**Publications Internes de l'IRISA**
**ISSN : 2102-6327**
**PI 1938 – octobre 2009**

# Consensus in Anonymous Distributed Systems: Is There a Weakest Failure Detector?

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

**Abstract:** This paper is on failure detectors to solve the consensus problem in asynchronous systems made up of anonymous processes prone to crash and connected by asynchronous reliable channels. Anonymity means that any two processes cannot be distinguished one from the other: they have no name and execute the same code.

The paper has several contributions. It first introduces two new classes of failures detectors, denoted $AP$ and $A\Omega$, and presents an $AP$-based algorithm and an $A\Omega$-based algorithm that solve the consensus problem despite the three computational adversaries that are asynchrony, failures and anonymity. Then, the paper shows that, in crash-prone non-anonymous systems, (a) $AP$ and the class of *perfect* failure detectors (denoted $P$) are equivalent, and (b) $A\Omega$ and the class of *eventual leader* failure detectors (denoted $\Omega$) are also equivalent. Finally, the paper addresses the question of the weakest failure detector to solve consensus in an asynchronous crash-prone anonymous system. In non-anonymous systems, the class $P$ of perfect failure detectors is strictly stronger than the class $\Omega$ of eventual leader failure detectors that has been shown to be the weakest failure detector class for consensus in asynchronous crash-prone system. Quite surprisingly, the paper shows that their anonymous counterparts cannot be compared. Hence, while $\Omega$ is the weakest failure detector class to solve consensus in non-anonymous systems, it is possible that there are several (hence incomparable) classes of weakest failure detectors for this problem in anonymous systems.

**Key-words:** Anonymity, Asynchronous system, Consensus, Distributed computability, Eventual leader, Failure detector class, Message passing system, Perfect failure detector, Weakest failure detector

---

*Consensus dans les Systèmes Distribués : Existe-t-il un Plus Faible Détecteur de Fautes*

**Résumé :** *Cet article s'intéresse au problème du consensus dans les systèmes distribués anonymes. Il propose deux algorithmes résolvant ce problème en utilisant deux détecteurs de fautes différents. Ces deux derniers étant incomparables, les auteurs soulèvent alors le problème de l'existence d'un plus faible détecteur de fautes pour le consensus dans ce modèle.*

**Mots clés :** *Anonymat, Système asynchrone à passage de messages, Problème du consensus, Algorithmique distribuée, Leader ultime, Classes de détecteur de fautes, Plus faible détecteur de fautes*

---

* 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** In a somewhat restrictive way, the aim of a real-time system is to master *on time computing*, and the main aim of parallelism is to obtain *efficient* algorithms. Similarly we can say that the central issue of distributed computing consists in *mastering uncertainty*. This uncertainty has first appeared under the form of asynchrony, failure occurrences, and the multiplicity of loci of control (also referred as locality). More recently, new facets of uncertainty (such as dynamicity, scalability and 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 [10]. In the same vein, not all the sensor networks assume that each sensor has a proper identity [3, 11].

One of the very first works (to our knowledge) that addressed anonymous systems is from D. Angluin [2]. In that paper, considering message passing systems, she 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 [4], or networks with a regular structure [16]. Failure-free message passing anonymous systems have also been investigated in [21, 22] 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.

**Consensus in non-anonymous message-passing systems** Consensus is one of the most famous distributed computing problem. It is a coordination problem defined as follows: each process proposes a value, and each non-faulty process has to decide a value (termination), such that no two processes decide different values (agreement) and the decided value is a proposed value (validity). While it has a very simple statement and can be easily solved in (anonymous or not) crash-prone synchronous systems, the consensus problem has no solution in asynchronous message-passing non-anonymous failure-prone systems, as soon as (even only) one process can be faulty [13]. Trivially, the problem cannot be solved either if anonymity is added to asynchrony and failures.

The failure detector-based approach [8] is one of the most popular approaches to circumvent the consensus impossibility in non-anonymous 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. It has been show that the failure detector class denoted $\Omega$ is the weakest failure detector class that allows consensus to be solved [9]. This class includes the failure detectors that provides each process $p_i$ with a read-only local variable $leader_i$ that contains a process identity, and such that there is a finite (but unknown) time after which the non-crashed processes have forever the same leader, this leader being a non-faulty process.

**Previous result on consensus in anonymous message passing systems** As far as we know, [6] is the only paper we are aware of that has addressed the consensus problem in anonymous crash-prone message passing systems. In that paper, we have introduced the class (denoted $AP$) of anonymous perfect failure detectors and presented $AP$-based consensus algorithms. Such a failure detector provides each process $p_i$ with an integer (denoted $aa\ell_i$) whose current value is always an upper bound of the number of alive processes, and eventually converges to the number of non-faulty processes. As we will see in this paper, $AP$ is actually the counterpart of the class $P$ of perfect failure detectors.

Solving consensus in a synchronous system or an asynchronous system enriched with a perfect failure detector requires $t+1$ communication rounds in the worst case, which is optimal ($t$, $0 < t < n$, is the maximum number of processes that may crash in a run) [1, 7, 12, 14]. One of the main results of [6] is the proof that solving consensus with an anonymous perfect failure detector doubles the price, more precisely, there is no consensus algorithm that allows processes to always decide in less than $2t + 1$ rounds. This is a noteworthy feature of anonymity as it shows that, when one wants to deterministically solve consensus despite anonymity, an additional price of $t$ rounds has to be paid The lesson learned is that, from a time complexity point of view, the combination of asynchrony and anonymity doubles the price.

**Content of the paper and roadmap** The paper is made up of 6 sections. Section 2 presents the base anonymous asynchronous system model denoted $\mathcal{AAS}_{n,t}[\emptyset]$. Then, Section 3 enriches $\mathcal{AAS}_{n,t}[\emptyset]$ with a failure detector of the class $AP$ and shows that $AP$ and $P$ are equivalent in crash-prone non-anonymous systems. Section 4 introduces the class $A\Omega$ of the anonymous eventual leader failure detectors, and presents an $A\Omega$-based consensus algorithm. It also shows that the classes $A\Omega$ and $\Omega$ are equivalent in non-anonymous systems.

Section 5 compares the failure detector classes $AP$ and $A\Omega$, and shows that they are incomparable. This is surprising, as their non-anonymous counterparts $P$ and $\Omega$ are such that $P$ is strictly stronger than $\Omega$ (i.e., $P$ provides more information on

failure than $\Omega$: given $P$, one can always build $\Omega$, while the converse is impossible). This discussion sets the question of the existence of the weakest failure detector class for solving the consensus problem in an asynchronous crash-prone anonymous system. We conjecture that such a class does not exist. Finally, 6 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. Processes are anonymous in the sense that they have no name and execute the same algorithm. They are asynchronous in the sense that there is no assumption on their respective speeds.

**Failure model** Up to $t$ processes can crash in a run, $0 < t < n$. 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. The value of the system parameter $t$ is known by the processes. 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, the parameter $f$, $0 \leq f \leq t$, denotes the actual number of processes that crash.

**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 brodcast() communication primitive that allows the invoking process to send the same message to all the processes (including itself). The brodcast() 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 a message, a process cannot determine the sender of the message. Moreover, given any set of messages it has received, a process cannot determine if these messages are from the same sender or from different senders.

**Notation** The previous computation model is denoted $\mathcal{AAS}_{n,t}[\emptyset]$. $\mathcal{AAS}$ is an acronym for *A*nonymous *A*synchronous *S*ystem; $\emptyset$ means that there is no additional assumption.

$\mathcal{AS}_{n,t}[\emptyset]$ is used to denote the non-anonymous counterpart of $\mathcal{AAS}_{n,t}[\emptyset]$ [5, 17].

# 3 Anonymous system enriched with an anonymous perfect failure detector

## 3.1 The failure detector class $AP$

This failure detector class is a variant of a class introduced in [18, 19]. As we will see below, the class $AP$ is the equivalent of the class of perfect failure detectors $P$, when we consider non-anonymous systems.

**Definition** Let $f^\tau$ denote the number of processes that have crashed up to time $\tau$. 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 $a\ell$ive processes) that satisfies the following properties ($aa\ell_i^\tau$ denotes the value of $aa\ell_i$ at time $\tau$):

- Safety: $\forall \tau :\ aa\ell_i^\tau \geq n - f^\tau$.
- Liveness: $\exists \tau :\ \forall \tau' \geq \tau :\ aa\ell_i^{\tau'} = n - f$.

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.

**Notation** $\mathcal{AAS}_{n,t}[AP]$ denotes the system model $\mathcal{AAS}_{n,t}[\emptyset]$ enriched with a failure detector of the class $AP$. Similarly, $\mathcal{AS}_{n,t}[AP]$ denotes the system model $\mathcal{AS}_{n,t}[\emptyset]$ enriched with a failure detector of the class $AP$.

## 3.2 The class $AP$ and the class $P$ are equivalent in a non-anonymous system

**The class $P$ of perfect failure detectors** This class of failure detectors 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 (strong accuracy), and eventually contains the identities of all the faulty processes (completeness).

**Equivalence** As already claimed, the classes $AP$ and $P$ are equivalent when we consider a non-anonymous system. To show it, we consider a non-anonymous system, and present two transformations: a first one from $P$ to $AP$, and a second one from $AP$ to $P$.

*From $P$ to $AP$ in a non-anonymous system.* The transformation in that direction is trivial. The reader can easily check that taking the current value $n-|suspected_i|$ to define the current value of $aa\ell_i$, constructs a failure detector of the class $AP$.

*From $AP$ to $P$ in a non-anonymous system.* A transformation that builds a failure detector of the class $P$ in $\mathcal{AS}_{n,t}[AP]$ 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 two 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 $ans_i[j]$.

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

T1: repeat wait until (n − aℓ_i > k_i);
          k_i ← n − aℓ_i; brodcast INQUIRY(k_i)
    until (k_i = t) end repeat.

    when INQUIRY(k) is received from p_j: send ALIVE(k) to p_j.
    when ALIVE(k) is received from p_j: ans_i[j] ← max(ans_i[j], k).

    ---------------------------------------------------------

T2: repeat m ← k_i; % m is local to T2, while k_i is not %
          X ← {x such that ans_i[x] ≥ m};
          if (|X| = n − m) then suspected_i ← {1, ..., n} \ X end if
    until (|suspected_i| = t) end repeat.
```

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

The core of the transformation is the task $T2$ 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 $T2$ proceed asynchronously, $p_i$ broadcasts INQUIRY $(m)$ only after it knows that $m$ processes have crashed).

**Theorem 1** *The algorithm described in Figure 1 constructs 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 $\tau$, 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 $T2$ 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 $T2$. 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. $\square_{Theorem\ 1}$

### 3.3 Anonymous consensus in $\mathcal{AAS}_{n,t}[AP]$

**Description of the algorithm** A consensus algorithm for $\mathcal{AAS}_{n,t}[AP]$ is described in Figure 2. This algorithm is a simple adaptation of a flood set-based algorithm designed for synchronous system. A process $p_i$ invokes propose($v_i$) where $v_i$ is the value it proposes to the consensus. It terminates when it executes the return($est_i$) statement (line 10) where $est_i$ is the value it decides. The processes execute $(2t + 1)$ asynchronous rounds (line 2). In each round, each process $p_i$ broadcasts its current estimate (denoted $est_i$ and initialized to $v_i$) of the decision value and updates it (by taking the minimum on the values it has received and taken into account up to now, lines 5-6).

```
operation propose(v_i):
(1)  est_i ← v_i; r_i ← 1;
(2)  while (r_i ≤ 2t + 1) do
(3)     begin asynchronous round
(4)     brodcast EST(r_i, est_i);
(5)     wait until ( aℓ_i messages EST(r_i, −) have been received );
(6)     est_i ← min(est values received at the previous line);
(7)     r_i ← r_i + 1;
(8)     end asynchronous round
(9)  end while;
(10) return(est_i).
```

Figure 2: Anonymous consensus in $\mathcal{AAS}_{n,t}[AP]$

**Remark** If $n$ is known by the processes, the algorithm can be improved to reduce the number of rounds in the particular case where $t = n - 1$ (wait-free case). Instead of $2t + 1$ rounds, the processes can then execute only $2t$ rounds.

**Theorem 2** [6] *The algorithm described in Figure 2 solves the consensus problem in* $(2t + 1)$ *rounds in the* $\mathcal{AAS}_{n,t}[AP]$ *model.*

**Remark** On the one hand, the statement of the previous algorithm is pretty simple, and very close to the flood-set consensus algorithm for non-anonymous synchronous systems. On the other hand, its proof is rather technical and very different from its synchronous counterpart. This is due to the independence between communication that is asynchronous and crash notifications supplied by the failure detector $AP$ through integers ($aa\ell_i$). This is captured in [6] with the notion of misleading messages. A message $m$ is *misleading* if it allows its receiver to terminate a round $r$, while the sender of $m$ has crashed after or during the broadcast of $m$ during round $r$. Misleading messages create confusion, as they can direct a receiver to miss messages from correct processes (if $x$ of the $aa\ell_i$ messages that $p_i$ receives during a round $r$ are misleading, $p_i$ can miss $x$ round $r$ messages from alive processes).

## 4 Anonymous system enriched with an anonymous eventual leader failure detector

### 4.1 The class $A\Omega$ of anonymous eventual leader oracles

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

**Equivalence** $\Omega$ and $A\Omega$ are equivalent in $\mathcal{AS}_{n,t}[\emptyset]$ (for any value of $t$). The two directions of the equivalence are explained below. The proofs are straightforward and left to the reader.

*From $\Omega$ to $A\Omega$ in a non-anonymous system.* For any process $p_i$, the variable $leader_i^{A\Omega}$ of $A\Omega$ is computed by the test $leader_i^{\Omega} = i$ where $leader_i^{\Omega}$ is the output of $\Omega$.

*From $A\Omega$ to $\Omega$ in a non-anonymous system.* The reduction consists in two tasks executed by all processes: (1) Each process $p_i$ checks periodically its boolean $leader_i^{A\Omega}$ and if its value is true, it sends a message $Leader(i)$ (note that this reduction is done in the non-anonymous model, hence $p_i$ knows its identity). (2) When $p_i$ received a message $Leader(k)$, it updates its $leader_i^{\Omega}$ to $k$.

**Notation** The system model denoted $\mathcal{AAS}_{n,t}[t < n/2, A\Omega]$ is $\mathcal{AAS}_{n,t}[\emptyset]$ enriched with a failure detector of the class $A\Omega$ and where there is always a majority of correct processes.

## 4.2 Anonymous consensus in $\mathcal{AAS}_{n,t}[t < n/2, A\Omega]$

**Description of the algorithm** A consensus algorithm for $\mathcal{AAS}_{n,t}[t < n/2, A\Omega]$ is described in Figure 4. This algorithm is a variant of a leader-based consensus algorithm designed for non-anonymous systems [20]. The processes proceed by executing asynchronous rounds. Each round is made up of three communication steps (called phases and numbered 0, 1 and 2). The local variable $r_i$ is the current round number of $p_i$; $est1_i$ and $est2_i$ are local variables that contain $p_i$'s current estimate of the decision value.

- During phase 0 of round $r$, a process $p_i$ first waits until either it discovers it is leader, or it receives a message PHASE0$(r, v)$. If it is currently leader ($leader_i$ is then equal to *true*), $p_i$ broadcasts its current estimate value $est1_i$ (message PHASE0$(r, est1_i)$) in order the processes strive to decide it. Otherwise, $p_i$ adopts the estimate value $v$ it has received as the new value of its estimate $est1_i$. In that case, $p_i$ also broadcasts the message PHASE0$(r, est1_i)$. This is to circumvent the possible crash of the process that sends the message PHASE0$(r, v)$ it has received.

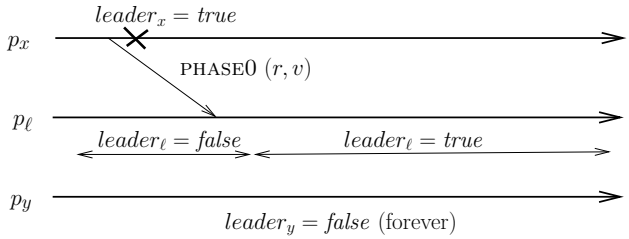


Figure 3: Why to forward PHASE0 () messages

As an example, let us consider Figure 3. Process $p_x$ sends PHASE0 $(r, v)$ to $p_\ell$ only, and then crashes. When, it receives this message, let us assume that $p_\ell$ proceeds to phase 1 of round $r$, without forwarding the message it has received. Moreover, let us assume that $p_\ell$ becomes the eventual leader after proceeding to phase 1 of round $r$ and all other boolean $leader_y$ take the value *false*. It is easy to see that $p_y$ remains blocked in round $r$ waiting forever for a message PHASE0 $(r, -)$.

- Then $p_i$ enters phase 1 of round $r$. During that phase, the processes exchange their current estimates $est1_i$. If $p_i$ received the same estimate value $v$ from a majority of processes it adopts $v$ as the current value of $est2_i$. Otherwise, it sets $est2_i$ to the default value $\perp$. As any two majorities do intersect, it is easy to see that for any two processes that terminate phase 1 of round $r$ we have $(est2_i \neq \perp) \wedge (est2_j \neq \perp) \Rightarrow (est2_i = est2_j = v)$ (quasi-agreement property).

- Finally, $p_i$ enters the phase 2 of round $r$. During that phase, the processes exchange the values of their estimates $est2_i$ (which are equal to the same non-$\perp$ value or $\perp$, hence we have $rec_i = \{v\}$, or $rec_i = \{v, \perp\}$, or $rec_i = \{\perp\}$. Moreover, due to the assumption $t < n/2$ and the quasi-agreement property on the estimates $est2_i$, it is impossible to have $rec_i = \{v\}$ and $rec_j = \{\perp\}$ for two processes $p_i$ and $p_j$ that terminate round $r$. The behavior of $p_i$ then depends on the content of $rec_i$.
  - If $rec_i = \{v\}$, $p_i$ decides $v$. As a process that decides stops executing the consensus algorithm, and processes wait for $n - t$ messages in phase 1 and 2 of a round, the stop of $p_i$ can entail other processes to wait forever during rounds $r' > r$. To prevent such a possible deadlock, a process $p_i$ is required to broadcast a message DECIDE $(v)$ before deciding.
  - If $rec_i = \{v, \perp\}$, $p_i$ adopts $v$ as its new estimate value $est1_i$ and proceeds to the next round.
  - If $rec_i = \{\perp\}$, $p_i$ proceeds to the next round without modifying $est1_i$.

```
operation propose (v_i):
  est1_i ← v_i; r_i ← 0;
  while true do
    begin asynchronous round
    r_i ← r_i + 1;
    % Phase 0 : select a value with the help of the oracle AΩ %
      wait until ((leader_i) ∨ (PHASE0(r_i, v) received));
      if (PHASE0(r_i, v) received) then est1_i ← v end if;
      broadcast PHASE0 (r_i, est1_i);
    % Phase 1 : from all to all %
      broadcast PHASE1 (r_i, est1_i);
      wait until (PHASE1 (r_i, −) received from n − t processes);
      if (the same estimate v has been received from > n/2 processes)
                              then est2_i ← v else est2_i ← ⊥ end if;
    % Phase 2 : try to decide a value from the est2 values %
      broadcast PHASE2 (r_i, est2_i);
      wait until (PHASE2 (r_i, −) received from n − t processes);
      let rec_i = {est2 | PHASE2 (r_i, est2) has been received};
      case (rec_i = {v})      then broadcast DECIDE(v); return(v)
           (rec_i = {v, ⊥})   then est1_i ← v
           (rec_i = {⊥})      then skip
      end case
    end asynchronous round
  end while.

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

Figure 4: A Consensus algorithm for $\mathcal{AAS}_{n,t}[t < n/2, A\Omega]$ (code for $p_i$)

**Theorem 3** *The algorithm described in Figure 4 solves the consensus problem in* $\mathcal{AAS}_{n,t}[t < n/2, \Omega]$.

**Proof** The proof of the validity property (a decided value is a proposed value) is trivial and left to the reader.

Proof of the agreement property. Let $r$ be the first round during which a process decides (say $v$). It follows from the quasi-agreement property that, if another process $p_j$ decides during the very same round, we have $rec_j = \{v\}$, and consequently $p_j$ decides the same value $v$.

Let us now consider a process $p_j$ that proceeds to round $r + 1$. As indicated previously, we have then $rec_j = \{v, \perp\}$ and $p_i$ sets $est1_j$ to the value $v$ before proceeding to the next round. It follows that all the processes $p_j$ that proceed to round $r + 1$ have $v$ as estimate value $est1_j$. Consequently, $v$ is the only value present in the system from the end of round $r$. It follows that no other value can be decided.

Proof of the termination property. Let us first observe that, if a process decides, due to the DECIDE() messages, every non-faulty process receives such a message and consequently decides. So, let us assume that no process ever decides.
Claim C. No non-faulty process blocks forever in a round.
Proof of claim C. The proof is by contradiction. let $r$ be the first round during which a non-faulty process (say $p_i$) blocks forever. Let us first consider phase 0 of round $r$. If $p_i$ is the eventual leader it cannot be blocked forever in phase 0 of $r$. Hence, assuming that $p_i$ is not the eventual leader, this means that $p_i$ never receives a message PHASE0 $(r_i, v)$. We consider several cases.

1. A non-faulty process $p_x$ is (momentarily or permanently) designated as leader, and that process broadcasts PHASE0 $(r_i, est1_x)$ that is received by $p_i$. In that case, process $p_i$ cannot block forever in the **wait** statement of phase 0.

2. A faulty process $p_x$ (momentarily designated as leader) sends PHASE0 $(r_i, est1_x)$ to $p_i$ (or to another process $p_y$ that forwards the message PHASE0 $(r_i, est1_x)$ to $p_i$). In that case, process $p_i$ cannot be blocked in the **wait** statement of phase 0.

3. No message PHASE0 $(r_i, est1_x)$ broadcast by a faulty process is received by $p_i$. In that case, the faulty processes either crash during round $r$ or are never designated as leader during round $r$. It then follows from the eventual leadership property of $A\Omega$, that there is a correct process that (momentarily or permanently) is designated as leader, and we are then in the scenario of Item 1.

If follows from the previous case analysis that no non-faulty process $p_i$ can block forever during phase 0 of round $r$. Due to the "majority of correct processes" assumption, it follows that $p_i$ can be blocked forever neither in the **wait** statement of phase 1 nor in the **wait** statement of phase 2. End of proof of claim C.

As (1) no process decides (assumption), (2) the non-faulty processes are never blocked in a round (Claim C), and (3), from their very definition, the faulty processes eventually crash, it follows that there is a round from which there are only non-faulty processes and one of them (say $p_\ell$) is forever designated as the single leader. Let $r$ be such a round.

During $r$, the leader $p_\ell$ is the only process such that $leader_\ell$ is equal to *true*, and it consequently the first process that broadcasts PHASE0 $(r, v)$. Moreover, as any other process $p_j$ is such that $\neg leader_j$ from round $r$, such a process can only receive (and then forwards) PHASE0 $(r, v)$. It follows that at the end of phase 0, the local estimates $est1_j$ are all equal to $v = est1_\ell$. Hence, the local estimates $est2_j$ are all equal to $v$ at the end of phase 1. Then, during phase 2, every non-faulty process is such that $rec_j = \{v\}$, and consequently decides, which contradicts the initial assumption and completes the proof of the termination property. $\square_{Theorem\ 3}$

**Number of communication steps** The previous algorithm assumes a majority of correct processes. Any of its execution requires a finite number of rounds (each round being made up of three communication steps). This number, which is independent of $t$, cannot be bounded (as it depends on the quality of the failure detector).

Let a failure detector of the class $A\Omega$ be *perfect* if a single non-faulty leader is elected from the very beginning. It is easy to see that, if the failure detector is perfect, a process decides at the end of the first round.

This is in contrast with the algorithm described in Figure 2 designed for the $\mathcal{AAS}_{n,t}[AP]$ model, that requires $(2t + 1)$ rounds. This algorithm works for any value of $t$ $(0 < t < n)$. Its constant number of rounds is independent of the behavior of the failure detector[1].

**An open problem** The two previous algorithms set the following question: Is it possible to design a consensus algorithm in $\mathcal{AAS}_{n,t}[AP, t < n/2, A\Omega]$ that combines the advantages of both previous algorithms, namely, (a) always at most $2t + 1$ rounds, (b) a single round in the runs where $A\Omega$ is perfect?

# 5 Is there a weakest failure detector for consensus in anonymous systems?

***AP* and $A\Omega$ are incomparable** It is known that the class $P$ is strictly stronger than the class $\Omega$. This means that, given any failure detector of the class $P$, it is possible to build a failure detector of the class $\Omega$ (e.g., elect as current leader the process with the smallest identity among the processes that are not suspected), while the converse is not true (namely, $\Omega$ eventually elects a correct process, but gives no information on which processes have crashed).

A little bit surprisingly, it appears that, in anonymous systems, $AP$ and $A\Omega$ are incomparable. None of them is stronger than the other.

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

**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 $aa\ell_i$.

[1] It is possible to design an early-deciding consensus algorithm for the model $\mathcal{AAS}_{n,t}[AP]$, in which a process decides in at most $\min(2t+1, 2f+2)$ rounds (let us remember that $f$ is the number of actual crashes) [6]. It is easy to see that, with this early-deciding algorithm, the number of rounds is no longer constant (as it depends on the run) but remains upper bounded by $2t + 1$.

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 A$\Omega$ 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 $\mathcal{AAS}_{n,t}[A\Omega]$. By construction $T$ does not rely on the process identities. Moreover, in a non-anonymous system, it is possible to transform $\Omega$ into $A\Omega$ (algorithm $T'$), it is possible to transform $AP$ into $P$ (algorithm $T''$). It is then possible to build a failure detector of the class $P$ in $\mathcal{AS}_{n,t}[\Omega]$ as follows. (All algorithms are executed in $\mathcal{AS}_{n,t}[\Omega]$.)

- 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 $\mathcal{AS}_{n,t}[\Omega]$. It follows that $T$ cannot exist. $\square_{Theorem\ 4}$

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

Given a problem $\mathcal{P}$ and a failure detector class $D$, $D$ is the *weakest failure detector class* for $\mathcal{P}$ in $\mathcal{AS}_{n,t}[\emptyset]$ if (a) there is an algorithm that solves $\mathcal{P}$ in $\mathcal{AS}_{n,t}[D]$, and (b) for any failure detector class $D'$, such that $\mathcal{P}$ can be solved in $\mathcal{AS}_{n,t}[D']$, we have $D \preceq D'$. It is shown in [15] that any problem has a weakest failure detector in $\mathcal{AS}_{n,t}[\emptyset]$.

Theorem 4 sets the question of the weakest failure detector class to solve consensus despite the three adversaries that are anonymity, crashes and asynchrony. We conjecture that there are several weakest failure detector classes for consensus in $\mathcal{AAS}_{n,t}[\emptyset]$. If follows that these classes are incomparable.

The traditional definition (of weakest failure detector class for a problem $\mathcal{P}$, stated above) does not allow several weakest failure detectors classes for $\mathcal{P}$ to exist. So, a new definition that allows several weakest failure detector class to co-exist is needed. Hence, we redefine the notion of *weakest failure detector class* for a problem $\mathcal{P}$ as follows. $D$ is a weakest failure detector class for $\mathcal{P}$ in a system model $\mathcal{XX}_{n,t}[\emptyset]$ (where $\mathcal{XX}$ stands for either $\mathcal{AS}$ or $\mathcal{AAS}$) if:

- There is an algorithm that solves $\mathcal{P}$ in $\mathcal{XX}_{n,t}[D]$, and
- $\forall\ D'$, $D' \prec D$, $\mathcal{P}$ cannot be solved in $\mathcal{XX}_{n,t}[D']$.

Let us notice that, as any problem has a weakest failure detector class when we consider the system model $\mathcal{AS}_{n,t}[\emptyset]$ [15], it follows that both definitions are equivalent in this system model. When we consider the system model $\mathcal{AAS}_{n,t}[\emptyset]$, the proposed definition is more general (in the sense that it extends the traditional definition while allowing several classes to be the weakest ones for a given problem we want to solve in $\mathcal{AAS}_{n,t}[\emptyset]$).

The example described on Figure 5 presents a situation where two weakest failure detectors exist for a problem $P$. Failure detectors $D_k$, $WFD_1$, and $WFD_2$ allow $P$ to be solved whereas failure detectors $D'_k$ do not. An arrow from $A$ to $B$ means that failure detector class $B$ is weaker than failure detector class $A$. In that case and according to the definition, each of $WFD_1$ and $WFD_2$ is a weakest failure detector class for the problem $P$ since both allow $P$ to be solved, and no strictly weaker failure detector class does so.

# 6 Conclusion

This paper was focused on the quest for discovering the weakest failure detector to solve the consensus problem in asynchronous anonymous crash-prone message passing systems. To that end, the paper has presented two classes of anonymous failure detectors. It bas been shown that these classes are the anonymous counterparts of the class of perfect failure detectors and eventual leader failure detectors encountered in non-anonymous systems. Consensus algorithms based on these failure detector classes have been presented and proved correct.

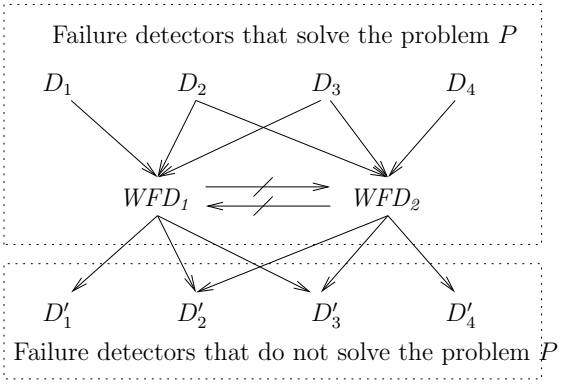


Figure 5: Hierarchy of failure detector classes

The paper has shown that anonymity collapses the hierarchy of failure detectors for non-anonymous systems. More precisely, while a perfect failure detector is strictly stronger than an eventual leader failure detector, the paper has shown that their anonymous counterparts are incomparable. Hence, the conjecture that maybe there is no weakest failure detector class for consensus in asynchronous crash-prone anonymous systems.

# References

[1] Aguilera M.K. and Toueg S., A Simple Bivalency Proof that $t$-Resilient Consensus Requires $t+1$ Rounds. *Information Processing Letters*, 71:155-178, 1999.

[2] 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.

[3] 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.

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

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

[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., Early Consensus in Message-passing Systems Enriched with a Perfect Failure Detector and its Application in the Theta Model. *Tech Report #1937*, IRISA, Université de Rennes (F), 13 pages, 2009.

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

[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] 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.

[11] 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.

[12] Fischer M.J., Lynch N.A., A Lower Bound on the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183-186, 1982.

[13] 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.

[14] Hélary J.-M., Hurfin M., Mostefaoui A., Raynal M. and Tronel F. Computing Global Functions in Asynchronous Distributed Systems with Perfect Failure Detectors. *IEEE Trans. on Parallel and Distributed Systems*, 11(9):897-909, 2000.

[15] 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.

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.