THE DISTRIBUTED COMPUTING COLUMN

BY

PANAGIOTA FATOUROU

Department of Computer Science, University of Crete
P.O. Box 2208 GR-714 09 Heraklion, Crete, Greece
and
Institute of Computer Science (ICS)
Foundation for Research and Technology (FORTH)
N. Plastira 100. Vassilika Vouton
GR-700 13 Heraklion, Crete, Greece
faturu@csd.uoc.gr
FAILURE DETECTORS TO SOLVE ASYNCHRONOUS $k$-SET AGREEMENT: A GLIMPSE OF RECENT RESULTS

Michel Raynal
Senior Member, Institut Universitaire de France
IRISA, Université de Rennes, 35042 Rennes Cedex, France
{raynal}@irisa.fr

Abstract

In the $k$-set agreement problem, each process proposes a value and has to decide a value in such a way that a decided value is a proposed value and at most $k$ different values are decided. This problem can easily be solved in synchronous systems or in asynchronous systems prone to $t$ process crashes when $t < k$. In contrast, it has been shown that $k$-set agreement cannot be solved in asynchronous systems when $k \leq t$. Hence, since several years, the failure detector-based approach has been investigated to circumvent this impossibility. This approach consists in enriching the underlying asynchronous system with an additional module per process that provides it with information on failures. Hence, without becoming synchronous, the enriched system is no longer fully asynchronous. This paper surveys this approach in both asynchronous shared memory systems and asynchronous message passing systems. It presents and discusses recent results and associated $k$-set agreement algorithms.

Keywords: Asynchronous system, Eventual leader, Failure detector, Message passing system, Quorum, $k$-Set agreement, Shared memory system, Wait-freedom.

1 Introduction

The $k$-set agreement problem This problem, that involves $n$ processes that may fail by crashing, is a coordination problem (sometimes called decision task) introduced by S. Chaudhuri [11]. Her aim was to explore the relation linking the number of process failures and the minimal number of values that the processes are
allowed to decide. This problem is defined as follows [11, 34, 39]. 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 problem) while \( k = n - 1 \) corresponds to its weakest non-trivial instance (called set agreement problem).

Let \( t \) be an upper bound on the number of processes that may crash in a run, \( 1 \leq t < n \). Hence, \( t \) is a model parameter. 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 any run.

Algorithms that solve the \( k \)-set agreement problem in message passing synchronous systems when \( k \geq t \) are presented in [3, 27, 37]. These algorithms are based on sequence of synchronous rounds. It is shown in these books (see also [12]) that \( \lfloor \frac{k}{t} \rfloor + 1 \) rounds are necessary and sufficient to solve \( k \)-set agreement. (It is shown in [37] that this lower bound is still valid in more severe failure models such as the general omission failure model.) For asynchronous systems, the situation is different. When \( t \geq k \), the \( k \)-set agreement problem has no solution [7, 24, 40].

The failure detector-based approach  A failure detector is a distributed oracle that gives alive processes hints on process failures [9, 35]. Failure detectors have been investigated to solve the \( k \)-set agreement problem since 2000 [30]. (Random oracles to solve the \( k \)-set agreement problem have also been investigated [31].) Lower bounds to solve \( k \)-set agreement in asynchronous message passing systems enriched with limited accuracy failure detectors have been conjectured in [30] and proved in [23]. The question of the weakest failure detector class for the \( k \)-set agreement problem \((k > 1)\) has been stated first in [38].

Let \( P \) be a problem that is impossible to solve in a pure asynchronous system. A non-trivial failure detector is a failure detector that allows a problem such as \( P \) to be solved. Implementing a non-trivial failure detector requires that the underlying system satisfies appropriate behavioral assumptions. The interested reader will find such assumptions and corresponding algorithms in [36] (Chapter 7) for asynchronous message passing systems and in [18] for asynchronous shared memory systems.
The Bulletin of the EATCS

Content of the paper  The paper is on the use of failure detectors that allows $k$-set agreement to be solved in asynchronous systems prone to process crashes. It is made up of 5 sections. Section 2 presents the process model and defines the $k$-set agreement problem. Then the two main sections of the paper follow. Section 3 considers the case where the communication medium is a read/write shared memory. It presents the weakest failure detector for the $k$-set agreement problem in such a setting. This failure detector, denoted $\Omega_k$, which was conjectured to be the weakest in [33], has been proved to be the weakest in [20]. A corresponding $k$-set agreement algorithm is also presented in that section. “Weakest” means here that any failure detector that can be used to solve the $k$-set agreement problem in a crash-prone asynchronous shared memory system provides us with information on failures from which $\Omega_k$ can be built. (More formally, showing that a failure detector is as strong as another one is based on reductions, e.g., [6, 10, 14]).

<table>
<thead>
<tr>
<th>FD class</th>
<th>Introduced in</th>
<th>Presented in Sec.</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega$</td>
<td>[10]</td>
<td>4.3</td>
<td>Weakest for Consensus in SM</td>
</tr>
<tr>
<td>$\Omega_k$</td>
<td>[32]</td>
<td>4.5</td>
<td>Solves $k$-set agreement in SM</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>[21]</td>
<td>3.2</td>
<td>Sufficient for $(n-1)$-set agreement in SM</td>
</tr>
<tr>
<td>$\Omega_{n-1}$</td>
<td>[41]</td>
<td>3.2</td>
<td>Weakest for $(n-1)$-set agreement in SM</td>
</tr>
<tr>
<td>$\Omega_k$</td>
<td>[33]</td>
<td>3.2</td>
<td>Weakest for $k$-set agreement in SM</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>[14]</td>
<td>4.2</td>
<td>Weakest for Register in MP</td>
</tr>
<tr>
<td>$(\Sigma, \Omega)$</td>
<td>[15]</td>
<td>4.3</td>
<td>Weakest for consensus in MP</td>
</tr>
<tr>
<td>$\Sigma_k$</td>
<td>[5]</td>
<td>4.5</td>
<td>Necessary for $k$-set agreement in MP</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>[16]</td>
<td>4.3</td>
<td>Weakest for $(n-1)$-set agreement in MP</td>
</tr>
<tr>
<td>$\mathcal{L}_k$</td>
<td>[4]</td>
<td>4.4</td>
<td>Solves $k$-set agreement in MP</td>
</tr>
<tr>
<td>$\Pi_k$</td>
<td>[5]</td>
<td>4.5</td>
<td>Same power as to $(\Sigma_k, \Omega_k)$</td>
</tr>
</tbody>
</table>

Table 1: Global picture: failure detector classes related to $k$-set agreement

Section 4 considers then the case where the communication medium is a reliable asynchronous message passing network. Maybe surprisingly, the weakest failure detector for solving $k$-set agreement is different in shared memory systems and message passing systems. Moreover, the corresponding weakest failure detectors are known only for the case $k = 1$ (consensus) and $k = n - 1$ (set agreement). This section presents the most recent results known for the other cases, which leaves open the discovery of the corresponding weakest failure detector. Several failure detector proposals and algorithms are also described. Finally, Section 5 concludes the paper. (To keep the presentation simple, the theorems and algorithms are presented without their proof. The reader will find them in the papers in which they have been introduced.)

To help the reader have a global view, Table 1 summarizes the main failure detector classes presented in this paper (FD, SM and MP stand for failure detec-
tor, shared memory and message passing, respectively). The reader interested in
the computability power and the robustness of $k$-set agreement-oriented failure
detector classes can consult [29].

2 Process model and $k$-set agreement

Process model The system consists of a set $\Pi = \{p_1, \ldots, p_n\}$ of $n$ asynchronous
processes. The integer $i$ is called the index or identity of process $p_i$.

Each process executes a sequence of atomic steps (each of which may contain
any finite amount of local computation and either a read from or a write to the
shared memory in case of a shared memory system, or a message send or receive
in case of a message passing system). A process executes its code until it possibly
-crashes. After it has crashed a process executes no more step.

A run is a sequence of steps issued by processes such that, according to the
communication model, any value read has been previously written or any message
received has been previously sent. A process that crashes during a run is faulty in
that run, otherwise it is correct. Given a run, $C$ denotes the set of processes that
are correct in that run.

>From a notation point of view, local variables are denoted with lowercase
letters and the process index $i$ is used as a subscript.

External global time For presentation and analysis purposes, we assume that
there is a discrete global clock which ticks every time a process takes a step. This
clock is not accessible by the processes.

Let $\text{var}_i^\tau$ be a local variable of process $p_i$. $\text{var}_i^\tau$ denotes the value of $\text{var}_i$ at time
$\tau$.

Failure pattern The failure pattern associated with a run, is a function $F(\tau)$
that outputs the set of processes crashed at time $\tau$, $\tau \geq 0$. As processes do not
recover, we have $F(\tau) \subseteq F(\tau + 1)$. $\mathcal{F} = \bigcup_{\tau \geq 0} F(\tau)$ (the set of faulty processes in
the corresponding run). Let us observe that $C = \Pi \setminus \mathcal{F}$.

The model parameter $t$, $1 \leq t < n$, denotes the upper bound on the number of
processes that may crash in a run. When $t = n - 1$, the set of all possible failure
patterns is called the wait-free environment. We say that an algorithm wait-free
solves a problem if any correct process terminates with the right result whatever
the number of faulty processes.

Failure detector A failure detector is a device (oracle) that provides each pro-
cess $p_i$ with read-only local variables containing information of process failures
The k-set agreement problem As already indicated, the k-set agreement problem has been introduced by S. Chaudhuri [11]. 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.

A process \( p_i \) invokes the operation \( \text{X.set\_agreement}_k(v_i) \) (where \( v_i \) is the value it proposes) to participate in a k-set agreement instance denoted \( X \). This operation returns to the invoking process \( p_i \) the value that \( p_i \) decides. The k-set agreement problem is a *one-shot problem*, which means that each problem instance is independent of the other instances.

3 **k-Set agreement in asynchronous shared memory systems**

This section presents first the class of the weakest failure detectors for k-set agreement in crash-prone shared memory systems. This class is denoted \( \Omega_k \). The section describes then an \( \Omega_k \)-based k-set agreement for these systems.

3.1 **Communication model**

The processes communicate by reading and writing atomic registers [25]. This means that all shared memory accesses appear as if they have been executed one after the other and this total order respects the partial order imposed by their execution. From a notation point of view, shared variables are denoted with uppercase letters.

The corresponding shared memory model (in which at most \( t \) processes may fail) is denoted by \( \text{ASM}_{n,t}[0] \). When the system is enriched with a failure detector \( X \), it will be denoted \( \text{ASM}_{n,t}[X] \). \( \text{ASM}_{n,t-1}[0] \) is consequently the asynchronous wait-free shared memory model (wait-free because algorithms designed for this model have to be correct and allow correct processes to terminate despite the occurrence of up to \( t = n - 1 \) process failures).
Despite the fact that the sentence “an algorithm wait-free solves a problem in a system model in which \( t = n - 1 \)” is a pleonasm (as it contains both “wait-free” and \( "t = n - 1" \)), we voluntarily use it in the following to insist on wait-free solvability.

### 3.2 The failure detector class \( \Omega_k \)

A failure detector called anti-\( \Omega \) (denoted here \( \Omega_{n-1} \)) has been introduced by Zielinsky [41] and shown to be the weakest to solve the \((n - 1)\)-set agreement problem. As indicated by Zielinsky, the failure detector class \( \mathcal{T} \), that has been previously proposed in [21], was instrumental in the discovery of \( \Omega_{n-1} \). A failure detector of the class \( \mathcal{T} \) eventually informs the processes that, in the current run, some set of processes cannot be the set of correct processes. It is shown in [21] that \( \mathcal{T} \) is sufficient for solving \((n - 1)\)-set agreement.

A generalization of \( \Omega_{n-1} \) denoted \( \Omega_k \) has been introduced in [33] where it is conjectured to be the weakest for shared memory \( k \)-set agreement. This conjecture has been proved by Gafni an Kuznetsov in [20].

**Definition** A failure detector of the class \( \Omega_k \) provides each process \( p_i \) with a read-only variable denoted \( mv\_leader_i \) (moving leaders) such that the following properties are satisfied.

- **Validity.** \( \forall i : \forall \tau : mv\_leaders_\tau^i \) is a set of \( k \) process identities.

- **Weak Eventual leadership.** \( \exists \tau : \exists \ell \in C : \forall \tau' \geq \tau : \forall i \in C : \ell \in mv\_leaders_{\tau'}^i \).

The weak eventual leadership property states that there is a time \( \tau \) and a correct process \( p_\ell \) such that, after time \( \tau \), no correct process “suspects \( p_\ell \) to have crashed”. Let us notice that the time \( \tau \) is never revealed to the processes. Moreover, no process explicitly knows the fact that a correct process is included (and will stay forever after) in all sets identified by \( mv\_leaders_\tau_{i} \), \( \forall i \in C \).

After a correct process \( p_\ell \) remains forever in all sets \( mv\_leaders_\tau_{i} \), no process knows this fact explicitly.

\( \Omega_1 \) is the same as \( \Omega \) (the eventual leader failure detector which has been proved to be the weakest failure detector for solving the consensus problem in asynchronous shared memory systems [10, 26]).

**Theorem 1.** [Gafni-Kuznetsov 2009]\(^1\) When considering failure detector-enriched systems, the model \( \mathcal{ASM}_{n,n-1}[\Omega_k] \) is the weakest asynchronous shared memory

---

\(^1\)This result has also been proved in [17] and [19] using different techniques.
model in which the k-set agreement problem can be wait-free solved. (Proof in
[20].)

In this theorem (and following ones), we use the sentence “When consider-
ing failure detector-enriched systems” to insist on the fact that we are interested
in systems enriched only with failure detectors. This is because one could imag-
ine other possible types of “enrichment” that would allow k-set agreement to be
solved in the corresponding enriched systems.

3.3 An Ωk-based k-set agreement algorithm

Underlying principle The principle that underlies the design of the Ωk-based
k-set agreement algorithm that follows [1, 41] is pretty simple: each process pi
participates in k independent parallel consensus instances, pi proposes the same
value to every instance and decides the value returned from the first instance that
locally terminates. To that end the algorithm is made up of two parts: an algorithm
that gives Ωk a vector shape denoted vector_Ωk, and a vector_Ωk-based algorithm
that solves k-set agreement.

From Ωk to vector_Ωk vector_Ωk is a vector denoted Omega[1..k] such that (a)
each Omega[x] returns a process identity each time it is called, and (b) at least
one Omega[x] behaves as Ω1 (which is the weakest failure detector that allows
consensus to be wait-free solved in ASMSn−1[∅]). This means that there is
at least one Omega[x] that outputs the same correct process pij at each correct
process after some finite time.

| Task Ti | % the task Ti is executed by pi % repeat forever seti ← my_leadersi; for each j ≠ set, do SUSPICIONS[i][j] ← SUSPICIONS[i][j] + 1 end for end repeat.
| when Omega[x] is queried by pi; | for each j ∈ [1..n] do total[j] ← Σ1≤istance[SUSPICIONS[x][j] end for; let p1, . . . , pn = permutation on the n proc. s. t. (total[j1], j1) < · · · < (total[jn], jn); return(ji).

Figure 1: From Ωk to vector_Ωk (code for pi) [41]
The code of the wait-free algorithm that constructs vector\( \Omega_k \) from \( \Omega_k \) is described in Figure 1. A shared array \( \text{SUSPICIONS}[i][1..n] \) is associated with each process \( p_i \). Only \( p_i \) can write it, but any process can read it. \( \text{SUSPICIONS}[i][j] \) contains the number of times process \( p_j \) has been suspected by process \( p_i \) ("suspected by \( p_i \)" means here "not belonging to the output of \( \Omega_k \) invoked by \( p_i \)).

When process \( p_i \) queries \( \Omega[x] \), it first computes the total number of suspicions of every process \( p_j \) (\( \text{total}[j] \)) and then orders the processes from the less to the more suspected. Process identities are used to obtain a total order (let us remember that lexicographical order \( (a, i) < (b, j) \) is defined as \( (a < b) \lor ((a = b) \land (i < j)) \)).

The intuition that underlies this algorithm is the following. Let \( p_\ell \) be a correct process that, after some finite time, belongs to the set \( \text{mv\_leaders}_i \) of every correct process \( p_i \). (Due to the definition of \( \Omega_k \), such a process \( p_\ell \) does exist.) Consequently, after some finite time, the quantity \( \Sigma_{1 \leq x \leq n} \text{SUSPICIONS}[x][\ell] \) stops increasing and the pair \( (\text{total}[\ell], \ell) \) will then be one of the \( k \) smallest pairs computed by any process.

```plaintext
when operation set_agreement(k_i) is invoked by \( p_i \):
    for each \( x \in [1..k] \) do CONS[x].propose(v_i) end for;
    let \( v \) be the value returned by the first consensus instance CONS[x] that terminates;
return(v).
```

Figure 2: Wait-free vector\( \Omega_k \)-based \( k \)-set agreement (code for \( p_i \)).

The vector\( \Omega_k \)-based \( k \)-set agreement algorithm is described in Figure 2. As already indicated, it consists of \( k \) parallel and independent consensus instances denoted \( \text{CONS}[1..k] \). Process \( p_i \) proposes \( v_i \) to each consensus instance and decides the first value returned by any of these instances. The \( \Omega \)-based consensus instance \( \text{CONS}[x] \) uses \( \Omega[x] \) as its underlying failure detector \( \Omega \). It is easy to see that at most \( k \) values can be decided, and that a process that does not crash decides a value. This is because at least one \( \Omega[x] \) -not known in advance- behaves as \( \Omega \) and \( \Omega \) allows consensus to be wait-free solved in asynchronous shared memory systems \([22, 26]\).

## 4 \( k \)-Set agreement in asynchronous message passing systems

This section focuses on failure detectors for the \( k \)-set agreement problem in crash-prone asynchronous message passing systems. In contrast to shared memory sys-
tems, the weakest class of failure detectors for these systems is not yet known. Hence, this section presents the last results in that direction.

The weakest class of failure detectors for $k$-set agreement suited to crash-prone asynchronous message passing systems is known only for $k = 1$ and $k = n - 1$. Let us remember that 1-set agreement is the consensus problem, i.e., the more constraining (or strongest) agreement problem, while $(n - 1)$-set agreement is the weakest in the sense that the processes have to eliminate a single value from the proposed values (when we assume that each process proposes a distinct value).

4.1 Communication model

The processes communicate by sending and receiving messages through channels. Every pair of processes is connected by a bidirectional channel. The channels are failure-free (there is no creation, alteration, duplication or loss of messages) and asynchronous (albeit the time taken by a message to travel from its sender to its destination process is finite, there is no bound on transfer delays). The notation “broadcast $\text{MSG\_TYPE}(m)$” is used to send a message $m$ (the type of which is $\text{MSG\_TYPE}$) to all the processes. It is a (non-atomic) shortcut for “for each $j \in \{1,\ldots,n\} \text{ do send } \text{MSG\_TYPE}(m) \text{ to } p_j \text{ end for}”.

4.2 From shared memory to message passing

The main question It is shown in [2] that $t < n/2$ is a necessary and sufficient requirement on the value of the model parameter $t$ in order to simulate a shared read/write register on top of a crash-prone asynchronous message passing system. Hence, a fundamental question is: “Which is the weakest failure detector that allows a register to be built in $\text{AMP}_{n,t}[0]$?” This question has been answered by Delporte, Fauconnier and Guerraoui who have shown that $\Sigma$ is this failure detector [14].

The failure detector class $\Sigma$ A failure detector of the class $\Sigma$ provides each process $p_i$ with a set $q_{ri}$ (called quorum) such that the set of variables $q_{ri}$ satisfies the following properties. After a process $p_i$ has crashed (if it ever does), it is assumed that $q_{ri}$ remains forever equal to $\{1,\ldots,n\}$.
• Intersection. \( \forall i, j : \forall \tau_i, \tau_j : (qr^\tau_i \cap qr^\tau_j \neq \emptyset) \).
• Liveness. \( \exists \tau : \forall \tau' \geq \tau : \forall i \in C : qr^\tau_{i} \subseteq C \).

The first property states that any pair of values of two quorums, each taken at any time, do intersect. The second property states that the quorum of any correct process eventually contains only correct processes. When we look at a \( \Sigma \)-based algorithm, the first property is used to ensure its safety/consistency while the second one is used to guarantee its progress.

**Theorem 2.** [Delporte-Fauconnier-Guerraoui 2010] When considering failure detector-enriched systems, \( \text{AMP}_{n-1}[\Sigma] \) is the weakest asynchronous message passing system model on top of which a shared read/write register can be wait-free built. (Proof in [6, 14].)

More developments and algorithms building a register in \( \text{ASM}_{n,t}[t < n/2] \) and \( \text{ASM}_{n,n-1}[\Sigma] \) can be found in [37].

### 4.3 The cases of consensus \((k = 1)\) and set agreement \((k = n - 1)\)

**The case of consensus \((k = 1)\) in message passing systems**

The failure detector \( \Omega \) has been introduced by Chandra, Hadzilacos and Toueg in [10]. It provides each process \( p_i \) with a read-only local variable \( \text{leader}_i \) that always contains a process identity. Moreover, after some unknown but finite time, the variables \( \text{leader}_i \) of all correct processes \( p_i \) contain the same process identity which is the identity of a correct process. As already said, when considering \( \Omega \), \( \Omega_1 \) is \( \Omega \). The fundamental result associated with \( \Omega \) is the following.

**Theorem 3.** [Chandra-Hadzilacos-Toueg 1996] When enriching a system with a failure detector \( \Omega \) is the weakest failure detector the system \( \text{AMP}_{n-1}[t < n/2] \) has to be enriched with in order for consensus to be solved. (Proof in [10].)

This result is extended in [14] where it is shown that the pair of failure detectors \( (\Omega, \Sigma) \) is the weakest to solve consensus in \( \text{ASM}_{n,n-1}[0] \). The corresponding \( (\Omega, \Sigma) \) failure detector provides two outputs, one for \( \Omega \) the other one for \( \Sigma \). Intuitively, \( \Sigma \) is used to simulate a shared memory while \( \Omega \) is used to allow correct processes to terminate. Several consensus algorithms for both system models \( \text{ASM}_{n,t}[t < n/2] \) and \( \text{ASM}_{n,n-1}[\Sigma, \Omega] \) are described in [36].

**The case of set agreement \((k = n - 1)\) in message passing systems**

The failure detector class \( \mathcal{L} \) (for loneliness), that has been introduced in [16], is defined as follows. Each process \( p_i \) is provided with a read-only boolean variable \( \text{alone}_i \), and...
The Bulletin of the EATCS

these boolean variables satisfy the following properties. (After a process \( p_i \) has crashed (if it ever crashes) its boolean \( alone_i \) is assumed to remain forever equal to \( false \).)

- **Stability.** \( \exists i : \forall \tau : alone_i^\tau = false \).
- **Loneliness.** \( |C| = |\{ i \}| \Rightarrow (\exists \tau : \forall \tau' \geq \tau : alone_\tau^\tau = true) \).

The stability property states that there is at least one process \( p_i \) whose boolean local variable \( alone_i \) remains forever equal to \( false \), while the loneliness property states that, if only one process (say \( p_i \)) is correct, its boolean local variable \( alone_i \) eventually outputs \( true \) forever. Let us notice that nothing prevents the value of a boolean local variable \( alone_i \) from changing infinitely often (as long as the corresponding process \( p_i \) is neither the one whose boolean local variable remains always false, nor the only correct process \( p_i \) in the the case where \( n - 1 \) processes crash). The main result associated with \( L \) is the following Theorem.

**Theorem 4.** [Delporte-Fauconnier-Guerraoui-Tielmann 2008] When considering failure detector-enriched systems, \( ASM_{n,n-1}[L] \) is the weakest asynchronous message passing system model in which \((n-1)\)-set agreement can be wait-free solved. (Proof in [16].)

An algorithm that solves the \((n-1)\)-set agreement problem in \( ASM_{n,n-1}[L] \) is described in [16], where it is also shown that \( L \) is strictly stronger than \( \Omega_{n-1} \) and strictly weaker than \( \Sigma \). This algorithm has given rise to a more general algorithm for \( k \)-set agreement (described in Figure 3) and can be obtained from it by taking \( k = n - 1 \).

### 4.4 The class of failure detectors \( L_k \) \((1 \leq k \leq n - 1)\)

More general failure detectors than the pair \((\Omega, \Sigma)\) (that is optimal for \( k = 1 \)) and \( L \) (that is optimal for \( k = n - 1 \)) have also been proposed (e.g., [5]). Unfortunately, none of them is the weakest for \( 1 \leq k < n - 1 \). This section presents one of them proposed by Biely, Robinson and Schmid [4].

**The failure detector class \( L_k \)** This class of failure detectors is a simple generalization of \( L \). More specifically it holds that \( L = L_{n-1} \). A failure detector of the class \( L_k \) is called \((n-k)\)-loneliness failure detector. It is formally defined as follows.

- **Stability.** \( \exists \) a set of processes \( K : |K| = n-k : \forall i \in K : \forall \tau : alone_i^\tau = false \).
- **Loneliness.** \( |C| \leq n-k \Rightarrow (\exists \ell \in C : \exists \tau : \forall \tau' \geq \tau : alone_\tau^\tau = true) \).
As we can see, this failure detector generalizes \( L \). While the loneliness property of \( L \) detects the case where only one process remains alive forever, the loneliness property of \( L_k \) detects the case where at most \( n - k \) processes remain alive forever.

An \( L_k \)-based \( k \)-set agreement algorithm  The \( L_k \)-based algorithm described in Figure 3 solves the \( k \)-set agreement problem [4]. This algorithm is based on a sequence of asynchronous rounds (\( r_i \) denotes the current round number of \( p_i \)). The local variable \( \text{est}_i \) is \( p_i \)'s current estimate of its decision value. The execution of the statement \( \text{return}(v) \) returns the value \( v \) and terminates the invocation of the \( \text{set\_agreement}_k() \).

```
when operation \( \text{set\_agreement}_k(v_i) \) is invoked by \( p_i \):
  (01) \( \text{est}_i \leftarrow v_i; r_i \leftarrow 1; \)
  (02) repeat forever
  (03) for each \( j \neq i \) do send \( \text{est}(r_i, \text{est}_i) \) to \( p_j \) end for;
  (04) wait until ( \( n - k \) messages \( \text{est}(r_i, -) \) have been received);
  (05) \( \text{est}_i \leftarrow \min(\text{est}_i, \text{the est}_j \text{received at the previous line}) \);
  (06) if \( r_i = k + 1 \)
  (07) then for each \( j \neq i \) do send \( \text{dec}(\text{est}_i) \) to \( p_j \) end for;
  (08) return(\( \text{est}_i \))
  (09) else \( r_i \leftarrow r_i + 1 \)
  (10) end if
  (11) end repeat.
```

```
when (alone, \( v \ \text{dec}(v) \) is received):
  (12) if (\( \text{dec}(v) \) received) then \( \text{est}_i \leftarrow v \) end if;
  (13) for each \( j \neq i \) do send \( \text{dec}(\text{est}_i) \) to \( p_j \) end for;
  (14) return(\( \text{est}_i \)).
```

Figure 3: An \( L_k \)-based \( k \)-set agreement algorithm (code for \( p_i \)) [4]

In each round, each non-crashed process \( p_i \) first broadcasts a message \( \text{est}(r_i, \text{est}_i) \) (line 03) to inform the other processes of its current estimate \( \text{est}_i \) 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 06). Then if \( r_i < k + 1 \), it proceeds to the next asynchronous round (line 09). If \( r_i = k + 1 \) it broadcasts \( \text{dec}(\text{est}_i) \) 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 crash. Such a permanent blocking is prevented by the use of $\text{acc}(\cdot)$ messages (that ensures that, as soon as a process decides, all correct processes eventually decide), and the use of the failure detector. Let us also observe that the boolean $\text{alone}_i$ of a correct process $p_i$ becomes true when the number of correct processes is smaller than or equal to $n - k$. In that case, this correct process $p_i$ unblocks the situation.

**On the power of $L_k$.** It is shown in [4] that, for $n > 2$ and $k \geq 2$, $L_k$ is either weaker than or not comparable to $\Sigma$. As (a) consensus can be solved in both system models $\text{AMP}_{n,n-1}[L_k]$ and $\text{AMP}_{n,n-1}[\Omega, \Sigma]$, and (b) $\text{AMP}_{n,n-1}[\Omega, \Sigma]$ is the weakest failure detector-based model in which consensus can be solved, it follows that $L_1$ is not the weakest failure detector with which $\text{AMP}_{n,n-1}[\emptyset]$ has to be enriched in order to solve consensus.

It also follows that, while $L_{n-1} = L$ is the weakest failure detector for $(n-1)$-set agreement [16], $L_k$, $1 \leq k < n-1$, is not the weakest failure detector for $k$-set agreement. But, as shown in the following theorem, $L_k$ seems to be not too much stronger than what is necessary.

**Theorem 5.** [Biely-Robinson-Schmid 2009] Let $2 \leq k \leq n-1$. $k$-Set agreement can be wait-free solved in $\text{AMP}_{n,n-1}[L_k]$ but $(k-1)$-set agreement cannot wait-free solved in $\text{AMP}_{n,n-1}[L_k]$. (Proof in [4].)

### 4.5 An important step: $\Sigma_k$ is necessary for $k$-set agreement

**Where is the difficulty** As far as the agreement property is concerned (at most $k$ values are decided), a main difficulty in the quest for the weakest failure detector that solves $k$-set agreement in a message passing system lies in capturing the shared memory properties needed to solve this problem. (Said, differently, while implementing shared registers in a message passing system is stronger than necessary when one wants to solve $k$-set agreement, it is not yet known how to weaken the register properties in such a way that, once these “weak” registers have been implemented in a message passing system, $k$-set agreement could be solved in such a system).

An effort in that direction is presented in [13] where are investigated the relations between the implementation of a register and $k$-set agreement in asynchronous crash-prone message passing systems. Let an $x$-register be a register that (a) is shared by $x$ processes only and (b) is implemented by processes that communicate by exchanging messages.
Let us remember that $\mathcal{AMP}_{n,n-1}[X]$ is $\mathcal{AMP}_{n,n-1}[\emptyset]$ enriched with objects $X$. It is shown in [13] that, for $n/2 \leq k \leq n-1$, $k$-set agreement can be solved in $\mathcal{AMP}_{n,n-1}[2(n-k)$-register] while a $2(n-k)$-register cannot be built in $\mathcal{AMP}_{n,n-1}[k$-set agreement].

The failure detector class $\Sigma_k$ This failure detector class, that generalizes $\Sigma$, has been introduced by Bonnet and Raynal [5] as an effort to capture the shared memory properties necessary to solve $k$-set agreement in message passing systems. As for $\Sigma$, each process is provided with a local read-only variable $qr_i$. It is assumed that after a process $p_i$ has crashed (if ever it does), $qr_i$ remains forever equal to $\{1, \ldots, n\}$. These variables satisfy the following properties.

- Intersection. Let $id_1, \ldots, id_{k+1}$ denote $k+1$ process ids, and $\tau_1, \ldots, \tau_{k+1}$ denote $k+1$ arbitrary time instants. $\forall id_1, \ldots, id_{k+1}, \tau_1, \ldots, \tau_{k+1} : \exists i, j : 1 \leq i \neq j \leq k+1 : (qr_{id_i}^{\tau_i} \cap qr_{id_j}^{\tau_j} \neq \emptyset)$.

- Liveness. $\exists r : \forall r' \geq r : \forall i \in C : qr_i^{r'} \subseteq C$.

The liveness property is the same as for $\Sigma$ while the intersection property generalizes the one of $\Sigma$ (we have $\Sigma_1 = \Sigma$). That property states that any set of $k+1$ quorums is such that any two of its quorums intersect whatever the time instants at which the values of these quorums have been obtained. (It is interesting to notice that this intersection property is the same as the one used to define $k$-coteries [28].) The main property of $\Sigma_k$ is the following theorem.

**Theorem 6.** [Bonnet-Raynal 2009] $\Sigma_k$ is a necessary requirement when one wants to wait-free solve $k$-set agreement (with a failure detector) in $\mathcal{AMP}_{n,n-1}[\emptyset]$. (Proof in [5].)

It is also shown in [5] that $\Sigma_{n-1}$ and $\mathcal{L}_{n-1} = \mathcal{L}$ are equivalent. This means that $\mathcal{L}$ can be built in $\mathcal{AMP}_{n,n-1}[\Sigma_{n-1}]$ and $\Sigma_{n-1}$ can be built in $\mathcal{AMP}_{n,n-1}[\mathcal{L}]$. The algorithm that builds $\mathcal{L}$ in $\mathcal{AMP}_{n,n-1}[\Sigma_{n-1}]$ is pretty trivial. At each process $p_i$, the boolean local variable $alone_i$ is initialized to $false$ and is set forever to $true$ when the quorum $qr_i$ becomes equals to $\{i\}$.

The algorithm that builds $\Sigma_{n-1}$ in $\mathcal{AMP}_{n,n-1}[\mathcal{L}]$ is described in Figure 4. At each process $p_i$, $qr_i$ is initialized to $\{i, j\}$ where $j \neq i$. Then, $p_i$ periodically broadcasts an $\text{alive}(i)$ message to indicate that it has not (yet) crashed. When $p_i$’s boolean local variable $alone$, becomes true, $qr_i$ is set to $\{i\}$ and keeps that value forever. When it receives a message $\text{alive}(j)$, $p_i$ resets $qr_i$ to $\{i, j\}$ if $qr_i \neq \{i\}$. A proof of correctness of this construction is given in [5].

Consequently, $\Sigma_{n-1}$ provides us with a quorum-based formulation of the weakest failure detector to solve $(n-1)$-set agreement. In contrast, while $\Sigma_1$ can be built in $\mathcal{AMP}_{n,n-1}[\mathcal{L}]$, the converse is not true.
The class of failure detectors $\Pi_k$ This class of failure detectors has been introduced by Bonnet and Raynal in in [5]. It is $\Sigma_k$ with the additional property.

- Eventual leadership. $\forall \tau : \exists LD = \{\ell_1, \ldots, \ell_k\} : \forall \tau' \geq \tau : \forall i \in C : qr^\tau_i \cap LD \neq \emptyset$.

It is shown in [5] that $\Pi_{n-1} = L_{n-1}$ and $\Pi_1 = (\Sigma, \Omega)$. Hence $\Pi_k$ captures in a single formulation the weakest failure detector for $k$-set agreement for $k = 1$ and $k = n - 1$. It seems that (unfortunately) $\Pi_k$ is not the weakest class of failure detectors for other values of $k$.

It is also shown in [5] that the class $\Pi_k$ and the class $(\Sigma_k, \Omega_k)$ are equivalent (any failure detector of one class can be used to build a failure detector of the other class). The failure detector class $\Omega_k$ has been introduced by Neiger [32]. This class (that has inspired the definition of $\Omega_k$) provides each process $p_i$ with a set $leaders_i$ such that the following properties are satisfied.

- Validity. $\forall i : \forall \tau : leaders_i^\tau$ is a set of $k$ process identities.

- Strong Eventual leadership. $\exists \tau : \exists LD = \{\ell_1, \ldots, \ell_k\} : (LD \cap C \neq \emptyset) \land (\forall \tau' \geq \tau : \forall i \in C : leaders_i^\tau = LD)$.

It is easy to see that $\Omega_k$ is strictly stronger than $\Omega_k$: any failure detector of the class $\Omega_k$ belongs to the class $\Omega_k$ while the opposite is not true.

4.6 What can be done with $\Sigma_x$

A $\Sigma_x$-based $k$-set agreement algorithm In [8], Bouzid and Travers present an interesting $k$-set algorithm for the system model $\mathcal{AMP}_{n,n-1}[\Sigma_x]$. This algorithm (that combines ideas from [13] and [16]) is described in Figure 5. A process invokes $\text{set\_agreement}_i(v_i)$ where $v_i$ is the value it proposes.
The processes are statically partitioned into \( x + 1 \) partitions, \( A_1, \ldots, A_{x+1} \) (i.e., \( \forall y \neq z : A_y \cap A_z = \emptyset \) and \( \bigcup_{1 \leq y \leq x} A_y = \{1, \ldots, n\} \)). Moreover their sizes are such that \( \forall y \in [1..x] : |A_y| = \lfloor \frac{n}{x+1} \rfloor \) and \( |A_{x+1}| = \lfloor \frac{n}{x+1} \rfloor + (n \mod (x+1)) \).

\[
\text{when operation set\_agreement}_i(v_i) \text{ is invoked by } p_i:\n\begin{align*}
(01) & \text{ for each } j \in A_{i+1} \cup \cdots \cup A_{x+1} \text{ do send est}(v_i) \text{ to } p_j \text{ end for}; \\
(02) & \text{ repeat aux } \leftarrow qr \text{ until } aux \subseteq A_i \text{ end repeat}; \\
(03) & \text{ for each } j \neq i \text{ do send dec}(v_i) \text{ to } p_j \text{ end for}; \\
(04) & \text{return}(v_i). 
\end{align*}
\]

\[
\text{when (est}(v) \text{ or dec}(v) \text{ is received) :}
\begin{align*}
(05) & \text{ for each } j \neq i \text{ do send dec}(v) \text{ to } p_j \text{ end for}; \\
(06) & \text{return}(v). 
\end{align*}
\]

Figure 5: \( k \)-Set agreement in in \( \mathcal{AMP}_{n,x-1}[\Sigma_x] \) (code for \( p_i \in A_y \)) [8]

Let \( p_i \) be a process belonging to partition \( A_y \). The idea is for \( p_i \) to decide the proposal of some process that belongs to a partition \( A_z \) such \( z < y \) (i.e., belonging to a partition “lower” than the one of \( p_i \)). To that end, each process \( p_i \) of \( A_y \) sends its proposal \( v_i \) to all processes of “higher” partitions (line 01). Let us notice that every process, other than the processes of the “highest” partition \( A_{x+1} \) and the processes that have initially crashed, sends an est() message.

When a process \( p_i \) receives a message est() (such a message is necessarily from a process belonging to a “lower” partition) it decides the value \( v \) (line 06). Moreover, just before deciding its value, it informs the other processes that it is about to decide \( v \) (message est() sent at line 05). Process \( p_i \) does the same processing when it receives a message dec() (let us notice that such a message can come from any other process). Let us observe that, as no process in \( A_{x+1} \) ever sends a message est(), at most \( n - |A_{x+1}| = \lfloor \frac{n}{x+1} \rfloor \) values can be decided from the reception of est() messages.

Unfortunately the previous mechanism is not sufficient to prevent processes from blocking forever. Such a blocking can occur when all correct processes belong to the very same partition. Line 02 is used to prevent such a definitive blocking. As after some finite time \( qr \) contains only correct processes, we eventually have \( qr \subseteq A_y \) if all correct processes belongs to \( A_y \). Hence, if \( qr \subseteq A_y \), \( p_i \) is allowed to decide its own proposal (line 04) after having informed the other processes with a dec() message (line 03). The proof (see [8]) shows that, due to the quorum intersection property of \( \Sigma_x \), at most \( n \mod (x+1) \) additional values can be decided, from which follows that no more than \( \lfloor \frac{n}{x+1} \rfloor + (n \mod (x+1)) \)
The Bulletin of the EATCS

are decided.

**Theorem 7.** [Bouzid-Travers 2010] The algorithm described in Figure 5 wait-free solves the \( k \)-set agreement problem in the system model \( \text{AMP}_{n, n-1}[\Sigma_i] \) for \( k \geq n - \left\lceil \frac{n}{x+1} \right\rceil \). Moreover, there is no wait-free \( k \)-set agreement algorithm in \( \text{AMP}_{n, n-1}[\Sigma_i] \) when the triple \((n, x, k)\) is such that \( k < n - \left\lceil \frac{n}{x+1} \right\rceil \). (Proof in [8].)

**Remark** As we have seen, \( \Sigma_1 \) is the weakest failure detector \( \text{AMP}_{n, n-1}[\emptyset] \) has to be enriched with in order to wait-free build a shared register. The previous theorem shows that \( \left\lceil \frac{n}{2} \right\rceil \)-set agreement can be wait-free solved in \( \text{AMP}_{n, n-1}[\Sigma_1] \). Hence, while neither a register nor \( \left\lceil \frac{n}{2} \right\rceil \)-set agreement can be built in \( \text{AMP}_{n, n-1}[\emptyset] \), both can be solved in \( \text{AMP}_{n, n-1}[\Sigma_1] \).

In contrast, when we consider the asynchronous shared memory system model \( \text{ASM}_{n, n-1}[\emptyset] \), shared registers are given for free while the weakest failure detector-based model in which \( \left\lceil \frac{n}{2} \right\rceil \)-set agreement can be wait-free solved is \( \text{ASM}_{n, n-1}[\Omega \left\lceil \frac{n}{2} \right\rceil] \).

**Using both \( \Sigma_i \) and \( \Omega \)** In [8] a \( k \)-set agreement algorithm for the system model \( \text{AMP}_{n, n-1}[\Sigma_i, \Omega_i] \) is presented. This algorithm works for \( k \geq x \times z \). Moreover, it is also shown in this paper that there is no \( k \)-set agreement algorithm in \( \text{AMP}_{n, n-1}[\Sigma_i, \Omega_i] \) when \( k < x \times z \) and \( n \geq x \times 2z \).

5 Conclusion

This paper focused on the \( k \)-set agreement problem in asynchronous systems prone to process crash failures. In such a context, it has considered two different communication models: the read/write shared memory model and the message passing model.

As \( k \)-set agreement cannot be solved in these models, the paper has presented recent results when the failure detector-based approach is used to circumvent the previous impossibility. As we have seen, while the weakest failure detector (i.e., the one that provides processes with “as few information on failures as possible”) is known when communication is by read/write shared memory, this is not the case when communication is by message passing. The paper has presented the most recent results in that direction. It is hoped that this paper not only will help readers to better understand the problem and its difficulty, but will also help them in the quest for the discovery of the weakest failure detector for the message passing case.
Acknowledgments

I want to thank Panagiota Fatourou for comments that helped improve the presentation of the paper. I also thank all the colleagues who have proposed the failure detector classes presented in the paper, designed the algorithms based on these failure detectors and proved the corresponding lower bounds. Without them, this short survey would not exist!

References


The Bulletin of the EATCS


The Bulletin of the EATCS

