# Relations Linking Failure Detectors Associated with k-Set Agreement in Message-Passing Systems

Achour Mostefaoui, Michel Raynal, Julien Stainer

**Publications Internes de l'IRISA**
**ISSN : 2102-6327**
**PI 1973 – Avril 2011**

# Relations Linking Failure Detectors Associated with $k$-Set Agreement in Message-Passing Systems

Achour Mostéfaoui[*] Michel Raynal[**] Julien Stainer[***]

**Abstract:** The $k$-set agreement problem is a coordination problem where each process is assumed to propose a value and each process that does not crash has to decide a value such that each decided value is a proposed value and at most $k$ different values are decided. While it can always be solved in synchronous systems, $k$-set agreement has no solution in asynchronous send/receive message-passing systems where up to $t \geq k$ processes may crash.

A failure detector is a distributed oracle that provides processes with additional information related to failed processes and can consequently be used to enrich the computability power of asynchronous send/receive message-passing systems. Several failure detectors have been proposed to circumvent the impossibility of $k$-set agreement in pure asynchronous send/receive message-passing systems. Considering three of them (namely, the generalized quorum failure detector $\Sigma_k$, the generalized loneliness failure detector $\mathcal{L}_k$ and the generalized eventual leader failure detector $\Omega_k$) the paper investigates their computability power and the relations that link them. It has three mains contributions: (a) it shows that the failure detector $\Omega_k$ and the eventual version of $\mathcal{L}_k$ have the same computational power; (b) it shows that $\mathcal{L}_k$ is realistic if and only if $k \geq n/2$; and (c) it gives an exact characterization of the difference between $\mathcal{L}_k$ (that is too strong for $k$-set agreement) and $\Sigma_k$ (that is too weak for $k$-set agreement).

**Key-words:** Asynchronous message-passing system, Distributed computability, Equivalence, Eventual leader, Failure detector, Fault-tolerance, Impossibility, Quorum, Realistic failure detector, Reduction, $k$-Set agreement, Theory.

---

## *Comprendre les détecteurs de fautes associés au problème de $k$-accord*

**Résumé :** *Ce rapport reprend les différents détecteurs de fautes introduits pour renforcer les systèmes asynchrones et rendre le problème de $k$-accord décidable. En effet, plusieurs détecteurs de fautes ont été introduits mais certains sont trop forts et d'autres trop faibles. Ce rapport a pour but de déterminer l'espace qui sépare les premiers des seconds afin d'essayer de se rapprocher du détecteur minimal.*

**Mots clés :** *détecteur de fautes, $k$-accord, leader, tolérance aux fautes, oracle, réduction algorithmique, système asynchrone.*

---

[*] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes, achour@irisa.fr
[**] IUF and Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes, raynal@irisa.fr
[***] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes, jstainer@irisa.fr

## 1 Introduction

**On failure detectors** Let us observe that in asynchronous systems where the only means for processes to communicate is using send/receive message-passing, no process is able to know if another process has crashed or is only very slow. The concept of a failure detector originates from this simple observation. A failure detector is a device (distributed oracle) whose aim is to enrich a distributed system by providing alive processes with information on failed processes [6]. Several classes of failure detectors can be defined according to the type of information on failures they provide to processes (see [16] for an introduction to failure detectors).

**The $k$-set agreement problem** This problem, that has been introduced by S. Chaudhuri [8], is a coordination problem that generalizes the consensus problem. It can be defined as follows [8, 15]. Each process proposes a value and every non-faulty process has to decide a value (termination) in such a way that any decided value is a proposed value (validity) and no more than $k$ different values are decided (agreement). The problem parameter $k$ defines the coordination degree: $k = 1$ corresponds to its most constrained instance (consensus) while $k = n - 1$ corresponds to its weakest non-trivial instance (called set agreement).

Let $t$ be the model parameter that denotes the upper bound on the number of processes that may crash in a run, $1 \leq t < n$. If $t < k$, $k$-set agreement can be trivially solved in both synchronous and asynchronous systems: $k$ predetermined processes broadcast the values they propose and a process decides the first proposed value it receives. Hence, the interesting setting is when $k \geq t$, i.e., when the number of values that can be decided is smaller or equal to the maximal number of processes that may crash in a run.

Algorithms that solve the $k$-set agreement problem in synchronous message-passing systems when $k \leq t$ are presented in [1, 13, 18]. These algorithms are based on a sequence of synchronous communication rounds. It is shown in the three books previously referenced that $\lfloor \frac{t}{k} \rfloor + 1$ rounds are necessary and sufficient to solve $k$-set agreement. (This lower bound is still valid in more severe failure models such as general omission failures [18].)

For crash-prone asynchronous systems (be the communication medium a read/write shared memory or a send/receive message-passing network) the situation is different, namely, the $k$-set agreement problem has no solution when $t \geq k$ [5, 12, 20].

**The cases $k = 1$ and $k = n - 1$ in message-passing systems** When $k = 1$, as already indicated $k$-set agreement boils down to consensus, and it is known that the failure detector denoted $\Omega$ is the weakest to solve consensus in asynchronous message-passing systems where $t < n/2$ [7]. $\Omega$ ensures that there is an unknown but finite time after which all the processes have the same non-faulty leader (before that time, there is an anarchy period during which each process can have an arbitrarily changing leader). This lower bound result is generalized in [10] where the failure detector $\Sigma$ is introduced and it is shown that the pair $\langle \Sigma, \Omega \rangle$ is the weakest failure detector to solve consensus in message-passing systems when $t < n$. This means that $\Sigma$ is the minimal additional power (as far as information on failures is concerned) required to overcome the barrier $t < n/2$ and attain $t \leq n - 1$. Actually, the power provided by $\Sigma$ is the minimal one required to implement a shared register in a message-passing system [4, 10]. $\Sigma$ provides each process with a quorum (set of process identities) such that the values of any two quorums (each taken at any time) intersect, and there is a finite time after which any quorum includes only correct processes. Fundamentally, $\Sigma$ prevents partitioning. A failure detector $\langle \Sigma, \Omega \rangle$ outputs a pair of values, one for $\Sigma$ and one for $\Omega$.

The *Loneliness* failure detector (denoted $\mathcal{L}$) has been proposed in [11] where it is proved that it is the weakest failure detector for solving $(n-1)$-set agreement in the asynchronous message-passing model. Such a failure detector provides each process $p$ with a boolean (that $p$ can only read) such that the boolean of at least one process remains always false and, if all but one process crash, the boolean of the remaining process becomes and remains true forever. It is important to notice that, albeit surprisingly, the weakest failure detector for $(n - 1)$-set agreement is not the same in the read/write shared memory model (where it is $\overline{\Omega}_{n-1}$) and the send/receive message-passing model (where it is $\mathcal{L}$).

Unfortunately, the weakest failure detector for $k$-set agreement when $1 < k < n - 1$ in message-passing asynchronous crash-prone systems is not yet known. The interested reader will find an introductory survey on failure detectors suited to $k$-set agreement in [19].

**Content of the paper** Among the failure detectors for $k$-set agreement, $1 \leq k \leq n-1$, that have been proposed in the past few years, this paper investigates the relations on three of them, namely the ones denoted $\mathcal{L}_k$, $\Omega_k$ and $\Sigma_k$ (their precise definitions are given in Section 3). The failure detector $\Omega_k$, introduced in [14], is a generalization of $\Omega$ ($\Omega_1$ is $\Omega$).

The failure detector $\mathcal{L}_k$, introduced in [2], is a generalization of $\mathcal{L}$. It allows the $k$-set agreement problem to be solved in message-passing despite asynchrony and any number of process crashes. Unfortunately, $\mathcal{L}_k$ has been proved to be (a little bit) too strong to solve $k$-set agreement. Hence, the question "How much stronger is it?"

The failure detector $\Sigma_k$, introduced in [3], is a generalization of $\Sigma$. It is shown in [3] that $\Sigma_k$ is necessary (but unfortunately not sufficient) to solve $k$-set agreement. Hence, the question "How much weaker is it?" It is also shown in [3] that $\Sigma_{n-1}$ and $\mathcal{L}_{n-1}$ are equivalent (they provide processes with the same computational power).

Answering the two previous questions seems difficult as it would provide us with key elements to obtain the weakest failure detector for asynchronous message-passing $k$-set agreement. Hence, we consider a more modest question in this paper whose answer will help us better understand and pave the way to the discovery of the weakest failure detector for message-passing $k$-set agreement. The question is "Which is the property that has to be added to $\Sigma_k$ in order to obtain exactly $\mathcal{L}_k$?" To be more explicit let $X_k$ be this property. Answering this question means solving the equation $\langle \Sigma_k, X_k \rangle \simeq \mathcal{L}_k$ where $X_k$ is the unknown and $\simeq$ means "have the same computational power". This paper has three contributions.

- It first focuses on the implementability of $\mathcal{L}_k$ in a synchronous system. Let us remember that a failure detector is *realistic* if it can be implemented in a synchronous system [9]. The paper shows that $k \geq n/2$ is a necessary and sufficient requirement for $\mathcal{L}_k$ to be realistic.
- It then answers the previous question by giving $X_k$.
- It finally shows that the "eventual" version of $\mathcal{L}_k$, which we denote $\diamond\mathcal{L}_k$, is nothing else than $\Omega_k$. ("Eventual" means that the property of $\mathcal{L}_k$ are required to be satisfied only after some finite time).

**Roadmap** The paper is made up of 8 sections. Section 2 presents the base computation model and the $k$-set agreement problem. Section 3 introduces the failure detectors in which we are interested, i.e., $\Sigma_k$, $\mathcal{L}_k$ and $\Omega_k$. Section 4 presents an $\mathcal{L}_k$-based $k$-set agreement algorithm due to [2]. The next three sections are the contributions of the paper: Section 5 is on the realism of $\mathcal{L}_k$, Section 6 solves the equation $\langle \Sigma_k, X_k \rangle \simeq \mathcal{L}_k$, while Section 7 shows that $\diamond\mathcal{L}_k$ and $\Omega_k$ are equivalent. Finally, Section 8 concludes the paper.

## 2 Base computation model and $k$-set agreement

### 2.1 Computation model

**Process model** The system consists of a set of $n$ sequential processes denoted $p_1$, ..., $p_n$. $\Pi = \{1, \dots, n\}$ is the set of process identities. Each process executes a sequence of (internal or communication) atomic steps. A process executes its code until it possibly crashes (if it ever crashes). After it has crashed, a process executes no more step. A process that crashes in a run is said *faulty* in that run, otherwise it is *correct*. Given a run, $\mathcal{C}$ and $\mathcal{F}$ denote the set of processes that are correct and the set of processes that are faulty, respectively. Up to $t = n-1$ processes may crash in a run, hence, $1 \leq |\mathcal{C}| \leq n$.

**Communication model** The processes communicate by executing atomic communication steps which are the sending or the reception of a message. Every pair of processes is connected by a bidirectional channel. The channels are failure-free (no creation, alteration, duplication or loss of messages) and asynchronous (albeit the time taken by a message to travel from its sender to its receiver is finite, there is no bound on transfer delays). The notation "broadcast MSG_TYPE$(m)$" is used as a (non-atomic) shortcut for "**for each** $j \in \Pi$ **do** send MSG_TYPE$(m)$ to $p_j$ **end for**".

**Underlying time model** The underlying *time model* is the set $\mathbb{N}$ of natural integers. As we are in an asynchronous system, this time notion is not accessible to processes (hence, the model is sometimes called time-free model). It can only be used from an external observer point of view to state or prove properties. Time instants are denoted $\tau$, $\tau'$, etc.

**Notation** The previous asynchronous crash-prone message-passing system model is denoted $\mathcal{AMP}[\emptyset]$. $\mathcal{AMP}$ stands for "$\mathcal{A}$synchronous $\mathcal{M}$essage-$\mathcal{P}$assing"; $\emptyset$ means the "base" system (not enriched with a failure detector).

### 2.2 The $k$-set agreement problem

As already indicated, the $k$-set agreement problem has been introduced by Soma Chaudhuri [8]. It generalizes the consensus problem (that corresponds to $k = 1$). It is defined as follows. Each process proposes a value and has to decide a value in such a way that the following properties are satisfied:

- Termination. Every correct process decides a value.
- Validity. A decided value is a proposed value.
- Agreement. At most $k$ different values are decided.

## 3 The failure detectors $\Omega_k$, $\Sigma_k$ and $\mathcal{L}_k$

This section presents the three failure detectors we are interested in. (People interested in the underlying assumptions and algorithms to implement failure detectors in asynchronous systems can consult Chapter 7 of [17].)

The system model $\mathcal{AMP}[\emptyset]$ enriched with any of these failure detectors $A$ is denoted $\mathcal{AMP}[A]$. A failure detector provides each alive process $p_i$ with a read-only local variable, say $xxx_i$. The value of $xxx_i$ at time $\tau$ is denoted $xxx_i^\tau$.

### 3.1 The *eventual leadership* failure detector $\Omega_k$

The failure detector $\Omega_k$ has been introduced in [14]. Its local output at $p_i$ is a set denoted $leaders_i$. $\Omega_k$ is defined by the two following properties.

- Validity. $\forall i, \forall \tau$: $(leaders_i^\tau \subset \Pi) \ \wedge \ (|leaders_i^\tau| = k)$.
- Eventual leadership. $\exists\, LD, \exists\, \tau$: $(LD \cap \mathcal{C} \neq \emptyset) \ \wedge \ (\forall \tau' \geq \tau,\ \forall i \in \mathcal{C}:\ leaders_i^{\tau'} = LD)$.

Validity means that the values of $leaders_i$ are $k$ process identities. Eventual leadership states that, after some unknown but finite time, all correct processes have the same set of leaders and at least one of these leaders is a correct process. Before all processes are provided with the same set of leaders, there is possibly an unknown but finite anarchy period during which the sets $leaders_i$ have arbitrary values.

The failure detector from which $\Omega_k$ originates is $\Omega$ (which is the same as $\Omega_1$). As already noticed, $\Omega$ has been introduced in [7] where it is shown to be the weakest failure detector to solve consensus in message-passing systems with a majority of correct processes. It has later been shown in [10] that $\langle \Sigma, \Omega \rangle$ is the weakest failure detector to solve consensus for $t = n - 1$.

### 3.2 The *quorum* failure detector $\Sigma_k$

The failure detector $\Sigma_k$ has been introduced in [3]. It is a generalization of the failure detector $\Sigma$ (called quorum failure detector) introduced in [10] where it is shown to be the weakest failure detector to implement a register in $\mathcal{AMP}[\emptyset]$ ($\Sigma_1$ is $\Sigma$).

The local output at $p_i$ of $\Sigma_k$ is a set $qr_i$ (called quorum) that satisfies the following properties.

- Liveness. $\exists\, \tau$: $\forall\, i \in \mathcal{C}, \forall\, \tau' \geq \tau$: $qr_i^{\tau'} \subseteq \mathcal{C}$.
- Intersection. Let $\mathcal{I} = \{id_x\}_{1 \leq x \leq k+1} \subset \Pi$ be a multiset of $k + 1$ process identities. Let $\mathcal{T} = \{\tau_x\}_{1 \leq x \leq k+1}$ be a multiset of $k + 1$ time instants. $\forall\, \mathcal{I}, \forall\, \mathcal{T}$: $\exists\, i, j \in [1..k + 1] : (i \neq j) \wedge (qr_{id_i}^{\tau_i} \cap qr_{id_j}^{\tau_j} \neq \emptyset)$.

The liveness property states that the quorum of a correct process eventually includes only correct processes. The intersection property states that any set of $k+1$ quorums, whose values are taken at any times, contains two intersecting quorums. This means that the intersection property prevents processes to partition in more than $k$ subsets. It is shown in [3] that $\Sigma_k$ is necessary when one wants to solve $k$-set agreement in $\mathcal{AMP}[\emptyset]$.

## 3.3 The *loneliness* failure detector $\mathcal{L}_k$

The $\mathcal{L}_k$ family introduced in [2] is a generalization of the failure detector $\mathcal{L}$ proposed in [11] where it is shown that $\mathcal{L}$ is the weakest failure detector for $(n-1)$-set agreement in asynchronous message-passing systems. $\mathcal{L}_{n-1}$ is $\mathcal{L}$.

The local output at $p_i$ of $\mathcal{L}_k$ is a boolean variable $alone_i$ that satisfies the following two properties (after a process $p_i$ has crashed, we have $alone_i = false$ by definition).

- Stability. $\exists\, K \subset \Pi : (|K| = n-k) \;\wedge\; (\forall\, i \in K,\, \forall\, \tau :\; alone_i^{\tau} = false)$.
- Loneliness. $(|\mathcal{C}| \leq n-k) \;\Rightarrow\; (\exists\, i \in \mathcal{C}, \exists \tau :\; \forall\, \tau' \geq \tau :\; alone_i^{\tau'} = true)$.

Stability states that the boolean variables of at most $k$ processes can take the value $true$, while the loneliness property states that, if at least $k$ processes crash, then there is a finite time after which there is a correct process $p_i$ whose boolean variable $alone_i$ remains forever equal to $true$. An algorithm solving the $k$-set agreement problem in $\mathcal{AMP}[\mathcal{L}_k]$ due to [2] is described in the next section. It is also shown in [2] that, for $1 < k < n-1$, $\mathcal{L}_k$ is not the weakest failure detector for $k$-set agreement in $\mathcal{AMP}[\emptyset]$.

# 4 A $k$-set agreement algorithm for $\mathcal{AMP}[\mathcal{L}_k]$

Algorithm 1 is an $\mathcal{L}_k$-based $k$-set agreement algorithm due to Biely, Robinson and Schmid [2]. This algorithm is based on a sequence of asynchronous rounds ($r_i$ denotes the current round number of process $p_i$). The local variable $est_i$ is $p_i$'s current estimate of its decision value; it is initialized to the value $v_i$ proposed by $p_i$. The execution of the statement return$(v)$ returns the value $v$ and terminates the invocation of set_agreement$_k()$.

```
when operation set_agreement_k($v_i$) is invoked by $p_i$:
(01) $est_i \leftarrow v_i$; $r_i \leftarrow 1$;
(02) repeat forever
(03)    for each $j \neq i$ do send EST($r_i, est_i$) to $p_j$ end for;
(04)    wait until $\big( (n-k)$ messages EST($r_i, -$) have been received $\big)$;
(05)    $est_i \leftarrow \min(est_i,$ the $est_j$ received at the previous line);
(06)    if ($r_i = k+1$)
(07)       then for each $j \neq i$ do send DEC($est_i$) to $p_j$ end for;
(08)            return($est_i$)
(09)       else $r_i \leftarrow r_i + 1$
(10)    end if
(11) end repeat.

when $\big(alone_i \vee$ DEC($v$) is received$\big)$:
(12) if (DEC($v$) received) then $est_i \leftarrow v$ end if;
(13) for each $j \neq i$ do send DEC($est_i$) to $p_j$ end for;
(14) return($est_i$).
```

Algorithm 1: $k$-Set agreement in $\mathcal{AMP}[\mathcal{L}_k]$ algorithm (code for $p_i$) [2]

**Algorithm description** In each round, each non-crashed process $p_i$ first broadcasts a message EST$(r_i, est_i)$ (line 03) to inform the other processes of its current estimate $est_i$ and waits until it has received $(n-k)$ estimate messages associated with its current round $r_i$ (line 04). When it has received these estimates, it computes the smallest of them including its own estimate (line 05). Then if $r_i < k+1$, $p_i$ proceeds to the next asynchronous round (line 09). If $r_i = k+1$, it broadcasts DEC$(est_i)$ to inform the other processes on the value it is about to decide (line 07) and then decides it (line 08).

When considering lines 01-11 only, let us observe that $p_i$ can block forever at line 04 if more than $k$ processes have crashed. Such a permanent blocking is prevented by the combined use of the failure detector $\mathcal{L}_k$ and DEC() messages (which ensures that, as soon as a process decides, all correct processes eventually decide). Observing that the boolean $alone_i$ of at least one correct process $p_i$ becomes true when the number of alive processes becomes smaller than or equal to $n-k$, we can conclude that this correct process $p_i$ unblocks the situation when this condition becomes satisfied.

**On the power of $\mathcal{L}_k$** It is shown in [2] that, for $n > 2$ and $k \geq 2$, $\mathcal{L}_k$ is either weaker than or not comparable to $\Sigma$. As (a) consensus can be solved in both system models $\mathcal{AMP}[\mathcal{L}_1]$ and $\mathcal{AMP}[\Sigma, \Omega]$, and (b) $\mathcal{AMP}[\Sigma, \Omega]$ is the weakest failure detector-based model in which consensus can be solved, it follows that $\mathcal{AMP}[\mathcal{L}_1]$ is not the weakest failure detector-based model in which consensus can be solved. It also follows that, while $\mathcal{L}_{n-1} = \mathcal{L}$ is the weakest failure detector for $(n-1)$-set agreement [11], $\mathcal{L}_k$, $1 \leq k < n-1$, is not the weakest failure detector for $k$-set agreement. But, as shown in the following theorem, $\mathcal{L}_k$ seems to be not too much stronger than what is necessary.

**Theorem 1** [2] *Let $2 \leq k \leq n-1$. $k$-Set agreement can be solved in $\mathcal{AMP}[\mathcal{L}_k]$. Moreover, $(k-1)$-set agreement cannot be solved in $\mathcal{AMP}[\mathcal{L}_k]$.* (Proof in [2].)

## 5 On the implementability of $\mathcal{L}_k$ in a synchronous system

This section addresses the realism of $\mathcal{L}_k$, i.e., its implementability in a synchronous distributed system.

**Synchronous distributed system** Synchrony is an abstraction that encapsulates (and hides) specific timing assumptions. A synchronous system provides the processes with a global clock $r$ called *round number* which entails the progress of the computation. The processes progress simultaneously from round to round. During a round, a process sends a message to all processes, receives messages from other processes and executes local computation. The fundamental property associated with a synchronous system is that a message sent by a process during a round $r$ is received by the other processes during the very same round $r$. (More information on synchronous systems can be found in the book [18] that is entirely devoted to such systems.)

Let $\mathcal{SMP}$ denotes a $\mathcal{S}$ynchronous $\mathcal{M}$essage-$\mathcal{P}$assing system made up of $n$ processes, in which up to $t = n-1$ processes may crash.

**Building $\mathcal{L}_k$ in $\mathcal{SMP}$ when $k \geq n/2$** Algorithm 2 presents the code of such a construction for a process $p_i$. Initially, $alone_i$ is false. Then, during each round $r$, $p_i$ broadcasts a message ALIVE$(i)$ to inform the other processes that it was alive at the beginning of round $r$ (line 04). Then, $p_i$ receives round $r$ messages and updates $rec\_ids_i$ accordingly (line 05). Finally, if $rec\_ids_i$ contains at most $n-k$ process identities, $p_i$ sets $alone_i$ to $true$ (line 06).

```
(01) init $alone_i \leftarrow false$;
(02) when $r = 1, 2, ...$ do
(03) begin synchronous round
(04)    broadcast ALIVE$(i)$;
(05)    $rec\_ids_i \leftarrow \{ j$ such that ALIVE$(j)$ received during the current round $\}$;
(06)    if $(|rec\_ids_i| \leq n-k)$ then $alone_i \leftarrow true$ end if
(07) end synchronous round.
```

Algorithm 2: Building $\mathcal{L}_k$ in $\mathcal{SMP}$ when $k \geq n/2$ (code for $p_i$)

**Lemma 1** *Let $k \geq n/2$. Algorithm 2 builds a failure detector $\mathcal{L}_k$ in $\mathcal{SMP}$.*

**Proof** Proof of the stability property. Let $r$ be the first round during which a process $p_i$ sets $alone_i$ to $true$. This means that at least $k$ processes have crashed before sending their round $r$ message to $p_i$. Consequently, at most $n-k$ processes will ever set their boolean $alone_i$ to $true$. As $k \geq n/2 \Rightarrow n-k \leq k$, it follows that at least $k$ processes will never set their boolean variables to $true$ which proves the property.

Proof of the loneliness property. If $k$ or more processes crash during a run, for each correct process $p_i$, there is a round $r$ such that $p_i$ receives at most $n-k$ messages at every round $r' \geq r$, from which it follows that, from round $r$, $alone_i$ remains forever equal to $true$. $\Box_{Lemma\ 1}$

**Lemma 2** *It is not possible to build $\mathcal{L}_k$ in $\mathcal{SMP}$ when $k < n/2$.*

**Proof** Assuming by contradiction that there is a synchronous algorithm $\mathcal{A}$ that builds $\mathcal{L}_k$ (with $k < n/2$) in $\mathcal{SMP}$, let us consider a run $r_{k+1}$ of $\mathcal{A}$ in which the processes $p_1, p_2, ..., p_k$ have initially crashed. It follows from the loneliness property (guaranteed by $\mathcal{A}$) that there is a process (say $p_{k+1}$) whose boolean $alone_{k+1}$ becomes eventually $true$ and remains afterward forever equal to $true$.

Let us now consider a second run $r_{k+2}$ of $\mathcal{A}$ that is identical to $r_{k+1}$ until $alone_{k+1}$ becomes and remains forever equal to $true$ and such that, after that round, $p_{k+1}$ crashes. As previously, it follows from the loneliness property (guaranteed by $\mathcal{A}$) that there is a process (say $p_{k+2}$) whose boolean $alone_{k+2}$ becomes and remains forever equal to $true$. It is possible to continue to design similar runs $r_{k+3}$, etc., until $r_n$ (in each $r_x$, $k+1 \leq x \leq n$, $x-1$ processes crash).

Hence, in the run $r_n$, $n-k$ processes $p_j$ have set their boolean variable $alone_j$ to $true$. As $k < n/2 \Rightarrow n-k > k$, this contradicts the fact that the algorithm $\mathcal{A}$ guarantees the stability property of $\mathcal{L}_k$, namely, at most $k$ processes never set their boolean variable to $true$. A contradiction from which the lemma follows. $\Box_{Lemma\ 2}$

The next theorem follows from Lemma 1 and Lemma 2.

**Theorem 2** *$k \geq n/2$ is a necessary and sufficient condition for $\mathcal{L}_k$ to be realistic.*

Let us notice that the failures detectors $\mathcal{P}$ (perfect failure detector) [6], $\Sigma$, $\Sigma_k$, $\Omega$, $\Omega_k$ are realistic.

## 6 Relating $\mathcal{L}_k$ and $\Sigma_k$

This section determines the property that has to be added to $\Sigma_k$ (which is necessary to solve $k$-set agreement) in order to obtain exactly $\mathcal{L}_k$. Let $X_k$ be this property. Hence, this section solves the equation $\langle \Sigma_k, X_k \rangle \simeq \mathcal{L}_k$ where $X_k$ is the unknown and $A \simeq B$ means "$\mathcal{AMP}[A]$ and $\mathcal{AMP}[B]$ have the same computational power".

### 6.1 The property $X_k$

$X_k$ is designed to work with $\Sigma_k$. It is defined by the following property where $qr_i$ is the output of $\Sigma_k$ at $p_i$.

- Loneliness. $(|\mathcal{C}| \leq n-k) \ \Rightarrow \ (\exists\, i \in \mathcal{C}, \exists \tau :\ qr_i^{\tau} = \{i\})$.

Hence, $X_k$ requires that, when at least $k$ processes are faulty, there a time instant at which the quorum of a correct process contains only itself.

The next theorem follows directly from the lemmas 3 and 4 that are proved in the next two sections.

**Theorem 3** *$\mathcal{AMP}[\mathcal{L}_k]$ and $\mathcal{AMP}[\langle \Sigma_k, X_k \rangle]$ have the same computational power.*

### 6.2 Building $\mathcal{L}_k$ in $\mathcal{AMP}[\langle \Sigma_k, X_k \rangle]$

Algorithm 3 presents a very simple algorithm that builds $\mathcal{L}_k$ in $\mathcal{AMP}[\langle \Sigma_k, X_k \rangle]$. The output $alone_i$ of each process $p_i$ is initialized to *false* and is set to *true* if the predicate $qr_i = \{i\}$ becomes satisfied at least once.

```
init $alone_i \leftarrow false$;
when $qr_i = \{i\}$: $alone_i \leftarrow true$.
```

Algorithm 3: Building $\mathcal{L}_k$ in $\mathcal{AMP}[\langle \Sigma_k, X_k \rangle]$

**Lemma 3** *Algorithm 3 builds $\mathcal{L}_k$ in $\mathcal{AMP}[\langle \Sigma_k, X_k \rangle]$.*

**Proof** The intersection property of $\Sigma_k$ guarantees that at most $k$ processes will execute the body of the **when** statement. It follows that at most $k$ processes $p_i$ will ever set their boolean to $true$ which (combined with the initialization of the variables $alone_i$) proves the stability property of $\mathcal{L}_k$.

The loneliness property of $X_k$ implies that if $k$ or more processes crash, there a time instant at which the predicate $qr_i = \{i\}$ will be satisfied at a correct process $p_i$. Hence, this process $p_i$ will execute the body of the **when** statement and we will then have $alone_i = true$ forever, which proves the loneliness property of $\mathcal{L}_k$. $\Box_{Lemma\ 3}$

**Remark** Let us notice that the previous proof does not use the liveness property of $\Sigma_k$.

## 6.3 Building $\langle \Sigma_k, X_k \rangle$ in $\mathcal{AMP}[\mathcal{L}_k]$

**Underlying principle and description of the algorithm** Algorithm 4 relies on the observation of the predicate $qr_i = \{i\}$ that it makes stable (once satisfied, it remains forever satisfied).

The variable $qr_i$ of each process is initialized to $\Pi$ in order not to compromise the intersection property. Then, if $alone_i$ becomes true, $p_i$ sets $qr_i$ to $\{i\}$ (line 03) and, from then on, $qr_i$ will keep that value forever and $p_i$ repeatedly informs of it the other processes by broadcasting a message $\text{ALONE}(i)$.

When it receives $\text{ALONE}(j)$ from another process (line 04), $p_i$ learns that $qr_j$ is keeping forever the value $\{j\}$. If $qr_i \neq \{i\}$, $p_i$ updates accordingly $qr_i$ to $\{i, j\}$ in order to preserve the intersection property of $\Sigma_k$ (line 05).

Independently of its other statements, $p_i$ repeatedly informs the other processes that it is alive (task $T1$ where messages $\text{ALIVE}(i)$ are broadcast forever). This triggers coordination among the processes even if no boolean $alone_i$ becomes equal to $true$. Its aim is to ensure the liveness property of $\Sigma_k$.

The task $T2$ is associated with the processing of the $\text{ALIVE}(j)$ received by $p_i$. When it has received such messages from $(n - k)$ distinct processes, if $qr_i$ has not stabilized on $\{i\}$, $p_i$ resets $qr_i$ to the set containing $i$ and the processes from which alive messages have been received.

```
(01) init: $qr_i \leftarrow \Pi$; start $T1$, $T2$.

(02) when $alone_i$ becomes equal to $true$:
(03)     $qr_i \leftarrow \{i\}$; repeat forever broadcast $\text{ALONE}(i)$ end repeat.

(04) when $\text{ALONE}(j)$ with $j \neq i$ is received:
(05)     if $(qr_i \neq \{i\})$ then $qr_i \leftarrow \{i, j\}$ end if.

(06) task $T1$: repeat forever broadcast $\text{ALIVE}(i)$ end repeat.

(07) task $T2$:
(08)     repeat forever
(09)         wait until (new $\text{ALIVE}(j)$ messages with $j \neq i$ received from $n - k$ processes);
(10)         let $proc_i$ = the set of $n - k$ processes from which messages have been received;
(11)         if $(qr_i \neq \{i\})$ then $qr_i \leftarrow \{i\} \cup proc_i$ end if
(12)     end repeat.
```

Algorithm 4: Building $\langle \Sigma_k, X_k \rangle$ in $\mathcal{AMP}[\mathcal{L}_k]$

**Lemma 4** *Algorithm 4 builds $\langle \Sigma_k, X_k \rangle$ in $\mathcal{AMP}[\mathcal{L}_k]$.*

**Proof** Proof of the liveness property of $\Sigma_k$ and the loneliness property of $X_k$.
Let $A$ be the set of correct processes $p_i$ whose boolean variable $alone_i$ takes the value $true$. Due to line 02, each process $p_i$ of $A$ sets $qr_i$ to $\{i\}$ and, due to lines 05 and 11, it follows that $qr_i$ remains forever equal to $\{i\}$, We consider two cases.

- Case 1: There are $k$ or more faulty processes ($|\mathcal{C}| \leq n - k$).
  As $|\mathcal{C}| \leq n - k$, it follows from the loneliness property of $\mathcal{L}_k$ that $A \neq \emptyset$ which establishes the loneliness property of $X_k$.

  It follows from line 03 that each process $p_i$ of $A$ broadcasts forever the message $\text{ALONE}(i)$ and from line 05 that, after all the messages $\text{ALONE}()$ sent by faulty processes have been received, no process $p_j$ will add the identity of a faulty process to its quorum $qr_j$ at line 05. Moreover, after $k$ processes have crashed, the task $T2$ of each correct process $p_i$ not in $A$ eventually remains blocked forever at line 09 and $p_i$ will never add faulty processes to $qr_i$ at line 11. Consequently, there is a time after which no faulty process will ever be added to the quorum $qr_i$ of a correct process from which we conclude that there is a time after which the quorum $qr_i$ of any correct process $p_i$ contains only its identity or its identity plus the identity of another process of $A$. This concludes the proof of the liveness property of $\Sigma_k$ when $|\mathcal{C}| \leq n - k$.

*Remark.* Let us observe that eventually any correct process $p_i$ of $A$ is such that $qr_i = \{i\}$ while any process $p_i$ not in $A$ is such that $qr_i = \{i, j\}$ where $p_j$ is a process of $A$ that can change with time. In the paragraph that follows the proof, the set $A$ is called *kernel*.

- Case 2: There are less than $k$ faulty processes ($|\mathcal{C}| > n - k$).
  As $|\mathcal{C}| > n - k$, there are at least $(n - k + 1)$ correct processes that forever broadcast messages ALIVE() (line 06) and consequently no process will ever block forever at line 09. Hence, every correct process $p_i$ that does not belong to $A$ will infinitely often updates $qr_i$ to sets of $(n - k + 1)$ correct processes (line 11). Moreover, after it has received the last message ALONE() sent by a faulty process, $p_i$ no longer includes a faulty process in its quorum $qr_i$ at line 05. It follows that there is a time after which the quorum of a correct process contains only correct processes which proves the liveness property of $\Sigma_k$. The loneliness property of $X_k$ follows trivially from $\neg(|\mathcal{C}| \leq n - k)$.

Proof of the intersection property of $\Sigma_k$.
Let us assume by contradiction that there is a set of quorum values $\{q_j\}_{1 \leq j \leq k+1}$ computed by the algorithm that are such that $\forall j_1 \neq j_2 :\ q_{j_1} \cap q_{j_2} = \emptyset$. Let $P$ be the set of identities of the processes from which these $k + 1$ quorum values have been obtained. As, for any process $p_i$, we always have $i \in qr_i$ (lines 01, 03, 05 and 11), it follows that $|P| = k + 1$.

Let us assume without loss of generality that $P = \{1, 2, \ldots, k + 1\}$ and $q_j$ has been computed by $p_j$. The value of each $q_j$ has been computed at line 03, 05 or 11 (it cannot be the initial value $qr_j = \Pi$ because that quorum value would intersect any other quorum and our contradiction assumption would not be satisfied).

Let $B \subseteq P$ the set of processes whose quorum values have been computed at line 03 or 05. Hence, each of these quorum values contains a process $p_x$ whose boolean variable $alone_x$ has taken the value $true$. It then follows from the stability property of $\mathcal{L}_k$ that at most $k$ processes $p_x$ can have $alone_x = true$, from which we conclude that $0 \leq |B| \leq k$. It follows from $B \subseteq P$, $|P| = k + 1$ and $0 \leq |B| \leq k$ that $P \setminus B \neq \emptyset$.

Let $i \in P \setminus B$. Due to the definition of $B$, $p_i$ has computed $q_i$ at line 11 and, consequently, we have $|q_i| = n - k + 1$. As $\forall j_1, j_2 \in P : (j_1 \neq j_2) \Rightarrow (q_{j_1} \cap q_{j_2} = \emptyset)$ and $\forall i \in P :\ i \in q_i$, we have $q_i \cap (P \setminus \{i\}) = \emptyset$. It follows that $|q_i| \leq |\Pi| - |P \setminus \{i\}| = n - k$ which contradicts $|q_i| = n - k + 1$ and concludes the proof of the intersection property of $\Sigma_k$. $\square_{Lemma\ 4}$

**A property of $\langle \Sigma_k, X_k \rangle$** The loneliness property of $X_k$ is $(|\mathcal{C}| \leq n - k) \Rightarrow (\exists\, i \in \mathcal{C}, \exists \tau :\ qr_i^\tau = \{i\})$. Actually, Algorithm 4 ensures a stronger property, that we call *strong loneliness*, defined as follows.

- Strong loneliness. $(|\mathcal{C}| \leq n - k) \Rightarrow \big(\exists \tau :\ \forall \tau' \geq \tau, \forall\, i \in \mathcal{C} : (\exists\, j \in qr_i^{\tau'} :\ \forall\, \tau'' \geq \tau : qr_j^{\tau''} = \{j\})\big)$.

When $|\mathcal{C}| \leq n - k$, let us call *kernel* the set of processes $p_i$ that after some time have forever $qr_i = \{i\}$ (this is the set called $A$ in the proof of the liveness property of $\Sigma_k$ and the loneliness property of $X_k$ in Lemma 4). As it requires that, after some time, each correct process $p_i$ either belongs to the kernel or has a quorum containing a process of the kernel (which can change with time) it is easy to see that strong loneliness property implies loneliness defined by $X_k$.

As far as the other direction is concerned we have the following. As algorithm 4 ensures strong loneliness, it follows from Lemma 3 and Lemma 4 that this stronger property is implicitly contained in $\langle \Sigma_k, X_k \rangle$.

## 7 Relating $\mathcal{L}_k$ and $\Omega_k$

This section shows that a simple and pretty natural weakening of the stability of $\mathcal{L}_k$ gives rise to a new failure detector which is equivalent to $\Omega_k$. This establishes a strong relation linking $\mathcal{L}_k$ and $\Omega_k$.

### 7.1 The failure detector $\Diamond\mathcal{L}_k$

Weakening the stability property of $\mathcal{L}_k$ from perpetual to eventual gives rise to the failure detector $\Diamond\mathcal{L}_k$. Hence, $\Diamond\mathcal{L}_k$ is defined by the following properties.

- Eventual stability. $\exists \tau, \exists\, K \subset \Pi : (|K| = n - k) \ \wedge\ (\forall\, i \in K,\ \forall\, \tau' \geq \tau :\ alone_i^{\tau'} = \mathit{false})$.

- Loneliness. $(|\mathcal{C}| \leq n-k) \Rightarrow (\exists\, i \in \mathcal{C}, \exists \tau :\ \forall\, \tau' \geq \tau :\ alone_i^{\tau'} = true)$.

Hence, when compared to $\mathcal{L}_k$, $\Diamond\mathcal{L}_k$ ensures the boolean variables of at least $n-k$ processes remains forever equal to $false$ only after an unknown but finite time.

The next two sections show that $\Diamond\mathcal{L}_k$ and $\Omega_k$ are equivalent in $\mathcal{AMP}[\emptyset]$, i.e., $\Diamond\mathcal{L}_k$ can be built in $\mathcal{AMP}[\Omega_k]$ and $\Omega_k$ can be built in $\mathcal{AMP}[\Diamond\mathcal{L}_k]$. The next theorem follows directly from the lemmas 5 and 7 that are proved in the next two sections.

**Theorem 4** *$\mathcal{AMP}[\Diamond\mathcal{L}_k]$ and $\mathcal{AMP}[\Omega_k]$ have the same computational power.*

## 7.2 Building $\Diamond\mathcal{L}_k$ in $\mathcal{AMP}[\Omega_k]$

Algorithm 5 is a very simple construction of $\Diamond\mathcal{L}_k$ in $\mathcal{AMP}[\Omega_k]$. The boolean $alone_i$ of each process $p_i$ is initialized to $false$. Then, a process $p_i$ repeatedly updates it to $true$ or $false$ according to the fact that its identity appears or does not appear in the local output $leaders_i$ currently provided by its underlying failure detector $\Omega_k$.

```
init $alone_i \leftarrow false$;
repeat forever $alone_i \leftarrow (i \in leaders_i)$ end repeat.
```

Algorithm 5: Building $\Diamond\mathcal{L}_k$ in $\mathcal{AMP}[\Omega_k]$ (code for $p_i$)

**Lemma 5** *Algorithm 5 builds $\Diamond\mathcal{L}_k$ in $\mathcal{AMP}[\Omega_k]$.*

**Proof** Let $\tau$ be a finite time after which all faulty processes have crashed and there is a set $LD$ of $k$ process identities such that $LD \cap \mathcal{C} \neq \emptyset, \forall\, i \in \mathcal{C}, \forall\, \tau' \geq \tau$: $leaders_i^{\tau'} = LD$. (Due to the eventual leadership property of $\Omega_k$, $\tau$ and $LD$ do exist.) It follows from the algorithm that, after $\tau$, each process $p_i$ with $i \in LD \cap \mathcal{C}$ executes forever $alone_i \leftarrow true$ (Observation O1) while each process $p_i$ with $i \in \mathcal{C} \setminus LD$ executes forever $alone_i \leftarrow false$ (Observation O2). Let $\tau' \geq \tau$ be a time instant at which each correct process $p_i$ has updated at least once its boolean variable $alone_i$.

Proof of the eventual stability property of $\Diamond\mathcal{L}_k$. Let $K = \Pi \setminus LD$ (hence $|K| = n-k$). For the faulty processes of $K$, let us remember that, by definition, the boolean variable $alone_x$ of a crashed process $p_x$ is equal to $false$. The correct processes $p_i$ of $K$ are such that $i \in \mathcal{C} \setminus LD$ and it follows from observation O2 that the boolean variables of all correct processes in $\mathcal{C} \setminus LD$ remain forever equal to $false$ after $\tau'$. Hence, we have $\exists \tau'' \geq \tau'$, $\exists\ K : (|K| = n-k) \ \wedge\ (\forall\, i \in K,\ \forall\, \tau''' \geq \tau'' :\ alone_i^{\tau'''} = false)$, which proves the eventual stability property of $\Diamond\mathcal{L}_k$.

Proof of the loneliness property of $\Diamond\mathcal{L}_k$. Let us observe that, due to $\Omega_k$, $\mathcal{C} \cap LD \neq \emptyset$, hence $\exists i \in \mathcal{C} \cap LD$. It then follows from observation O1 that, after $\tau'$, $p_i$ forever executes $alone_i \leftarrow true$ which concludes the proof. $\Box_{Lemma\ 5}$

## 7.3 Building $\Omega_k$ in $\mathcal{AMP}[\Diamond\mathcal{L}_k]$

**Underlying principle** The idea of the algorithm is the following. We know from $\Diamond\mathcal{L}_k$ that there is a finite time after which there is a set $X$ of at least $n-k$ processes $p_i$ that will no longer have their boolean $alone_i$ equal to $true$. The algorithm strives to capture the complementary set $Y$ of $X$ in order to have $Y$ in the final output $LD$ of $\Omega_k$. Let us observe that we have the following when $Y$ has been captured. If $k$ or more processes are faulty, it follows from the loneliness property of $\Diamond\mathcal{L}_k$ that there is a correct process $p_i$ whose boolean becomes and remains true forever, hence we have $i \in Y$ and we can take any $LD$ such that $Y \subseteq LD$. If less than $k$ processes are faulty, it is relatively easy for the correct processes to agree on any set of $k$ processes (as any such set includes a correct process).

**A list of all subsets of size $k$** Let us consider all possible subsets of $k$ processes. Moreover, let us order all of them according to lexicographical ordering when considering each subset as a sorted array. Hence, the set $\{1, 2, \ldots, k-1, k\}$ is the first, $\{1, 2, \ldots, k-1, k+1\}$ the second, etc., until the set $\{n-k+1, \ldots, n-1, n\}$ that is the last one. Let $L$ be this sorted list of all subsets of size $k$. Moreover, let $L'$ be the list $L$ where $\{1, 2, \ldots, k-1, k\}$ is defined as the successor of the last subset $\{n-k+1, \ldots, n-1, n\}$.

Finally, let next_$\ell$d$(sbst)$ be the function that returns the subset that follows $sbst$ in $L'$.

**Local variables** To attain their goal, the processes execute asynchronous rounds. The local variable $r_i$ contains the current round number of $p_i$. The current local output computed by $p_i$ to implement $\Omega_k$ is kept in the local variable $leaders_i$.

Each process $p_i$ manages a set called $next\_set_i$. This set contains all the pairs $(r, leaders)$ received by $p_i$. The aim of this set is to allow the processes to proceed in the *very same order* from the pair $(1, \{1, 2, \dots, k-1, k\})$, to the pair $(1, \{1, 2, \dots, k-1, k+1\})$, etc., until the pair $(1, \{n-k+1, \dots, n-1, n\})$ during the first round, then from $(2, \{1, 2, \dots, k-1, k\})$ until $(2, \{n-k+1, \dots, n-1, n\})$ during the second round, etc., until they stop during a round $r$ on the very same pair $(r, leaders)$ and define accordingly $LD = leaders$.

```
(01) init $leaders_i \leftarrow \{1, 2, \dots, k\}$; $r_i \leftarrow 1$; $next\_set_i \leftarrow \emptyset$;
(02) repeat forever if $(alone_i)$ then broadcast ALONE$(i)$ end if end repeat.

(03) when ALONE$(j)$ is received: if $(j \notin leaders_i)$ then broadcast NEXT$(r_i, leaders_i)$ end if.

(04) when NEXT$(r, leaders)$ is received for the first time:
(05)   broadcast NEXT$(r, leaders)$;
(06)   $next\_set_i \leftarrow next\_set_i \cup \{(r, leaders)\}$;
(07)   while $\big((r_i, leaders_i) \in next\_set_i\big)$ do
(08)       $leaders_i \leftarrow$ next_ℓd$(leaders_i)$;
(09)       if $(leaders_i = \{1, 2, \dots, k\})$ then $r_i \leftarrow r_i + 1$ end if
(10)   end while.
```

Algorithm 6: Building $\Omega_k$ in $\mathcal{AMP}[\Diamond\mathcal{L}_k]$ (code for $p_i$)

**Behavior of a process $p_i$** Algorithm 6 describes the behavior of a process $p_i$. Let us first observe that, if no correct process $p_i$ ever receives a message ALONE$(j)$, we can conclude from the eventual stability property of $\Diamond\mathcal{L}_k$ that less than $k$ processes are faulty. Then no message is ever exchanged among the correct processes $p_i$ and they all agree on $leaders_i = LD = \{1, 2, \dots, k\}$ (that contains at least one correct process) and $\Omega_k$ is trivially implemented.

Let us now consider the general case. Each time it reads $true$ from $alone_i$, process $p_i$ broadcasts a message ALONE$(i)$. When it receives a message ALONE$(j)$ such that $p_j$ is not currently in $leaders_i$, process $p_i$ broadcasts NEXT$(r_i, leaders_i)$ (line 03). This message is to indicate to the others processes that its current set $leaders_i$ seems not to be the final one and consequently $p_i$ demands the others processes to try the next candidate leader set obtained from the list $L'$.

When $p_i$ receives a message NEXT$(r, leaders)$ for the first time, it forwards it to all in case the sender crashed during its broadcast (line 05) and saves it in $next\_set_i$ (line 06). Then if the current pair $(r_i, leaders_i)$ belongs to $next\_set_i$ (line 07), $p_i$ progresses to the next candidate set of the list $L'$ (line 08). If the new value of $leaders_i$ is $\{1, \dots, k\}$, all the elements of $L$ have been tried during round $r_i$ and consequently $p_i$ progresses to the next round (line 09) to try again the elements of $L$ with the aim to stop on one of them.

Let $(r1, leaders1) < (r2, leaders2) \overset{def}{=} (r1 < r2) \vee \big((r1 = r2) \wedge (leaders1 < leaders2)\big)$ (where $leaders1 < leaders2$ is the order of the sorted list $L$).

**Lemma 6** *Let $p_i$ be a correct process. Let $r_i = r$ and $leaders_i = \ell d$. Process $p_i$ has received and broadcast all the messages* NEXT$(r', \ell d')$ *such that $(r', \ell d') < (r, \ell d)$.*

**Proof** Due to lines 08-09, when modified, the new value of the pair $(r_i, leaders_i)$ is the direct successor pair, according to the order defined on these pairs. If follows from this observation and the initial value of the pair $(r_i, leaders_i)$ (namely, $(1, \{1, \dots, k\})$) that, when $(r_i, leaders_i) = (r, \ell d)$, the pair $(r_i, leaders_i)$ has taken all the pair values $(r', \ell d')$ such that $(1, \{1, \dots, k\}) \leq (r', \ell d') < (r, \ell d)$.

Let us now observe that the pair $(r_i, leaders_i)$ progresses to its next value only when its current value belongs to the set $next\_set_i$ (line 07). Moreover, a pair $(r', \ell d')$ is added to $next\_set_i$ only when a message NEXT$(r', \ell d')$ is received by $p_i$ for the first time, and this message is then systematically forwarded to all (lines 04-07).

It follows that, when $(r_i, leaders_i) = (r, \ell d)$, $p_i$ has received and broadcast all the messages $\text{NEXT}(r', \ell d')$ such that $(1, \{1, \dots, k\}) \leq (r', \ell d') < (r, \ell d)$. $\square_{Lemma\ 6}$

**Lemma 7** *Algorithm 6 builds $\Omega_k$ in $\mathcal{AMP}[\Diamond\mathcal{L}_k]$.*

**Proof** Let us first observe that the values taken by any set $leaders_i$ are the subsets of size $k$ defined in the list $L$. The validity property of $\Omega_k$ follows trivially.

Let $\Pi_1$ denote the set of processes that broadcast an infinity of messages $\text{ALONE}()$. It follows from the eventual stability property of $\Diamond\mathcal{L}_k$ that there is a finite time $\tau_1$ after which at least $n-k$ boolean variables $alone_i$ remain forever equal to $false$, hence $|\Pi_1| \leq k$. Moreover, there is a time $\tau_2 \geq \tau_1$ from which (a) all received messages $\text{ALONE}()$ are from processes of $\Pi_1$ and (b) no process broadcasts at line 03 a message $\text{NEXT}(r, ld)$ with $\Pi_1 \subset ld$ (this follows from the predicate used in the **if** statement of line 03).

Let $(r0, \ell d0)$ be the greatest pair such that $\Pi_1 \subseteq \ell d0$ and there is a message $\text{NEXT}(r0, \ell d0)$ that entailed an update of $leaders_i$ at some correct process $p_i$ (line 08). If there is no such pair, let $(r0, \ell d0) = (0, \alpha)$ where $\text{next\_}\ell\text{d}(\alpha) = \{1, \dots, k\}$. Let $(r1, \ell d1)$ be the smallest pair such that $(r0, \ell d0) < (r1, \ell d1)$ and $\Pi_1 \subset \ell d1$.

It follows from the definition of $(r0, \ell d0)$ and Lemma 6 that each message $\text{NEXT}(r, \ell d)$ such that $(r, \ell d) \leq (r0, \ell d0)$ has been sent by a correct process from which we conclude that each correct process $p_i$ eventually updates $leaders_i$ to $\text{next\_}\ell\text{d}(\ell d0)$.

If $\Pi_1 \not\subset leaders_i$ at some correct process $p_i$, it follows from the definition of $\Pi_1$ that $p_i$ eventually receives a message $\text{ALONE}(j)$ sent by $p_j$ such that $j \in \Pi_1 \setminus leaders_i$. When it receives such a message (line 03), process $p_i$ broadcasts $\text{NEXT}(-, leaders_i)$. It follows that, for each pair $(r, \ell d)$ such that $(r0, \ell d0) < (r, \ell d) < (r1, \ell d1)$, each correct process broadcasts (at line 03 or line 05) a message $\text{NEXT}(r, \ell d)$. Moreover, once all these messages have been received, the correct processes $p_i$ eventually agree on the same set $leaders_i = \ell d1$ and no longer broadcast $\text{NEXT}(-, -)$ messages.

If there are $k$ or more faulty processes, the loneliness property of $\Diamond\mathcal{L}_k$ implies that $\Pi_1 \cap \mathcal{C} \neq \emptyset$ and consequently $\ell d1$ contains at least one correct process. If there are less than $k$ faulty processes, as $|\ell d1| = k$, $\ell d1$ contains at least one correct process, which concludes the proof of the lemma. $\square_{Lemma\ 7}$

## 8 Conclusion

This paper has investigated the computability power and explored the relations linking three failure detectors that have been proposed to solve the $k$-set agreement problem in asynchronous crash-prone message-passing systems. (The $k$-set agreement problem is a coordination problem that generalizes the consensus problem.) These three failure detectors are the generalized quorum failure detector $\Sigma_k$, the generalized loneliness failure detector $\mathcal{L}_k$ and the generalized eventual leader failure detector $\Omega_k$.

The paper has (a) shown that the failure detector $\Omega_k$ and the eventual version of $\mathcal{L}_k$ have the same computational power; (b) shown that $\mathcal{L}_k$ is realistic if and only if $k \geq n/2$; and (c) given an exact characterization of the difference between $\mathcal{L}_k$ and $\Sigma_k$. Hence, this paper provides us with a better understanding of these failure detectors in the quest for the weakest failure detector to solve the $k$-set agreement problem in asynchronous message-passing crash-prone systems.

## References

[1] Attiya H. and Welch J., Distributed Computing: Fundamentals, Simulations and Advanced Topics, (2d Edition), *Wiley-Interscience*, 414 pages, 2004.

[2] Biely M., Robinson P. and Schmid U., Weak Synchrony Models and Failure Detectors for Message Passing (k-)Set Agreement. *Proc. 11th Int'l Conference on Principles of Distributed Systems (OPODIS'09)*, Springer Verlag LNCS #5923, pp. 285-299, 2009.

[3] Bonnet F. and Raynal M., On the Road to the Weakest Failure Detector for $k$-Set Agreement in Message-passing Systems. To appear in *Theoretical Computer Science*, 2010.

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

[5] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for $t$-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, San Diego (CA), pp. 91-100, 1993.

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

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

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

[9] Delporte-Gallet C., Fauconnier H. and Guerraoui R., A Realistic Look At Failure Detectors. *Proc. 42th Int'l IEEE/IFIP Conference on Dependable Systems and Networks (DSN'02)*, IEEE Press, pp. 345-353, 2002.

[10] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Tight Failure Detection Bounds on Atomic Object Implementations. *Journal of the ACM*, 57(4):Article 22, 2010.

[11] 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.

[12] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923,, 1999.

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

[14] Neiger G., Failure Detectors and the Wait-free Hierarchy. *14th ACM Symposium on Principles of Distributed Computing (PODC'95)*, ACM Press, pp. 100-109, Las Vegas -NV), 1995.

[15] Raynal M., Set Agreement. *Encyclopedia of Algorithms*, Springer-Verlag, pp. 829-831, 2008 (ISBN 978-0- 387-30770-1).

[16] Raynal M., Failure Detectors for Asynchronous Distributed Systems: an Introduction. *Wiley Encyclopdia of Computer Science and Engineering*, Vol. 2, pp. 1181-1191, 2009 (ISBN 978-0-471-38393-2).

[17] Raynal M., Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems. *Morgan & Claypool Publishers*, 251 pages, 2010 (ISBN 978-1-60845-293-4).

[18] Raynal M., Fault-Tolerant Agreement in Synchronous Message-Passing Systems. *Morgan & Claypool Publishers*, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[19] Raynal M., Failure Detectors to solve Asynchronous $k$-set Agreement: a Glimpse of Recent Results. *The Bulletin of EATCS*, 103:75-95, 2011.

[20] Saks M. and Zaharoglou F., Wait-Free $k$-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.