i_2}^{\tau_2} \neq \emptyset$, which concludes the proof of the safety property of Σ.

Liveness property. We have to show that $\forall i \in Correct$: $\exists \tau$: $\forall \ \tau' \geq \tau$: $sigma_i^{\tau'} \subseteq Correct$.

Let τ_0 be a time instant after which the faulty processes have crashed, all messages $\text{ALIVE}(-,-)$ sent by faulty processes have been received and processed, and each correct process has received a message $\text{ALIVE}(-,-)$ from each correct process after it has received all the messages from faulty processes. Let $i \in Correct$. It follows from lines 3 and 6 that, from time τ_0, the correct processes are always before the faulty processes in $alive_i$.

Moreover, due to the monotonicity and liveness properties of $A\Sigma$, there is a time τ_1 after which there is a pair $(x, y) \in a\_sigma_i$ such that we always have $|S(x) \cap Correct| \geq y$. This means that there are at least y correct processes in $S(x)$. All the time instant considered in the rest of the proof of the liveness property are time instants after $\max(\tau_0, \tau_1)$.

Let us consider a pair $(x, y) \in a\_sigma_i$ (as defined previously). As there are at least y correct processes in $S(x) \cap Correct$, each correct process p_j with $j \in S(x) \cap Correct$ broadcasts forever $\text{ALIVE}(j, labels)$ with $x \in labels$ (line 3). As each process p_i such that $i \in S(x) \cap Correct$ receives these messages, it executes lines 8-9, hence the processes in $S(x) \cap Correct$ eventually remain forever at the beginning of $queue_i[x]$, which means that we eventually have forever $|queue_i[x]| \geq |S(x) \cap Correct| \geq y$ and consequently $(x, y) \in candidates$, which means the predicate $candidates = \emptyset$ remains forever false.

As, after $\tau = \max(\tau_0, \tau_1)$, the faulty processes remain forever at the tail of $alive_i$, it follows that the pair (x', y') that is selected at lines 15-17 to define the current value of $sigma_i$, is such that the processes in $queue_i[x'][1..y']$ are not "globally after in $alive_i$" the processes in $queue_i[x][1..y]$. As all processes in $queue_i[x][1..y]$ are correct, it follows from the structure of $alive_i$, that the processes in $queue_i[x'][1..y']$ are correct. Hence, there is a time instant after which $sigma_i$ always contains correct processes, which concludes the proof of the liveness property of Σ. $\square_{Theorem\ 3}$

5   Failure detector class hierarchy: non-anonymous vs anonymous systems

Non-anonymous systems   In a non-anonymous system, as seen in the previous section: (a) P and AP are equivalent, (b) Ω and $A\Omega$ are equivalent, and (c) Σ and $A\Sigma$ are equivalent. Moreover, it is known that (in non-anonymous systems) the class P is strictly stronger than both the class Σ and the class Ω, while Σ and Ω cannot be compared [14].

Anonymous systems   In an anonymous system, AP and $A\Omega$ cannot be compared (see Theorem 4 below). This is different from what occurs in a non-anonymous system where P is strictly stronger than Ω.

Moreover, AP is strictly stronger than $A\Sigma$. The construction from AP to $A\Sigma$ is simple: a process outputs $a\_sigma_i = \{(0, aa\ell_i)\}$. For the other direction, we have the following. If it was possible to go from $A\Sigma$ to AP, we could go from Σ to P in a non-anonymous system, which is impossible [14].) The fact that AP is strictly stronger than $A\Sigma$ is consequently similar to the fact that P is strictly stronger than Σ in a non-anonymous system.

Theorem 4 *It is impossible to construct a failure detector of the class AP in $\mathcal{AAS}[A\Omega]$, and it is impossible to construct a failure detector of the class $A\Omega$ in $\mathcal{AAS}[AP]$.*

Proof   From AP to AΩ : impossibility. Let us remember that all the processes execute the same code. Whatever the code they execute, there is a run in which all the processes proceed at the same speed and read exactly the same value from their failure detector variable $aa\ell_i$. In such a run, there is no way to break the symmetry in order to distinguish a process from the other processes. It follows that a failure detector of the class $A\Omega$ cannot be built.

From AΩ to AP : impossibility. The proof is by contradiction. Let us suppose that there is an algorithm T that builds a failure detector of the class AP in $\mathcal{AAS}[A\Omega]$. By construction T does not rely on the process identities. Moreover, in a non-anonymous system, it is possible to transform Ω into $A\Omega$ (algorithm T'), and it is also possible to transform AP into P (algorithm T''). It is then possible to build a failure detector of the class P in $\mathcal{AS}[\Omega]$ as follows. (All algorithms are executed in $\mathcal{AS}[\Omega]$.)

- First, use T' to transform the failure detector $\omega \in \Omega$ into a failure detector $a\omega \in A\Omega$.

- Then, use T to transform $a\omega$ into a failure detector $ap \in AP$.

- Finally, use T'' to transform ap into a failure detector $p \in P$.

This construction contradicts the fact that it is impossible to build a failure detector of the class P in $\mathcal{AS}[\Omega]$. It follows that T cannot exist. $\square_{Theorem\ 4}$

6 Anonymous consensus in $\mathcal{AAS}[A\Sigma, A\Omega]$

This section presents and proves correct an algorithm that solves the consensus problem in $\mathcal{AAS}[A\Sigma, A\Omega]$. Interestingly, in addition of being anonymous, the processes do not need to know how many they are. The algorithm borrows its "three-phase per round" structure from the non-anonymous consensus algorithm presented in [28]. Differently from that algorithm, the message exchange pattern used inside the second and third phases is based on an entirely new principle. Moreover, due to the novelty of $A\Sigma$, the way the safety and liveness properties are ensured are also new.

6.1 Description of the algorithm

The algorithm is described in Figure 4. It is round-based: a process executes a sequence of asynchronous rounds until it decides. A process p_i invokes the operation propose(v_i) (where v_i is the value it proposes), and decides when it executes the statement return(v) (line 25 or 38, where v is the value it decides). As in other non-anonymous consensus algorithms, when a process decides it stops participating in the consensus algorithm. Consequently, before deciding a process p_i broadcasts a message DECIDE(v) in order to prevent the other processes from blocking forever (waiting for a message that p_i will never send).

The three main local variables associated with a round are r_i (the local round number), and a pair of estimates of the decision value $est1_i$ and $est2_i$. The variable $est1_i$ contains p_i's current estimate of the decision value when a new round starts while $est2_i$, whose value is computed during the second phase of every round, contains either a new estimate of the decision value or a default value $\perp$.

The behavior of a process p_i during a round r is made up of three phases, denoted phase 1, 2 and 3 which are as follows. The first phase of a round is the only one where $A\Omega$ is used, while $A\Sigma$ is used only the second and third phases of a round.

- In the first phase of a round, a process p_i that considers it is leader broadcasts a message PHASE1(r_i, v). If a_leader_i is false, p_i waits for a message PHASE1(r, v), adopts v as its new estimate and forwards PHASE1(r_i, v) to all (to prevent other processes from blocking forever in that phase of round r).

- Similarly to [27], the aim of the second phase of a round r is to assign a value to the variables $est2_i$ in such a way that the following round property is always satisfied (where $est2_i[r]$ denotes the value assigned to $est2_i$ at line 12 of round r):

$$P(r) \equiv \big[(est2_i[r] \neq \perp) \wedge (est2_j[r] \neq \perp)\big] \Rightarrow \big(est2_i[r] = est2_j[r]\big).$$

  To attain this goal, a classical non-anonymous algorithm directs a process to wait for messages from processes defining a quorum [27]. In an anonymous system, this is no longer possible as the notion of process name is outside $A\Sigma$. A process p_i can use only the pairs $(x, y) \in a\_sigma_i$, which supply no "immediately usable" information on which processes have sent messages. This issue is solved as follows. During each round, the messages broadcast by processes carry appropriate label-based information, and processes can be required to re-broadcast messages related to the very same round when this information does change.

  Hence, a process p_i first broadcasts a message PHASE2($r_i, sr_i, labels_i, est1_i$) where sr_i is a sub-round number (initialized to 1), and $labels_i$ is the set of labels it knows (line 8). As we are about to see, this information (the pair sr_i and $labels_i$) allows p_i to wait for message from an appropriate quorum of processes.

  Process p_i then enters a waiting loop (lines 9-19), that (as we will see in the proof) it eventually exits at line 10 or 13 after having assigned a value to $est2_i$. The exit at line 10 is to prevent p_i from blocking forever in phase 2 when processes have already progressed to phase 3 of the current round r_i.

  As far the exit of the repeat loop at line 13 is concerned, p_i exits when it has received "enough" (namely y) messages PHASE2($r_i, sr, labels_j, -$) (these messages carry a round number equal to r_i and the same sub-round number sr -which can be different from sr_i-) such that (a) $\exists (x, y) \in a\_sigma_i$ and (b) $x \in labels_j$ for every of the y received messages (line 11). When, this occurs, if the y messages carry the same estimate value v, p_i assigns that value v to $est2_i$, otherwise, it assigns it the default value $\perp$ (line 12). In both cases, p_i exits the loop and starts the third phase.

  If the test of line line 11 is not satisfied, p_i checks (line 14) if it has new information from its failure detector (predicate $labels_i \neq \{x \mid (x, -) \in a\_sigma_i\}$), or has received a message PHASE2($r_i, sr, -, -$) such that $sr > sr_i$ (which means that this message refers to a sub-round of r_i more advanced than sr_i). If this test is satisfied, while remaining at the same round, p_i increase sr_i and broadcasts the message PHASE2($r_i, sr_i, labels_i, est1_i$) which refreshes the values of sr_i and $labels_i$ it had sent previously. Otherwise, p_i continues waiting for messages.

- The aim of the third phase of a round is to allow a process to decide when it discovers that a quorum of processes have the same non-$\perp$ value v in their estimates $est2_i$.

```
operation propose (vᵢ):
(1)    est1ᵢ ← vᵢ; rᵢ ← 0;
(2)    while true do
(3)      begin asynchronous round
(4)        rᵢ ← rᵢ + 1;
           % Phase 1 : assign a value to est1ᵢ with the help of AΩ %
(5)          wait until ((a_leaderᵢ) ∨ (PHASE1(rᵢ, v) received));
(6)          if (PHASE1(rᵢ, v) received) then est1ᵢ ← v end if;
(7)          broadcast PHASE1(rᵢ, est1ᵢ);
           % Phase 2 : assign a value v or ⊥ to est2ᵢ %
(8)          srᵢ ← 1; labelsᵢ ← {x | (x, −) ∈ a_sigmaᵢ}; broadcast PHASE2(rᵢ, srᵢ, labelsᵢ, est1ᵢ);
(9)          repeat
(10)           if (PHASE3(rᵢ, −, −, est2) received) then est2ᵢ ← est2; exit repeat loop end if;
(11)           if (∃ (x, y) ∈ a_sigmaᵢ ∧ ∃ sr ∈ ℕ

                      such that y msgs PHASE3(rᵢ, sr, labelsⱼ, −) received with x ∈ labelsⱼ for each message)
(12)             then if (all y previous messages contain the same estimate v) then est2ᵢ ← v else est2ᵢ ← ⊥ end if;
(13)                  exit repeat loop
(14)             else if (labelsᵢ ≠ {x | (x, −) ∈ a_sigmaᵢ}) ∨ (PHASE2(rᵢ, sr, −, −) received with sr > srᵢ)
(15)                  then srᵢ ← srᵢ + 1; labelsᵢ ← {x | (x, −) ∈ a_sigmaᵢ};
(16)                       broadcast PHASE2(rᵢ, srᵢ, labelsᵢ, est1ᵢ)
(17)                  end if
(18)           end if
(19)         end repeat;
           % Phase 3 : try to decide a value from the est2 values %
(20)         srᵢ ← 1; labelsᵢ ← {x | (x, −) ∈ a_sigmaᵢ}; broadcast PHASE3(rᵢ, srᵢ, labelsᵢ, est2ᵢ);
(21)         repeat
(22)           if (PHASE1(rᵢ + 1, −) received) then exit repeat loop end if;
(23)           if (∃ (x, y) ∈ a_sigmaᵢ ∧ ∃ sr ∈ ℕ

                      such that y msgs PHASE3(rᵢ, sr, labelsⱼ, −) received with x ∈ labelsⱼ for each message)
(24)             then let recᵢ = the set of estimates est2 contained in the y previous messages;
(25)                  case (recᵢ = {v})      then broadcast DECIDE(v); return(v)
(26)                       (recᵢ = {v, ⊥})   then est1ᵢ ← v
(27)                       (recᵢ = {⊥})      then skip
(28)                  end case;
(29)                  exit repeat loop
(30)             else if (labelsᵢ ≠ { x | (x, −) ∈ a_sigmaᵢ }) ∨ (PHASE3(rᵢ, sr, −, −) received with sr > srᵢ)
(31)                  then srᵢ ← srᵢ + 1; labelsᵢ ← {x | (x, −) ∈ a_sigmaᵢ};
(32)                       broadcast PHASE3(rᵢ, srᵢ, labelsᵢ, est2ᵢ)
(33)                  end if
(34)           end if
(35)         end repeat
(36)      end asynchronous round
(37)   end while.

(38)   when DECIDE(v) is received: broadcast DECIDE(v); return(v).
```

Figure 4: A Consensus algorithm for $\mathcal{AAS}[A\Sigma, A\Omega]$ (code for p_i)

The message exchange pattern used in this phase (where the notion of sub-round is used) is exactly the same as in the one used in the second phase where the value of $est2_i$ replaces the value of $est1_i$.

The only thing that changes with respect to the second phase is the processing done when the predicate of line 23 is satisfied (let us notice that this predicate is the same as the one of line 11 when the message tag PHASE2 is replaced by the tag PHASE3.

If p_i has received the same value v from all the processes that compose the last quorum defined from the predicate of line 23, it decides v (line 25). If it has received a value v and also $\perp$, it adopts v as its new estimate $est1_i$ (line 26). Finally, if it has received only $\perp$, it keeps its previous estimate $est1_i$ (line 27). As we will see in the proof, the property $P()$ established by the second phases, and the fact that the quorums defined by the predicates of lines 10 and 23, ensure that no two processes can decide differently.

6.2   Proof of the anonymous consensus algorithm

Lemma 1 *If no process decides, the correct processes execute an infinite number of rounds.*

Proof The proof is by contradiction. Assuming that no process decides, and correct processes block forever, let r be the smallest round at which a process blocks forever and let p_i be such a correct process. It can be blocked in the **wait until** statement in phase 1, or in the **repeat** loop in phase 2 or 3.

Phase 1. If p_i is the eventual leader it cannot block forever in phase 1. So, let p_ℓ, $\ell \neq i$, be the eventual leader. Process p_ℓ cannot be blocked forever at phase 1 of round r either because a_leader_ℓ becomes eventually true, or because p_ℓ receives a message PHASE1$(r, -)$. Whatever the case, p_ℓ broadcasts PHASE1$(r, -)$, and eventually p_i receives it. Hence, no correct process can block in phase 1 of round r.

Phases 2 and 3. As these cases are identical (from the point of view of process blocking), we only show that no correct process can loop forever in the **repeat** loop of phase 2. If process p_i loop forever in phase 2, we conclude from line 10 that no correct process has entered phase 3 (or the next round when considering blocking in phase 3), which means that all correct processes are looping forever in phase 2 of round r.

It follows from the liveness property of $A\Sigma$ that there is a finite time τ after which, for each correct process p_i, there is a pair $(x, y) \in a\_sigma_i$ such that $|S(x) \cap Correct| \geq y$. Due to (a) the definition of $S(x) = \{j \mid \exists \tau \in \mathbb{N} : (x, -) \in a\_sigma_j^\tau\}$, (b) the fact that, after a finite time, $S(x)$ contains at least y correct processes, and (c) the repeated broadcast of PHASE2$(r, -, -, -)$ messages with increasing sub-round numbers in **repeat** loop by the correct processes, it follows that, after a finite time, there is necessarily a sub-round sr during which p_i receives y messages PHASE2$(r, sr, labels_j, -)$ with $x \in labels_j$, and the predicate of line 11 is then satisfied. Consequently, no correct process p_i can block forever in phase 2 of round r, which concludes the proof of the lemma. $\hfill \square_{Lemma\ 1}$

Lemma 2 *Every correct process eventually decides.*

Proof Before deciding (line 25), a process broadcasts a message DECIDE$(-)$. Hence, if a process decides, all correct processes decide. Hence, let us assume, by contradiction, that no process decides.

Due to the definition of $A\Omega$, there is a time τ_0 after which there is exactly one correct process (say p_ℓ) whose boolean variable a_leader_ℓ remains forever true, while all other a_leader_i boolean variables remain forever false. Let τ_1 be a time after which all faulty processes have crashed. Finally, let $\tau \geq \max(\tau_0, \tau_1)$.

Due to Lemma 1, this means that there is a round r, entered by the correct processes after τ, from which p_ℓ is the only process such that $a\_leader_\ell = true$. Process p_ℓ is consequently the only process to broadcast PHASE1(r, v) with $v = est_\ell$, and each correct process receives this message (either directly from p_ℓ or after forwarding by another process). The important point here is that each correct process p_i is such that $est1_i = v$ at the end of the first phase of round r. Hence, they all broadcast PHASE2$(r, -, -, v)$ at line 7 (Observation O1).

As no process blocks forever in round r, there is a process (say p_i) that exits phase 2 of round r because the predicate of line 11 is satisfied. Hence, p_i exits the second phase of round r at line 13. Due to observation O1, we necessarily have $est2_i$ to $v = est_\ell \neq \perp$. All other processes exit phase 2 at line 13 or 10, and are consequently such that $est2_i = v = est_\ell$. Hence, they all broadcast PHASE3$(r, -, -, v)$.

A similar reasoning applies to phase 3. The first process p_j that stops looping in phase 3 is such that the predicate of line 23 is satisfied. As all the $est2$ it has received are equal to v, we have $rec_j = \{v\}$, and p_j decides v, a contradiction which complete the proof of the lemma. $\hfill \square_{Lemma\ 2}$

Lemma 3 *No two processes decide different values.*

Proof If no process decides or a single process decides at line 25, agreement is trivially satisfied. Hence, assuming that at least two processes decides at line 25, let r be the first round at which a process decides. Let p_i be a process that decides at that round, and v the value it decides. Let p_j be another process that decides at round $r' \geq r$. We consider two cases.

Case 1: p_j decides at round $r' = r$. As p_i decides v during round r, the predicate of line 23 is satisfied, i.e., $\exists (x_1, y_1) \in a\_sigma_i$ such that p_i has received y_1 messages PHASE3$(r, sr1, labels_k, v)$ with $x_1 \in labels_k$ (each message carrying its own value $labels_k$). Let $T_1 \subseteq S(x_1)$ be the set of processes that have sent these y_1 messages, hence $|T_1| = y_1$.

Similarly, as p_j decides (say v') during round r, there is pair $(x_2, y_2) \in a\_sigma_j$ such that p_j has received y_2 messages PHASE3$(r, sr2, labels'_k, v')$ with $x_1 \in labels'_k$ (each message carrying its own value $labels'_k$). Let $T_2 \subseteq S(x_2)$ be the set of processes that have sent these y_2 messages, hence $|T_2| = y_2$.

It follows from the safety property of $A\Sigma$ that $T_1 \cap T_2 \neq \emptyset$. Consequently, there is a process p_ℓ that sent messages PHASE3$(r, sr1, labels_\ell, v)$ to p_i, and PHASE3$(r, sr2, labels'_\ell, v')$ to p_j. As p_ℓ does not change the value of $est2_k$ while executing the third phase, we have $v = v' = est2_\ell$. Hence, $rec_i = rec_j = \{v\}$. Consequently, both p_i and p_j decide the same value v.

Case 2: p_j decides at round $r' > r$. Hence, p_j proceeds from r to $r+1$. In that case, during round r, we have $rec_j \neq \{v\}$ at line 24. But, the same reasoning as above applies, and we consequently have $rec_j = \{v, \bot\}$. It follows that any p_j that proceeds to $r+1$, is such that $est1_j = v$ when p_j starts round $r+1$. Said another way, v is the only estimate value present in round $r+1$, from which follows that no other value can be decided at line 24 in a round $r' > r$, which completes the proof of the consensus agreement property. $\square_{Lemma\ 3}$

Theorem 5 *The algorithm described in Figure 4 solves the consensus problem in $\mathcal{AAS}[A\Sigma, A\Omega]$.*

Proof The consensus validity property (a decided value is a proposed value) follows directly from the following observations O1 and O2. All the $est1_i$ variables are been initialized to proposed values (O1). A decided value is a non-$\bot$ value of an $est2_i$ local variable, which has been assigned the value of an $est1_j$ variable (O2).

The proof of consensus agreement follows from Lemma 2. The proof of consensus termination follows from Lemma 3.
                                                                                                         $\square_{Theorem\ 5}$

7   Which is the weakest failure detector class for anonymous consensus?

Reductions Given two failure detector classes $D1$ and $D2$, let us remember that $D1$ is *strictly weaker than* $D2$ in the system model $\mathcal{AS}[\emptyset]$ (denoted $D1 \prec D2$) if there is an algorithm that constructs a failure detector of the class $D1$ in $\mathcal{AS}[D2]$ (such an algorithm is called an *extraction* algorithm), while there is no algorithm that constructs a failure detector of the class $D2$ in $\mathcal{AS}[D1]$. Moreover, two failure detector classes $D1$ and $D2$ are equivalent (denoted $D1 \simeq D2$) in the system model $\mathcal{AS}[\emptyset]$ if (1) there is an algorithm that constructs a failure detector of the class $D1$ in $\mathcal{AS}[D2]$, and (2) there is an algorithm that constructs a failure detector of the class $D2$ in $\mathcal{AS}[D1]$. Finally, the notation $D1 \preceq D2$ is a shortcut for $(D1 \prec D2) \vee (D1 \simeq D2)$.

Notion of weakest failure detector class for a given problem Given a problem $\mathcal{P}$ and a failure detector class D, D is the *weakest failure detector class* for $\mathcal{P}$ in $\mathcal{XX}[\emptyset]$ (where $\mathcal{XX}$ stands for $\mathcal{AS}$ or $\mathcal{AAS}$) if (a) there is an algorithm that solves $\mathcal{P}$ in $\mathcal{XX}[D]$, and (b) for any failure detector class D' such that $\mathcal{P}$ can be solved in $\mathcal{XX}[D']$, we have $D \preceq D'$. It is shown in [21] that, in $\mathcal{AS}[\emptyset]$, any problem has a weakest failure detector class.

A new failure detector class Given two failure detectors classes $D1$ and $D2$, let us define a new failure detector class $D1 \oplus D2$ as follows. During an arbitrary but finite period of time, $D1 \oplus D2$ outputs $\bot$ at every process, and then behaves either as $D1$ or as $D2$ at all processes p_i. (A similar composition of failure detector classes in non-anonymous systems appears in [15].)

Let us observe that, if $D1$ and $D2$ cannot be compared, $D1$ (resp.,$D2$) is strictly stronger that $D1 \oplus D2$. This is because a failure detector of the class $D1 \oplus D2$ can trivially be built in $\mathcal{XX}[D1]$ (resp., $\mathcal{XX}[D2]$), while a failure detector of the class $D1$ (resp.,$D2$) cannot be built in $\mathcal{XX}[D1 \oplus D2]$.

The case of anonymous failure detectors for consensus The discussion in Section 5, where it is shown that AP and $A\Omega$ cannot be compared (Theorem 4), sets the question of the weakest failure detector class to solve consensus despite the three adversaries that are anonymity, crashes and asynchrony.

Let us consider the class of anonymous failure detectors $(A\Sigma, A\Omega) \oplus AP$. As $A\Omega$ and AP cannot be compared, it follows from the previous discussion that the class $(A\Sigma, A\Omega) \oplus AP$ is strictly weaker than both $(A\Sigma, A\Omega)$ and AP.

Moreover there is a simple (not genuine) algorithm that solves anonymous consensus in the system model $\mathcal{AAS}[(A\Sigma, A\Omega) \oplus AP]$. This algorithm is as follows. Each process p_i waits until the output of $\mathcal{AAS}[(A\Sigma, A\Omega) \oplus AP]$ is different from $\bot$. Then, according to the actual output of the failure detector (that is non-deterministic), it executes either $(A\Sigma, A\Omega)$-based algorithm presented in Section 6 or the AP-based algorithm described in [6]. (Let us observe that, as the algorithm presented in [6] is not genuine, the resulting algorithm is not genuine either.)

A conjecture We conjecture that $(A\Sigma, A\Omega) \oplus AP$ is the weakest failure detector class for solving anonymous consensus (with a non-genuine algorithm). This conjecture is motivated by the following observation.

In a non-anonymous system we have $((\Sigma, \Omega) \oplus P) \simeq (\Sigma, \Omega)$. This is because, as $\Sigma \prec P$ and $\Omega \prec P$, the behaviors of $((\Sigma, \Omega) \oplus P)$ when it behaves as P, cannot be obtained from (Σ, Ω). Said in another way, as $((\Sigma, \Omega) \oplus P) \simeq (\Sigma, \Omega)$, the weakest failure detector class for consensus in non-anonymous system is $((\Sigma, \Omega) \oplus P)$.

Let us finally observe that, as it has been shown that P is the weakest class of realistic failure detectors [13], there is maybe a (strong?) connection relating anonymous failure detectors and realistic failure detectors.

Remark It is interesting to compare the AP-based consensus algorithm described in [6], and the $(A\Sigma, A\Omega)$-based (genuine) consensus algorithm introduced in Section 6 (Figure 4).

The AP-based consensus algorithm [6] requires $\min(2f+2, 2t+1)$ communication (steps) rounds, where t is the maximum number of processes that may crash and f the actual number of crashes. It is shown in [6] that $(2t+1)$ is a lower bound. AP is a failure detector with a perpetual flavor in the sense that $aa\ell_i$ is always an upper bound on the number of correct processes.

Differently, while $A\Sigma$ has also a perpetual flavor, $A\Omega$ is defined by an eventual property. As a consequence, while finite, the number of rounds needed for consensus cannot be bounded. Moreover, the number of sub-rounds inside phase 2 and phase 3 of a round cannot be bounded either. They depend on the asynchrony pattern and the current output of $A\Sigma$.

This means that, not only AP and $(A\Sigma, A\Omega)$ cannot be compared in anonymous systems, but they are also far from being "equivalent" from an efficiency point of view. AP allows anonymous consensus to be solved in a bounded number of rounds, while the pair $(A\Sigma, A\Omega)$ does not. However in the best scenario, the second algorithm requires only one round (three communication steps) for the processes to decide (whatever n, t, and f).

8 Conclusion

This paper was on failure detectors in anonymous systems. It has presented three main contributions. The first is the class $A\Sigma$ of anonymous quorum failure detectors. The paper has shown that this class is the anonymous counterpart of the class Σ of quorum failure detectors (which means that there are equivalent in non-anonymous systems). The paper has also investigated the class AP of anonymous perfect detectors and presented a quiescent bounded construction that builds a failure detector of the class P in non-anonymous asynchronous systems enriched with AP.

The paper has then presented and proved a consensus algorithm for anonymous systems enriched with a failure detector of the class $A\Omega$ (the class of anonymous eventual leader failure detectors) and a failure detector of the class $A\Sigma$. As each process is not only anonymous, but additionally does not know the total number of processes, the consensus algorithm that is obtained is not trivial. It uses a new appropriate exchange pattern based on a finite number of sub-rounds inside every round.

Finally, the paper has discussed the hierarchy notion for anonymous failure detector classes. It has also discussed the notion of "weakest failure detector class" for anonymous consensus, and shown that, contrarily to their non-anonymous counterparts P and (Ω, Σ), the anonymous classes AP and $(A\Omega, A\Sigma)$ cannot be compared.

References

[1] Angluin D., Local and Global Properties in Networks of Processes. *Proc. 12th Symposium on Theory of Computing (STOC'80)*, ACM Press, pp. 82-93, 1980.

[2] Angluin D., Aspnes J., Diamadi Z., Fischer M.J. and Peralta R., Computation in Networks of Passively Mobile Finite-state Sensors. *Distributed Computing*, 18(4):235-253, 2006.

[3] Attiya H., Snir M. and Warmuth M.K., Computing on an Anonymous Ring. *Journal of the ACM*, 35(4):845-875, 1988.

[4] Attiya H. and Welch J., Distributed Computing, Fundamentals, Simulation and Advanced Topics (Second edition). *Wiley Series on Parallel and Distributed Computing*, 414 pages, 2004.

[5] Biely M., Robinson P. and Schmid U., Weak Synchrony Models and Failure Detectors for Message-passing (k)Set Agreement. *Proc. 13th Int'l Conference on Principles of Distributed Systems (OPODIS'09)*, Springer-Verlag LNCS #5923, pp. 285-299, 2009.

[6] Bonnet F. and Raynal M., The Price of Anonymity: Optimal Consensus despite Asynchrony, Crash and Anonymity. *Proc. 23th Int'l Symposium on Distributed Computing (DISC'09)*, Springer-Verlag LNCS #5805, pp. 341-355, 2009.

[7] Bonnet F. and Raynal M., Looking for the Weakest Failure Detector for k-Set Agreement in Message-passing Systems: Is Π_k the End of the Road? *Proc. 11th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'09)*, Springer-Verlag LNCS #5873, pp. 149-164, 2009.

[8] Bonnet F. and Raynal M., A Simple Proof of the Necessity of the Failure Detector Σ to Implement an Atomic Register in Asynchronous Message-passing Systems. *Information Processing Letters*, 110(4):153-157, 2010.

[9] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722, 1996.

[10] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[11] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.

[12] Chothia T. and Chatzikokolakis K., A Survey of Anonymous Peer-to-Peer File-Sharing. *Proc. Satellite workshop of the Int'l Conference on Embedded and Ubiquitous Systems (EUS'05)*, pp. 744-755, 2005.

[13] Delporte-Gallet C., Fauconnier H. and Guerraoui R., A Realistic Look at Failure Detectors, *Proc. Int'l Conference International on Dependable Systems and Networks (DSN'02)*, IEEE Computer Press, pp. 345-353, 2002.

[14] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Shared Memory *vs* Message Passing. To appear *Journal of the ACM. Tech Report* IC/2003/77, EPFL, Lausanne, December 2003.

[15] Delporte-Gallet C., Fauconnier H., Guerraoui R., Hadzilacos V., Kouznetsov P. and Toueg S., The Weakest Failure Detectors to Solve Certain Fundamental Problems in Distributed Computing. *Proc. 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, ACM Press, pp. 338-346, 2004.

[16] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Tielmann A., The Weakest Failure Detector for Message Passing Set-Agreement. *Proc. 22th Int'l Symposium on Distributed Computing (DISC'08)*, Springer-Verlag LNCS #5218, pp. 109-120, 2008.

[17] Delporte-Gallet C., Fauconnier H. and Tielmann A., Fault-Tolerant Consensus in Unknown and Anonymous Networks. *Proc. 29th IEEE Int'l Conference on Distributed Computing Systems (ICDCS'09)*, IEEE Computer Press, pp. 368-375, 2009.

[18] Durresi A., Paruchuri V., Durresi M. and Barolli L., A Hierarchical Anonymous Communication Protocol for Sensor Networks. *Proc. Int'l Conference on Embedded and Ubiquitous Systems (EUS'05)*, Springer Verlag LNCS #3824, pp. 1123-1132, 2005.

[19] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[20] Gifford D.K., Weighted Voting for Replicated Data. *Proc. 7th ACM Symposium on Operating System Principles (SOSP'79)*, ACM Press, pp. 150-172, 1979.

[21] Jayanti P. and Toueg S., Every Problem has a Weakest Failure Detector. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 75-84, 2008.

[22] Keidar I. and Shraer A., How to Choose a Timing Model. *IEEE Transactions on Parallel Distributed Systems*, 19(10):1367-1380, 2008.

[23] Lakshman T.V. and Wei V.K., Distributed Computing on Regular Networks with Anonymous Nodes. *IEEE Transactions on Computers*, 43(2):211-218, 1994.

[24] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.

[25] Mostefaoui A., Rajsbaum S., Raynal M. and Travers C., On the Computability Power and the Robustness of Set Agreement-oriented Failure Detector Classes. *Distributed Computing*, 21(3):201-222, 2008.

[26] Mostefaoui A., Rajsbaum S., Raynal M. and Travers C., The Combined Power of Conditions and Information on Failures to Solve Asynchronous Set Agreement. *SIAM Journal of Computing*, 38(4):1974-1601, 2008.

[27] Mostefaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach. *Proc. 13th Int'l Symposium on Distributed Computing (DISC'99)*, Springer-Verlag LNCS #1693, pp. 49-63, 1999.

[28] Mostefaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 2001.

[29] Yamashita M. and Kameda T., Computing on Anonymous Networks: Part I -Characterizing the Solvable Cases. *IEEE Transactions on Parallel Distributed Systems*, 7(1):69-89, 1996.

[30] Yamashita M. and Kameda T., Computing on Anonymous Networks: Part II -Decision and Membership Problems. *IEEE Transactions on Parallel Distributed Systems*, 7(1):90-96, 1996.