Consensus is Easier Than Reliable Broadcast

Carole Delporte-Gallet, Stéphane Devismes, Hugues Fauconnier, Franck Petit, Sam Toueg

With Finite Memory
Consensus is Easier Than Reliable Broadcast

Carole Delporte-Gallet[1] Stéphane Devismes[2] Hugues Fauconnier[1]
Franck Petit[3*] Sam Toueg[4]

Abstract

We consider asynchronous distributed systems with message losses and process crashes. We study the impact of finite process memory on the solution to *consensus*, *repeated consensus* and *reliable broadcast*. With finite process memory, we show that in some sense consensus is easier to solve than reliable broadcast, and that reliable broadcast is as difficult to solve as repeated consensus: More precisely, with finite memory, consensus can be solved with failure detector $\mathcal{S}$, and $\mathcal{P}^-$ (a variant of the perfect failure detector which is stronger than $\mathcal{S}$) is necessary and sufficient to solve reliable broadcast and repeated consensus.

Keywords: Distributed algorithms, failure detectors, reliable broadcast, consensus, repeated consensus.

1 Introduction

Designing fault-tolerant protocols for asynchronous systems is highly desirable but also highly complex. Some classical agreement problems such as *consensus* and *reliable broadcast* are well-known tools for solving more sophisticated tasks in faulty environments (e.g., [19, 16]). Roughly speaking, with consensus processes must reach a common decision on their inputs, and with reliable broadcast processes must deliver the same set of messages.

It is well known that consensus cannot be solved in asynchronous systems with failures [15], and several mechanisms were introduced to circumvent this impossibility result: *randomization* [7], *partial synchrony* [11, 12] and *(unreliable) failure detectors* [6].

Informally, a failure detector is a distributed oracle that gives (possibly incorrect) hints about the process crashes. Each process can access a local failure detector module that monitors the processes of the system and maintains a list of processes that are suspected of having crashed.

Several classes of failure detectors have been introduced, *e.g.*, $\mathcal{P}$, $\mathcal{S}$, Ω, etc. Failure detectors classes can be compared by reduction algorithms, so for any given problem P, a natural question is "*What is the weakest failure detector (class) that can solve P ?*". This question has been extensively studied for several problems in systems *with infinite process memory* (*e.g.*, uniform and non-uniform versions of consensus [5, 13], non-blocking atomic commit [9], uniform reliable broadcast [1, 21],

[1]LIAFA, Université D. Diderot, Paris, France, {cd,hf}@liafa.jussieu.fr
[2]VERIMAG, Université Joseph Fourier, Grenoble, France, stephane.devismes@imag.fr
[3]INRIA/LIP Laboratory,Univ. of Lyon/ENS Lyon, Lyon, France, franck.petit@ens-lyon.fr
[4]Department of Computer Science, University of Toronto, Toronto, Canada, sam@cs.toronto.edu

*This work was initiated while Franck Petit was with MIS Lab., Université of Picardie, France. Research partially supported by Région Picardie, Proj. APREDY.

implementing an atomic register in a message-passing system [9], mutual exclusion [10], boosting obstruction-freedom [18], set consensus [23, 25], etc.). This question, however, has not been as extensively studied in the context of systems *with finite process memory.*

In this paper, we consider systems where processes have finite memory, processes can crash and links can lose messages (more precisely, links are fair lossy and FIFO[1]). Such environments can be found in many systems, for example in sensor networks, sensors are typically equipped with small memories, they can crash when their batteries run out, and they can experience message losses if they use wireless communication.

In such systems, we consider (the uniform versions of) reliable broadcast, consensus and repeated consensus. Our contribution is threefold: First, we establish that the weakest failure detector for reliable broadcast is $\mathcal{P}^-$ — a failure detector that is almost as powerful than the perfect failure detector $\mathcal{P}$.[2] Next, we show that consensus can be solved using failure detector $\mathcal{S}$. Finally, we prove that $\mathcal{P}^-$ is the weakest failure detector for repeated consensus. Since $\mathcal{S}$ is strictly weaker than $\mathcal{P}^-$, in some precise sense these results imply that, in the systems that we consider here, consensus is easier to solve than reliable broadcast, and reliable broadcast is as difficult to solve as repeated consensus.

The above results are somewhat surprising because, when processes have infinite memory, reliable broadcast is easier to solve than consensus[3], and repeated consensus is not more difficult to solve than consensus.

Roadmap. The rest of the paper is organized as follows: In the next section, we present the model considered in this paper. In Section 4, we show that in case of process memory limitation and possibility of crashes, $\mathcal{P}^-$ is necessary and sufficient to solve reliable broadcast. In Section 5, we show that consensus can be solved using a failire detector of type $\mathcal{S}$ in our systems. In Section 6, we show that $\mathcal{P}^-$ is necessary and sufficient to solve repeated consensus in this context.

For space considerations, all the proofs are relegated to an optional appendix.

2 Model

Distributed System. A system consists of a set $\Pi = \{p_1, ..., p_n\}$ of processes. We consider *asynchronous* distributed systems where each process can communicate with each other through *directed links.*[4] By asynchronous, we mean that there is no bound on message delay, clock drift, or process execution rate.

A process has a local memory, a local sequential and deterministic algorithm, and input/output capabilities. In this paper we consider systems of processes having either a *finite* or an *infinite* memory. In the sequel, we denote such systems by $\Phi^{\mathcal{F}}$ and $\Phi^{\mathcal{I}}$, respectively.

We consider links with unbounded capacities. We assume that the messages sent from p to q are *distinguishable*, *i.e.*, if necessary, the messages can be numbered with a non-negative integer. These numbers are used for notational purpose only, and are unknown to the processes. Every link satisfies the *integrity*, *i.e.*, if a message m from p is received by q, m is received by q at most once, and only if p previously sent m to q. Links are also *unreliable* and *fair.* Unreliable means that the

[1]The FIFO assumption is necessary because, from the results in [22], if lossy links are not FIFO, reliable broadcast requires unbounded message headers.

[2]Note that $\mathcal{P} \subseteq \mathcal{P}^-$ and $\mathcal{P}^-$ is *unrealistic* according to the definition in [8].

[3]With infinite memory and fair lossy links, (uniform) reliable broadcast can be solved using Θ [3], and Θ is strictly weaker than (Σ, Ω) which is necessary to solve consensus.

[4]We assume that each process knows the set of processes that are in the system; some papers related to failure detectors do not make this assumption e.g. [4, 17, 14].

messages can be lost. Fairness means that for each message m, if process p sends infinitely often m to process q and if q tries to receive infinitely often a message from p, then q receives infinitely often m from p. Each link are *FIFO*, *i.e.*, the messages are received in the same order as they were sent.

To simplify the presentation, we assume the existence of a discrete global clock. This is merely a fictional device: the processes do not have access to it. We take the range $\mathcal{T}$ of the clock's ticks to be the set of natural numbers.

Failures and Failure Patterns. Every process can fail by permanently *crashing*, in which case it definitively stops to execute its local algorithm. A *failure pattern* F is a function from $\mathcal{T}$ to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed through time t. Once crashed, a process never recoves, *i.e.*, $\forall t : F(t) \subseteq F(t+1)$. We define $crashed(F) = \bigcup_{t \in \mathcal{T}} F(t)$ and $correct(F) = \Pi \setminus crashed(F)$. If $p \in crashed(F)$ we say that p *crashes in* F (or simply *crashed* when it is clear in the context) and if $p \in correct(F)$ we say that p is *correct in* F (or simply *correct* when it is clear in the context). An environment is a set of failure patterns. We do not restrict here the number of crash and we consider as environment $\mathcal{E}$ the set of all failure patterns.

Failure Detectors. A failure detector [6] is a local module that outputs a set of processes that are currently suspected of having crashed. A *failure detector history* H is a function from $\Pi \times \mathcal{T}$ to 2^{Π}. $H(p,t)$ is the value of the failure detector module of process p at time t. If $q \in H(p,t)$, we say that p *suspects* q *at time* t *in* H. We omit references to H when it is obvious from the context.

Formally, *failure detector* $\mathcal{D}$ is a function that maps each failure pattern F to a set of failure detector histories $\mathcal{D}(F)$.

A failure detector can be defined in terms of two *abstract properties*: *Completeness* and *Accuracy* [6] . Let us recall one type of *completeness* and two types of *accuracy.*

Definition 1 (Strong Completeness) *Eventually every process that crashes is permanently suspected by* every correct process. *Formally,* $\mathcal{D}$ *satisfies strong completeness if:* $\forall F \in \mathcal{E}, \forall H \in \mathcal{D}(F), \exists t \in \mathcal{T}, \forall p \in crashed(F), \forall q \in correct(F), \forall t' \geq t : p \in H(q,t')$

Definition 2 (Strong Accuracy) *No process is suspected before it crashes. Formally,* $\mathcal{D}$ *satisfies strong accuracy if:* $\forall F \in \mathcal{E}, \forall H \in \mathcal{D}(F), \forall t \in \mathcal{T}, \forall p, q \in \Pi \setminus F(t) : p \notin H(q,t)$

Definition 3 (Weak Accuracy) *A correct process is never suspected. Formally,* $\mathcal{D}$ *satisfies weak accuracy if:* $\forall F \in \mathcal{E}, \forall H \in \mathcal{D}(F), \forall t \in \mathcal{T}, \exists p \in correct(F), \forall q \in \Pi : p \notin H(q,t)$

We introduce a last type of accuracy:

Definition 4 (Almost Strong Accuracy) *No correct processes are suspected. Formally,* $\mathcal{D}$ *satisfies almost strong accuracy if:* $\forall F \in \mathcal{E}, \forall H \in \mathcal{D}(F), \forall t \in \mathcal{T}, \forall p \in correct(F), \forall q \in \Pi : p \notin H(q,t)$

For all these aformentioned properties, we can assume, without loss of generality, that when a process is suspected it remains suspected forever.

We now recall the definition of the perfect and the strong failure detectors [6] and we introduce our almost perfect failure detector:

Definition 5 (Perfect) *A failure detector is said to be* perfect *if it satisfies the strong completeness and the strong accuracy properties. This failure detector is denoted by* $\mathcal{P}$*.*

Definition 6 (Almost Perfect) *A failure detector is said to be* almost perfect *if it satisfies the strong completeness and the almost strong accuracy properties. This failure detector is denoted by* $\mathcal{P}^-$.

Definition 7 (Strong) *A failure detector is said to be* strong *if it satisfies the strong completeness and the weak accuracy properties. This failure detector is denoted by* $\mathcal{S}$.

Algorithms, Runs, and Specification. A distributed algorithm is a collection of n sequential and deterministic algorithms, one for each process in Π. Computations of distributed algorithm $\mathcal{A}$ proceed in *atomic steps*. In a step, a process p executes each of the following actions at most once: (1) p gets an input, (2) p outputs a value, (3) p receives a message, (4) p queries its failure detector module, (5) p modifies its (local) state, and (6) p sends a message to a single process.

A *run* of Algorithm $\mathcal{A}$ using a failure detector $\mathcal{D}$ is a tuple $R = \langle F, H_{\mathcal{D}}, \gamma_{init}, E, T\rangle$ where F is a failure pattern, $H_{\mathcal{D}} \in \mathcal{D}(F)$ is an history of failure detector $\mathcal{D}$ for the failure pattern F, γ_{init} is an *initial configuration* of $\mathcal{A}$, E is an infinite sequence of steps of A, and T is a list of increasing time values indicating when each step in E occurred. A run must satisfy certain well-formedness and fairness properties. In particular, (1) E is applicable to γ_{init}, (2) a process cannot take steps after it crashes, (3) when a process takes a step and queries its failure detector module, it gets the current value output by its local failure detector module, (4) every process that is correct in F takes an infinite number of local steps in E, and (5) any message sent is eventually received or lost.

A *problem* P is defined by a set of properties that runs must satisfy. An algorithm A *solves a problem* P using a failure detector $\mathcal{D}$ if and only if all the runs of A using $\mathcal{D}$ satisfy the properties required by P.

A failure detector $\mathcal{D}$ is said to be *weaker* than another failure detector $\mathcal{D}'$ (denote $\mathcal{D} \leq \mathcal{D}'$) if there is an algorithm that uses only $\mathcal{D}'$ to emulate the output of $\mathcal{D}$ for every failure pattern. If $\mathcal{D}$ is weaker than $\mathcal{D}'$ but $\mathcal{D}'$ is not weaker than $\mathcal{D}$ we say that $\mathcal{D}$ is *strictly weaker* than $\mathcal{D}'$ (denote $\mathcal{D} < \mathcal{D}'$).

From [6] and our definition of $\mathcal{P}^-$, we get:

Proposition 1

$$\mathcal{S} < \mathcal{P}^- < \mathcal{P}$$

The *weakest* [5] failure detector $\mathcal{D}$ to solve a given problem is a failure detector $\mathcal{D}$ that is sufficient to solve the problem and that is also necessary to solve the problem, i.e. $\mathcal{D}$ is weaker than any failure detector that solves the problem.

Notations. In the sequel, v_p denotes the value of the variable v at process p. Also, v_p^τ denotes the value of v_p at time τ. Finally, a datum in a message can be replaced by "–" when this value has no impact on the reasonning.

3 Problem specifications

Reliable Broadcast. The reliable broadcast [20] is defined with two primitives: `BROADCAST`(m) and `DELIVER`(m). Informally, after a process p invokes `BROADCAST`(m), every correct processes must eventually execute `DELIVER`(m). In the formal definition below, we denote by *sender*(m) the process that invokes `BROADCAST`(m).

Specification 1 (Reliable-Broadcast) *A run R satisfies* `reliable-broadcast` *if and only if the following three requirements are satisfied in R:*

Validity: *If a correct process invokes* `BROADCAST`(m)*, then it eventually executes* `DELIVER`(m)*.*

(Uniform) Agreement: *If a process executes* `DELIVER`(m)*, then all other correct processes eventually execute* `DELIVER`(m)*.*

Integrity: *For every message* m*, every process executes* `DELIVER`(m) *at most once, and only if sender*(m) *previously invokes* `BROADCAST`(m)*.*

Consensus. In the consensus problem, all correct processes *propose* a value and must reach a unanimous and irrevocable *decision* on some value that is chosen between the proposed values. We define the consensus problem in terms of two primitives, `PROPOSE`(v) and `DECIDE`(u). When a process executes `PROPOSE`(v), we say that it *proposes* v; similarly, when a process executes `DECIDE`(u), we say that it *decides* u.

Specification 2 (Consensus) *A run R satisfies* `consensus` *if and only if the following three requirements are satisfied in R:*

(Uniform) Agreement: *No two processes* decide *differently.*

Termination: *Every correct process eventually* decides *some value.*

Validity: *If a process* decides v*, then* v *was* proposed *by some process.*

Repeated Consensus. We now define repeated consensus. Each correct process has as input an infinite sequence of proposed values, and outputs an infinite sequence of decision values such that (1) two correct processes have the same output (the output of a faulty process is a prefix of this output) and (2) the i^{th} value of the output is the i^{th} value of the input of some process. We define the repeated consensus problem in terms of two primitives, `R-PROPOSE`(v) and `R-DECIDE`(u). When a process executes the i^{th} `R-PROPOSE`(v), v is the i^{th} value of its input (we say that it *proposes* v for the i^{th} consensus); similarly, when a process executes the i^{th} `R-DECIDE`(u) u is the i^{th} value of its output (we say that it *decides* v for the i^{th} consensus).

Specification 3 (Repeated Consensus) *A run R satisfies* `repeated-consensus` *if and only if the following three requirements are satisfied in R:*

Agreement: *If* u *and* v *are the outputs of two processes, then either* u *is a prefix*[5] *or* v *is a prefix of* u*.*

Termination: *Every correct process has an infinite output.*

Validity: *If the* i^{th} *value of the output of a process is* v*, then* v *is the* i^{th} *value of the input of some process.*

4 Reliable Broadcast in $\Phi^{\mathcal{F}}$

We show in this section that $\mathcal{P}^-$ is the weakest failure detector to solve the reliable broadcast in $\Phi^{\mathcal{F}}$.

[5]If $u = v$, then u is also a prefix of v.

$\mathcal{P}^-$ **is Necessary.** To show that $\mathcal{P}^-$ is necessary to solve the reliable broadcast the following lemma is central to the proof:

Lemma 1 *Let $\mathcal{A}$ be an algorithm solving* `reliable-broadcast` *in $\Phi^{\mathcal{F}}$ with a failure detector $\mathcal{D}$. There exists an integer k such that for every process p and every correct process q, for every run R of $\mathcal{A}$ where process p* `BROADCAST`*s and* `DELIVER`*s k messages, at least one message from q has been received by some process.*

Assume now that there exists an algorithm $\mathcal{A}$ that implements `reliable-broadcast` in $\Phi^{\mathcal{F}}$ using the failure detector $\mathcal{D}$. To show our result we have to give an algorithm that uses only $\mathcal{D}$ to emulate the output of $\mathcal{P}^-$ for every failure pattern.

Actually, we give an algorithm $\mathcal{A}_{(p,q)}$ (Figure 1) where a given process p monitors a given process q. This algorithm uses one instance of $\mathcal{A}$ with $\mathcal{D}$. Note that all processes except q participate to this algorithm following the code of $\mathcal{A}$. In this algorithm *Output_q* is equal to either $\{q\}$ (q is faulty) or $\emptyset$ (q is correct).

The algorithm $\mathcal{A}_{(p,q)}$ works as follows: p tries to `BROADCAST` k messages, all processes execute the code of the algorithm $\mathcal{A}$ using $\mathcal{D}$ except q that does nothing. If p `DELIVER`s k messages, it sets *Output_q* to q and never changes the values of *Output_q*. By lemma 1, if q is correct p can't deliver k messages and so it never sets *Output_q* to q. If q is faulty and p is correct: as $\mathcal{A}$ solve `reliable-broadcast`, p has to deliver `DELIVER` k messages and so p *Output_q* to q.[6]

To emulate $\mathcal{P}$, each process p uses algorithm $\mathcal{A}_{(p,q)}$ for every process q. As $\mathcal{D}$ is a failure detector it can be used for each instance. The output of $\mathcal{P}^-$ at p (Variable *Output*) is then the union of *Output_q* for every process q.

```
1:      /* CODE FOR PROCESS $p$ */
2: begin
3:     $Output\_q \leftarrow \emptyset$
4:     for $i = 1$ to $k$ do
5:         BROADCAST($m$)      /* using $\mathcal{A}$ with $\mathcal{D}$ */
6:         wait for DELIVER($m$)
7:     end for
8:     $Output\_q \leftarrow \{q\}$
9: end

10:      /* CODE FOR PROCESS $q$ */
11: begin
12: end

13:      /* CODE FOR EVERY PROCESS $\Pi - \{p, q\}$ */
14: begin
15:     execute the code of $\mathcal{A}$ with $\mathcal{D}$ for these messages
16: end
```

Figure 1: $\mathcal{A}_{(p,q)}$

Theorem 1 *$\mathcal{P}^-$ is necessary to solve* `reliable-broadcast` *in $\Phi^{\mathcal{F}}$.*

$\mathcal{P}^-$ **is Sufficient.** In Algorithm $\mathcal{B}$ (Figure 2), every process uses a failure detector module of type $\mathcal{P}^-$ and a finite memory. Theorem 2 shows that Algorithm $\mathcal{B}$ solves the reliable broadcast in $\Phi^{\mathcal{F}}$ and directly implies that $\mathcal{P}^-$ is sufficient to solve the reliable broadcast in $\Phi^{\mathcal{F}}$ (Corollary 1).

In Algorithm $\mathcal{B}$, each process p executes broadcasts sequentially: p starts a new broadcast only after the termination of the previous one. To that goal, any process p initializes `Mes`$[p]$ to $\perp$. Then, p periodically checks if an external application invokes `BROADCAST`$(-)$. In this case, `MesToBrd`$()$ returns the message to broadcast, say m. When this event occurs, `Mes`$[p]$ is set to m and the broadcast procedure starts. `Mes`$[p]$ is set to $\perp$ at the end of the broadcast, p checks again, and so on.

Algorithm $\mathcal{B}$ has to deal with two types of faults: process crashes and message loss.

[6] If q is faulty and p is faulty, the property of failure detector is trivially ensured.

```
    /* CODE FOR EVERY PROCESS $q$ */
variables:
    $\texttt{Flag}[1\dots n][1\dots n] \in \{0,1\}^{n^2}$;
        $\forall (i,j) \in \Pi^2$, $\texttt{Flag}[i][j]$ is initialized to 0
    FD: failure detector of type $\mathcal{P}^-$
    $\texttt{Mes}[1\dots n]$: array of data messages;
        $\forall i \in \Pi$, $\texttt{Mes}[i]$ is initialized to $\bot$
function: MesToBrd(): returns a message or $\bot$
begin
    repeat forever
        if $\texttt{Mes}[p] = \bot$ then
            $\texttt{Mes}[p] \leftarrow \texttt{MesToBrd}()$
            if $\texttt{Mes}[p] \neq \bot$ then
                $\texttt{Flag}[p][p] \leftarrow (\texttt{Flag}[p][p] + 1) \bmod 2$
            end if
        end if
        for all $i \in \Pi \setminus \texttt{FD}$ do
            for all $j \in \Pi \setminus (\texttt{FD} \cup \{p,i\}), \texttt{Flag}[i][p] \neq \texttt{Flag}[i][j]$ do
                if (rcv$\langle i\text{-ACK}, F\rangle$ from $j$)
                    $\wedge$ ($F = \texttt{Flag}[i][p]$) then
                    $\texttt{Flag}[i][j] \leftarrow F$
                else
                    send$\langle i\text{-BRD}, \texttt{Mes}[i], \texttt{Flag}[i][p]\rangle$ to $j$
                end if
            end for
            if ($\texttt{Mes}[i] \neq \bot$) $\wedge$
                ($\forall q \in \Pi \setminus \texttt{FD}$, $\texttt{Flag}[i][i] = \texttt{Flag}[i][q]$) then
                $\texttt{DELIVER}(\texttt{Mes}[i])$
                $\texttt{Mes}[i] \leftarrow \bot$
            end if
        end for
        for all $i \in \Pi \setminus \texttt{FD} \setminus \{p\}$ do
            for all $j \in \Pi \setminus (\texttt{FD} \cup \{p\})$ do
                if (rcv$\langle i\text{-BRD}, m, F\rangle$ from $j$) then
                    if ($\forall q \in \Pi \setminus \texttt{FD}$, $\texttt{Flag}[i][q] = \texttt{Flag}[i][i]$)
                        $\wedge$ ($F \neq \texttt{Flag}[i][p]$) then
                        $\texttt{Mes}[i] \leftarrow m$
                        $\texttt{Flag}[i][p] \leftarrow F$
                    end if
                    if $i = j$ then
                        $\texttt{Flag}[i][i] \leftarrow F$
                    end if
                    if ($i \neq j$) $\vee$
                    ($\forall q \in \Pi \setminus \texttt{FD}$, $\texttt{Flag}[i][q] = \texttt{Flag}[i][i]$) then
                        send$\langle i\text{-ACK}, \texttt{Flag}[i][p]\rangle$ to $j$
                    end if
                end if
            end for
        end for
    end repeat
end
```

Figure 2: Algorithm $\mathcal{B}$.

- *Dealing with process crashes.* Every process uses a failure detector of type $\mathcal{P}^-$ to detect the process crashes. Note that, as mentionned in Section 2, we assume that when a process is suspected it remains suspected forever.

  Assume that a process p broadcasts the message m: p sends a broadcast message (p-BRD) with the datum m to any process it believes to be correct.

  In Algorithm $\mathcal{B}$, p executes DELIVER(m) only after all other processes it does not suspect receives m. To that goal, we use acknowledgment mechanisms. When p received an acknowledgment for m (p-ACK) from every other process it does not suspect, p executes DELIVER(m) and the broadcast of m terminates ($\texttt{Mes}[p] \leftarrow \bot$).

  To ensure the *agreement* property, we must guarantee that if p crashes but another process q already executes DELIVER(m), then any correct process eventually executes DELIVER(m). To that goal, any process can execute DELIVER(m) only after all other processes it does not suspect except p receive m. Once again, we use acknowledgment mechanisms to that end: q also broadcasts m to every other process it does not suspect except p (this induces that a process can now receive m from a process different of p) until receiving an acknowledgment for m from all these processes and the broadcast message from p if q does not suspect it.

  To prevent m to be still broadcasted when p broadcasts the next message, we synchronize the system as follows: any process acknowledges m to p only after it received an acknowledgment for m from every other process it does not suspect except p. By contrast, if a process i receives a message broadcasted by p (p-BRD) from the process $j \neq p$, i directly acknowledges the message to j.

- *Dealing with message loss.* The broadcast messages have to be periodically retransmitted until they are acknowledged. To that goal, any process q stores the last broadcasted message from p into its variable $\texttt{Mes}_q[p]$ (initialized to $\perp$). However, some copies of previously received messages can be now in transit at any time in the network. So, each process must be able to distinguish if a message it receives is copy of a previously received message or a new one, say *valid*. To circumvent this problem, we use the traditional alternating-bit mechanism [2, 24]: a flag value (0 or 1) is stored into any message and a two-dimentionnal array, noted $\texttt{Flag}[1 \ldots n][1 \ldots n]$, allows us to distinguish if the messages are *valid* or not. Initially, any process sets $\texttt{Flag}[i][j]$ to 0 for all pairs of processes (i,j). In the code of Process p, the value $\texttt{Flag}_p[p][p]$ is used to mark every p-BRD messages sent by p. In the code of every process $q \neq p$, $\texttt{Flag}_q[p][q]$ is equal to the flag value of the last valid p-BRD message q receives (not necessarily from p). For all $q' \neq q$, $\texttt{Flag}_q[p][q']$ is equal to the flag value of the last valid p-BRD message q receives from q'.

Theorem 2 *Algorithm $\mathcal{B}$ is a reliable broadcast algorithm in $\Phi^{\mathcal{F}}$ with $\mathcal{P}^-$.*

Corollary 1 *$\mathcal{P}^-$ is sufficient for solving reliable broadcast in $\Phi^{\mathcal{F}}$.*

5 Consensus in $\Phi^{\mathcal{F}}$

We show in this section that we can solve consensus in system $\Phi^{\mathcal{F}}$ with a failure detector that is strictly weaker than the failure detector necessary to solve reliable broadcast and repeated consensus. We solve consensus with the strong failure detector $\mathcal{S}$. $\mathcal{S}$ is not the weakest failure detector to solve consensus whatever the number of crash but it is strictly weaker than $\mathcal{P}^-$ and so enough to show our results.

We customize the algorithm of Chandra and Toueg [6] that works in an asynchronous message-passing system with reliable links and augmented with a strong failure detector, to our model.

In this algorithm, $\mathcal{CS}$ (Figure 3), the processes execute n asynchronous rounds. First, processes execute $n-1$ asynchronous rounds (r denotes the current round number) during which they broadcast and relay their proposed values. Each process p waits until it receives a round r message from every process that is not suspected by its failure detector module in $\mathcal{S}$ (as mentionned in Section 2, we assume that when a process is suspected it remains suspected forever) before proceeding to round $r+1$. Then, processes execute a last asynchronous rounds during which they eliminate some proposed values. Again each process p waits until it receives a round n message from every process that is not suspected by its failure detector module. By the end of these n rounds, correct processes *agree* on a vector based on the proposed values of all processes. The i^{th} element of this vector either contains the proposed value of process i or $\perp$. This vector contains the proposed value of *at least one process* : the process that is never suspected by its failure detector. Correct processes decide the first nontrivial component of this vector.

To customize this algorithm to our model, we have to ensure the progress of each process: While a process has not ended the asynchronous round r it must be able to retransmit all the messages[7] that it has previously sent in order to allow others processes to progress despite the link failures. As we have used a strong failure detector and fair loss links, it is possible that one process has decided and the other ones still wait messages of asynchronous rounds. When a process has terminated the n asynchronous rounds, it uses a special `Decide` message to allow others processes to progress.

In the algorithm, we first use a function *consensus*. This function takes the proposed value in parameter and returns the decision value and uses a failure detector. Then, at processes request, we propagate the `Decide` message.

[7]Notice that they are in finite number

```
  /* Code for process $p$ */
function consensus($v$) with fd
    variables:
        Flag[$1 \dots n$] $\in \{true, false\}^n$;
                $\forall i \in \Pi$, Flag[$i$] is initialized to false
        V[$1 \dots n$]: array of propositions;
                $\forall i \in \Pi$, V[$i$] is initialized to $\perp$
        Mes[$1 \dots n$]: array of arrays of propositions;
                $\forall i \in \Pi$, Mes[$i$] is initialized to $\perp$
        $r$: integer; $r$ is initialized to 1
    begin
        V[$p$] $\leftarrow v$ the proposed values
        Mes[1] $\leftarrow$ V
        while ($r \leq n$) do
            send⟨R-$r$, Mes[$r$]⟩ to every process,
                    except $p$
            for all $i \in \Pi \setminus$ (fd $\cup \{p\}$), Flag[$i$] $= false$ do
                if (rcv⟨R-$r$, $W$⟩ from $i$) then
                    Flag[$i$] $\leftarrow true$
                    if $r < n$ then
                        if V[$i$] $= \perp$ then
                            V[$i$] $\leftarrow W[i]$; Mes[$r+1$][$i$] $\leftarrow W[i]$
                        end if
                    else
                        if V[$i$] $\neq \perp$ then
                            V[$i$] $\leftarrow W[i]$
                        end if
                    end if
                end if
            end for
            for all $i \in \Pi \setminus \{p\}$ do
                if (rcv⟨R-$x$, $W$⟩ from $i$), $x < r$ then
                    send⟨R-$x$, Mes[$x$]⟩ to $i$
                end if
            end for
            if $\forall q \in \Pi \setminus$ (fd $\cup \{p\}$), Flag[$q$] $= true$ then
                if $r < n$ then
                    for all $i \in \Pi$ do
                        Flag[$i$] $\leftarrow false$
                    end for
                end if
                if $r = n - 1$ then
                    Mes[$n$] $\leftarrow$ V
                end if
                $r \leftarrow r + 1$
            end if
            for all $i \in \Pi \setminus \{p\}$ do
                if (rcv⟨Decide, $d$⟩ from $i$) then
                    $return(d)$
                end if
            end for
        end while
        $d$ = the first component of V different from $\perp$
        $return(d)$
    end
end function
procedure PROPOSE($v$)
    variables:
        $u$: integer; FD: failure detector of type $\mathcal{S}$
    begin
        $u \leftarrow$consensus($v$) with FD
        DECIDE($u$)
        repeat forever
            for all $j \in \Pi \setminus \{p\}, x \in \{1, ..., n\}$ do
                if (rcv⟨R-$x$, $W$⟩ from $j$) then
                    send⟨Decide, $u$⟩ to $j$
                end if
            end for
        end repeat
    end
end procedure
```

Figure 3: Algorithm $\mathcal{CS}$, consensus with $\mathcal{S}$.

Theorem 3 shows that Algorithm $\mathcal{CS}$ solves the consensus in $\Phi^{\mathcal{F}}$ and directly implies that $\mathcal{S}$ is sufficient to solve te consensus problem in $\Phi^{\mathcal{F}}$ (Corollary 2).

Theorem 3 *Algorithm $\mathcal{CS}$ is a consensus algorithm in $\Phi^{\mathcal{F}}$ with $\mathcal{S}$.*

Corollary 2 *$\mathcal{S}$ is sufficient for solving consensus in $\Phi^{\mathcal{F}}$.*

6 Repeated consensus in $\Phi^{\mathcal{F}}$

We show in this section that $\mathcal{P}^-$ is the weakest failure detector to solve the reliable consensus problem in $\Phi^{\mathcal{F}}$.

$\mathcal{P}^-$ is Necessary. The proof is similar to the one in Section 4, and here the following lemma is central to the proof:

Lemma 2 *Let $\mathcal{A}$ be an algorithm solving* `repeated-consensus` *in $\Phi^{\mathcal{F}}$ with a failure detector $\mathcal{D}$. There exists an integer k such that for every process p and every correct process q for every run R of $\mathcal{A}$ where process p* `R-PROPOSE`*s and* `R-DECIDE`*s k times, at least one message from q has been received by some process.*

Assume that there exists an algorithm $\mathcal{A}$ that implements `repeated-consensus` in $\Phi^{\mathcal{F}}$ using the failure detector $\mathcal{D}$. To show our result we have to give an algorithm that uses only $\mathcal{D}$ to emulate the output of $\mathcal{P}^-$ for every failure pattern.

In fact we give an algorithm $\mathcal{A}_q$ (Figure 4) where processes monitor a given process q. This algorithm uses one instance of $\mathcal{A}$ with $\mathcal{D}$. Note that all processes except q participate to this algorithm following the code of $\mathcal{A}$. In this algorithm $Output\_q$ is equal to either $\{q\}$ (q is crashed) or $\emptyset$ (q is correct).

The algorithm $\mathcal{A}_q$ works as follows: processes try to `R-DECIDE` k times, all processes execute the code of the algorithm $\mathcal{A}$ using $\mathcal{D}$ except q that does nothing. If p `R-DECIDE` k messages, it sets $Output\_q$ to q and never changes the values of $Output\_q$.

By lemma 2, if q is correct p cannot decides k times and so it never sets $Output\_q$ to q. If q is faulty and p is correct[8]: as $\mathcal{A}$ solve `repeated-consensus`, p has to `R-DECIDE` k times and so p sets $Output\_q$ to q.

To emulate $\mathcal{P}^-$, each process p uses Algorithm $\mathcal{A}_q$ for every process q. As $\mathcal{D}$ is a failure detector it can be used for each instance. The output of $\mathcal{P}^-$ at p (Variable $Output$) is then the union of $Output\_q$ for every process q.

```
1:     /* CODE FOR PROCESS $p$ OF $\Pi \setminus q$ */
2: begin
3:     $Output\_q \leftarrow \emptyset$
4:     for $i = 1$ to $k$ do
5:         R-PROPOSED($v$)     /* using $\mathcal{A}$ with $\mathcal{D}$ */
6:         wait for R-DECIDE($v$)
7:     end for
8:     $Output\_q \leftarrow \{q\}$
9: end
```

```
10:     /* CODE FOR PROCESS $q$ */
11: begin
12: end
```

Figure 4: $\mathcal{A}_q$

Theorem 4 *$\mathcal{P}^-$ is necessary to solve* `repeated-consensus` *problem in $\Phi^{\mathcal{F}}$.*

$\mathcal{P}^-$ is Sufficient. In this section, we show that $\mathcal{P}^-$ is sufficient to solve the repeated consensus in $\Phi^{\mathcal{F}}$. To that goal, we consider an algorithm called Algorithm $\mathcal{RCP}$ (Figures 5 and 6). In this algorithm, any process uses a failure detector module of type $\mathcal{P}^-$ and a finite memory. Theorem 5 shows that Algorithm $\mathcal{RCP}$ solves the repeated consensus in $\Phi^{\mathcal{F}}$ and directly implies that $\mathcal{P}^-$ is sufficient to solve the repeated consensus in $\Phi^{\mathcal{F}}$ (Corollary 3).

We assume that each correct processes has an infinite sequence of input and when it terminates `R-PROPOSED`(v) where v is the i^{th} value of its input, it executes `R-PROPOSED`(w) where w is the $(i+1)^{th}$ value of its input.

When a process executes `R-PROPOSED`(v), it first executes a consensus in which it proposes v. The decision of this consensus is then outputted. Then, processes have to avoid that the messages of two consecutive consensus are mixed up. We construct a synchronization barrier. Without message loss, and with a perfect failure detector, it is sufficient that each process waits a `Decide` message from every process trusted by its failure detector module. By FIFO property, no message $\langle$`R-`$x, -\rangle$ sent before this `Decide` message can be received in the next consensus.

To deal with message loss, the synchronization barrier is obtained by two asynchronous rounds: In the first asynchronous rounds, each process sends a `Decide` message and waits to receive a

[8]If q is faulty and p is faulty, the property of failure detector is trivially ensured.

```
/* CODE FOR PROCESS $p$ */
function consensus($v$) with the failure detector fd
    variables:
        Flag[1...n] $\in \{true, false\}^n$;
                $\forall i \in \Pi$, Flag[i] is initialized to false
        V[1...n]: array of propositions;
                $\forall i \in \Pi$, V[i] is initialized to $\bot$
        Mes[1...n]: array of arrays of propositions;
                $\forall i \in \Pi$, Mes[i] is initialized to $\bot$
        $r$: integer; $r$ is initialized to 1
    begin
        V[p] $\leftarrow v$ the proposed values
        Mes[1] $\leftarrow$ V
        while ($r \leq n$) do
            send$\langle$R-$r$, Mes[r]$\rangle$ to every process,
                    except $\{p\} \cup$ fd
            for all $i \in \Pi \setminus (\text{fd} \cup \{p\})$, Flag[i] = $false$ do
                if (rcv$\langle$R-$r$, $W\rangle$ from $i$) then
                    Flag[i] $\leftarrow true$
                    if $r < n$ then
                        if V[i] $= \bot$ then
                            V[i] $\leftarrow W[i]$; Mes[r+1][i] $\leftarrow W[i]$
                        end if
                    else
                        if V[i] $\neq \bot$ then
                            V[i] $\leftarrow W[i]$
                        end if
                    end if
                end if
            end for
            if $\forall q \in \Pi \setminus (\text{fd} \cup \{p\})$, Flag[q] $= true$ then
                if $r < n$ then
                    for all $i \in \Pi$ do
                        Flag[i] $\leftarrow false$
                    end for
                end if
                if $r = n - 1$ then
                    Mes[n] $\leftarrow$ V
                end if
                $r \leftarrow r + 1$
            end if
            for all $i \in \Pi \setminus (\text{fd} \cup \{p\})$ do
                if (rcv$\langle$Decide, $d\rangle$ from $i$) then
                    $return(d)$
                end if
            end for
        end while
        $d$ = the first component of V different from $\bot$
        $return(d)$
    end
end function
```

Figure 5: Algorithm $\mathcal{RCP}$, repeated consensus with $\mathcal{P}^-$. Part 1: function consensus()

Decide message or a Start message from every process trusted by its failure detector module (in $\mathcal{P}^-$). In the second one, each process sends a Decide message and waits to receive a Start message or a $\langle$R-$x, -\rangle$ message. Actually, due to message loss it is possible that a process goes to its second round despite some process have not received its Decide message, but it cannot finish the second round before every correct processes have finished the first one.

As a faulty process can be suspected by $\mathcal{P}^-$ before it crashes, it is possible that a faulty process will not be waited by other processes although it is still alive. To avoid that this process disturbs the round, when a process p has suspected a process q, p never waits a message from this process and p never sends a message to this process.

Note also that if the consensus function is executed with $\mathcal{P}^-$, then there is no need to send $\langle$R-$x, -\rangle$ in round $r > x$ again. We have rewritten the consensus function to take these facts in account, but the behaviour remains the same.

Theorem 5 *Algorithm $\mathcal{RCP}$ (Figure 5 and 6) is a repeated consensus algorithm in $\Phi^{\mathcal{F}}$ with $\mathcal{P}^-$.*

Corollary 3 *$\mathcal{P}$ is sufficient for solving repeated consensus in $\Phi^{\mathcal{F}}$.*

Contrary to these results in system $\Phi^{\mathcal{F}}$, in system $\Phi^{\mathcal{I}}$, we have the same weakest failure detector to solve the consensus problem and the repeated consensus problem :

Proposition 2 *In system $\Phi^{\mathcal{I}}$, if there is an algorithm $\mathcal{A}$ with failure detector $\mathcal{D}$ solving the consensus problem then there exists an algorithm solving repeated consensus with $\mathcal{D}$.*

```
    /* CODE FOR PROCESS $p$ */
variables:
    FD: failure detector of type $\mathcal{P}^-$
procedure R-PROPOSED($v$)
    variables:
        FlagR$[1 \ldots n] \in \{true, false\}^n$;
                $\forall i \in \Pi$, FlagR$[i]$ is initialized to false
        stop: boolean; stop is initialized to false
        $u$: integer;
    begin
        $u \leftarrow$consensus($v$) with FD
        R-DECIDE($u$)
        repeat
            send⟨Decide, $u$⟩ to every process,
                        except $\{p\} \cup$ FD
            for all $i \in \Pi \setminus ($FD$ \cup \{p\})$, FlagR$[i] = false$ do
                if (rcv⟨Decide, $u$⟩ from $i$) $\vee$
                    (rcv⟨Start⟩ from $i$) then
                    FlagR$[i] \leftarrow true$
                end if
            end for
            if $\forall q \in \Pi \setminus ($FD$ \cup \{p\})$, FlagR$[q] = true$ then
                stop $\leftarrow true$
            end if
        until stop
        for all $i \in \Pi$ do
            FlagR$[i] \leftarrow false$
        end for
        stop $\leftarrow false$
        repeat
            send⟨Start⟩ to every process,
                        except $\{p\} \cup$ FD
            for all $i \in \Pi \setminus ($FD$ \cup \{p\})$, FlagR$[i] = false$ do
                if (rcv⟨Start⟩ from $i$) $\vee$
                    (rcv⟨R-1, $W$⟩ from $j$) then
                    FlagR$[i] \leftarrow true$
                end if
            end for
            if $\forall q \in \Pi \setminus ($FD$ \cup \{p\})$, FlagR$[q] = true$ then
                stop $\leftarrow true$
            end if
        until stop
    end
end procedure
```

Figure 6: Algorithm $\mathcal{RCP}$, repeated consensus with $\mathcal{P}^-$. Part 2.

References

[1] Marcos K. Aguilera, Sam Toueg, and Boris Deianov. Revisiting the weakest failure detector for uniform reliable broadcast. In *DISC '99: Proceedings of the thirteenth International Symposium on Distributed Computing*, pages 13–33, LNCS vol. 1693. Springer-Verlag, September 1999.

[2] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over halfduplex links. *Journal of the ACM*, 12:260–261, 1969.

[3] Rida A. Bazzi and Gil Neiger. Simulating crash failures with many faulty processors (extended abstract). In *6th International Workshop on Distributed Algorithms (WDAG '92)*, volume 647 of *Lecture Notes in Computer Science*, pages 166–184. Springer, 1992.

[4] David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors, *ADHOC-NOW*, volume 3158 of *Lecture Notes in Computer Science*, pages 135–148. Springer, 2004.

[5] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[7] Benny Chor and Brian A. Coan. A simple and efficient randomized byzantine agreement algorithm. *IEEE Trans. Software Eng.*, 11(6):531–539, 1985.

[8] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. A realistic look at failure detectors. In *DSN*, pages 345–353. IEEE Computer Society, 2002.

[9] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental

problems in distributed computing. In *Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC 2004)*, pages 338–346, 2004.

[10] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Petr Kouznetsov. Mutual exclusion in asynchronous systems with failure detectors. *Journal of Parallel and Distributed Computing*, 65(4):492–505, April 2005.

[11] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[12] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[13] Jonathan Eisler, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector to solve nonuniform consensus. *Distributed Computing*, 19(4):335–359, 2007.

[14] Antonio Fernández, Ernesto Jiménez, and Michel Raynal. Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. In *DSN*, pages 166–178. IEEE Computer Society, 2006.

[15] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[16] Eli Gafni and Leslie Lamport. Disk paxos. *Distributed Computing*, 16(1):1–20, 2003.

[17] Fabíola Greve and Sébastien Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *DSN*, pages 82–91. IEEE Computer Society, 2007.

[18] Rachid Guerraoui, Michal Kapalka, and Petr Kouznetsov. The weakest failure detectors to boost obstruction-freedom. In *DISC '06: Proceedings of the twentieth International Symposium on Distributed Computing*, pages 399–412, LNCS vol. 4167. Springer-Verlag, September 2006.

[19] Rachid Guerraoui and André Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, 2001.

[20] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR 94-1425, Department of Computer Science, Cornell University, 1994.

[21] Joseph Y. Halpern and Aleta Ricciardi. A knowledge-theoretic analysis of uniform distributed coordination and failure detectors. In *Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 73–82, 1999.

[22] Nancy A. Lynch, Yishay Mansour, and Alan Fekete. Data link layer: Two impossibility results. In *Symposium on Principles of Distributed Computing*, pages 149–170, 1988.

[23] Michel Raynal and Corentin Travers. In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. In Alexander A. Shvartsman, editor, *OPODIS*, volume 4305 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2006.

[24] Vic Stenning. A data transfer protocol. *Computer Networks*, 1:99–110, 1976.

[25] Piotr Zielinski. Anti-omega: the weakest failure detector for set agreement. Technical Report UCAM-CL-TR-694, Computer Laboratory, University of Cambridge, Cambridge, UK, July 2007.

A Proof of Lemma 1

Let $\mathcal{A}$ be an algorithm solving `reliable-broadcast` *in $\Phi^{\mathcal{F}}$ with a failure detector $\mathcal{D}$. There exists an integer k such that for every process p and every correct process q, for every run R of $\mathcal{A}$ where p* `BROADCAST`*s and* `DELIVER`*s k messages, at least one message from q has been received by some process.*

Let x be the maximum number of messages which can be stored in the memory of the processes. x exists since the memory of processes are finite. Let $k = x \times (n-1) + 1$. Consider a process p and a correct process q. Assume by contradiction that there exists a run R of $\mathcal{A}$ in which:

- p `BROADCAST`s and `DELIVER`s k messages.
- No message from q has been received by any process.

Consider the time τ at which process p `DELIVER`s the k^{th} message. As no message from q has been received, there exists another run R' in which:

- All processes except process q take the same steps, send and receive the same messages as in R.
- All messages to q are lost (this does not contradict the fairness hypothesis of the link).
- At time τ all messages in the links are lost (this does not contradict the fairness hypothesis of the links).

At this point the knowledge of at least one message has disappeared: it is neither in the memory of the processes, nor in the links, nor given by the failure detector (which output is a function of time and failure pattern). q can't `DELIVER` all the k messages. This contradicts the agreement property of the `reliable-broadcast`. $\square$

B Proof of Lemma 2

Let $\mathcal{A}$ be an algorithm solving `repeated-consensus` *in $\Phi^{\mathcal{F}}$ with a failure detector $\mathcal{D}$. There exists an integer k such that for every process p and every correct process q, for every run R of $\mathcal{A}$ where p* `R-PROPOSE`*s and* `R-DECIDE`*s k times, at least one message from q has been received by some process.*

Let x be the maximum number of values from the set of proposed values which can be stored in the memory of the processes. x exists since the memory of processes are finite. Let $k = x \times (n-1) + 1$. Consider a process p and a correct process q. Assume by contradiction that there exists a run R of $\mathcal{A}$ in which:

- p `R-PROPOSE`s and `R-DECIDE`s k times.
- No message from q has been received by any process.

Consider the time τ at which process p `R-DECIDE`s for the k^{th} times. As no message from q has been received, there exists another run R' in which:

- All processes except process q take the same steps, send and receive the same message as in R.
- All messages to q are lost (this does not contradict the fairness hypothesis of the link).

- At time τ all messages in the links are lost (this does not contradict the fairness hypothesis of the links).

At this point the knowledge of at least one decision has disappeared: it is neither in the memory of the processes, nor in the links, nor given by the failure detector (which output is a function of time and failure pattern). q can't `R-DECIDE` all the k values. This contradicts the agreement property of the `repeated-consensus`. $\Box$

C Proof of Theorem 2

We now show that Algorithm $\mathcal{B}$ solves the reliable broadcast problem (using a failure detector of type $\mathcal{P}^-$ and finite local memories on processes) in any system $\Phi^{\mathcal{F}}$.

We begin the proof by formally defining the notions of "start" and "termination".

Definition 8 (Start) *When a process p switches* `Mes` *from $\perp$ to* m *with* m $\neq \perp$*, we say that p* starts *a broadcast of the message* m*.*

Definition 9 (Termination) *When a process p switches* `Mes` *from* m *to $\perp$ with* m $\neq \perp$*, we say that the broadcast of* m *(invoked by p)* terminates*.*

Definition 10 (Player) *Let $k \in \mathbb{N}^+$. Let q be a correct process. Let q-Player(p,k) be the set of processes defined as follows:*

- *q-Player$(p,k) = correct(F)$* **if** *p crashes before starting its k^{th} broadcast,*
- *q-Player$(p,k) = \Pi \setminus \mathtt{FD}_q^{\tau}$ with τ, the time where p starts its k^{th} broadcast,* **otherwise***.*

Remark 1 *Let q be a correct process. Due to the quality of $\mathcal{P}^-$, we have $\forall k \in \mathbb{N}^+$, q-Player$(p,k) \subseteq$ q-Player$(p,k-1)$ and $Correct(F) \subseteq q$-Player(p,k) .*

Definition 11 (Participating To/Still Executing) *We say that a process q is* participating to *or* still executing *the k^{th} broadcast of the process p if:*

- *$\mathtt{Mes}_q[p] = \mathrm{m}$ when* m *if the k^{th} message broadcasted by p,*
- *$\mathtt{Flag}_q[p][q] = k \bmod 2$, and*
- *$\exists j \in \Pi \setminus \mathtt{FD}_q$, if j is not crashed, then $\mathtt{Flag}_q[p][q] \neq \mathtt{Flag}_q[p][j]$.*

Definition 12 (Waiting For) *We say that a process q is* waiting for *the k^{th} broadcast of the process p at time τ if:*

- *$\exists t \leq \tau$, $\mathtt{Mes}_q[p] = \mathrm{m}_{k-1}$ where m_{k-1} is the $(k-1)^{th}$ message broadcasted by p,*
- *$\forall t \leq \tau$, $\mathtt{Mes}_q[p] \neq \mathrm{m}_k$ where m_k is the k^{th} message broadcasted by p, and*
- *$\forall j \in \Pi \setminus \mathtt{FD}_q^{\tau}$, if j is not crashed, then $\mathtt{Flag}_q[p][j] = (k-1) \bmod 2$.*

The technical lemma below will be used in all the other proofs.

Lemma 3 *When the i^{th} broadcast invoked by the process p starts, $\forall q \in correct(F) \cup \{p\}$, we have:*

(1) $\forall j \in$ q-Player(p,i), either (a) j is crashed or (b) $\texttt{Flag}_q[p][j] = (i-1) \bmod 2$, $\texttt{Flag}_j[p][j] = (i-1) \bmod 2$, *and* $\texttt{Flag}_j[p][q] = (i-1) \bmod 2$.

(2) Any p-`BRD` *message in transit from q to another process j in q-Player$(p,i) \setminus \{p\}$ is of the form $\langle p$-*`BRD`*$,-,(i-1) \bmod 2\rangle$, if j is not crashed.*

(3) Any p-`ACK` *message in transit to q from another process in q-Player$(p,i) \setminus \{p\}$ is of the form $\langle p$-*`ACK`*$,(i-1) \bmod 2\rangle$.*

(4) $\texttt{Mes}_j[p] \in \{\bot, \mathrm{m}_{i-1}\}$ *where m_{i-1} is the $(i-1)^{th}$ message broadcasted by p, except p that sets* $\texttt{Mes}_p[p]$ *to m_i where m_i is the i^{th} message it broadcasts.*

Proof. We prove this lemma by induction on the sequence number of the broadcasts invoked by p.

Case i = 1. At the initialization, $\forall q \in correct(F) \cup \{p\}$, q sets $\texttt{Mes}_q[p]$ to $\bot$ and $\forall j \in$ *q-Player*(p,i), q sets $\texttt{Flag}_q[p][j]$ to 0. If not crashed, j both sets $\texttt{Flag}_j[p][j]$ and $\texttt{Flag}_j[p][q]$ to 0, and $\texttt{Mes}_i[p]$ to $\bot$. Then, until p starts its first broadcast, no p-`BRD` message is sent, no p-`ACK` message is sent, and as a consequence, $\texttt{Flag}_q[p][j]$ remains equal to 0, $\texttt{Mes}_q[p]$ remains equal to $\bot$ and if j is not crashed, $\texttt{Flag}_j[p][q]$ remains equal to 0 and $\texttt{Mes}_j[p]$ remains equal to $\bot$. Thus, when p starts its first broadcast, there is no p-`BRD` or p-`ACK` message in transit, and $\forall q \in correct(F) \cup \{p\}$, $\forall j \in$ *q-Player*(p,i), we have $\texttt{Flag}_q[p][j] = 0$, $\texttt{Flag}_j[p][j] = 0$, $\texttt{Flag}_j[p][q] = 0$, and $\texttt{Mes}_j[p] = \bot$ except for $j = p$: $\texttt{Mes}_p[p]$ is set to m. Hence, the induction holds for $i = 1$.

Induction Assumption: Assume that $\exists k \in \mathbb{N}^+$ such that when the k^{th} broadcast invoked by p starts, $\forall q \in correct(F) \cup \{p\}$, we have:

(1) $\forall j \in$ *q-Player*(p,k), either (a) j is crashed or (b) $\texttt{Flag}_q[p][j] = (k-1) \bmod 2$, $\texttt{Flag}_j[p][j] = (k-1) \bmod 2$, and $\texttt{Flag}_j[p][q] = (k-1) \bmod 2$.

(2) Any p-`BRD` message in transit from q to another process j in *q-Player*$(p,i) \setminus \{p\}$ is of the form $\langle p$-`BRD`$,-,(k-1) \bmod 2\rangle$, if j is not crashed.

(3) Any p-`ACK` message in transit to q from another process in *q-Player*$(p,i) \setminus \{p\}$ is of the form $\langle p$-`ACK`$,(k-1) \bmod 2\rangle$.

(4) $\texttt{Mes}_j[p] \in \{\bot, m_{k-1}\}$ where m_{k-1} is the $(k-1)^{th}$ message broadcasted by p, except p that sets $\texttt{Mes}_p[p]$ to m_k where m_k is the k^{th} message it broadcasts.

Case i = k + 1. As we study the case $i = k+1$, let us assume that p eventually start its $(k+1)^{th}$ broadcast.

Let $q \in correct(F) \cup \{p\}$. Let m be the k^{th} message broadcasted by p. Consider two cases:

- $q = p$. First, $\texttt{Mes}_p[p]$ is set to m when p starts its k^{th} broadcast. Then, **(i)** $\texttt{Mes}_p[p]$ remains equal to m until p terminates the broadcast and, from the termination of the broadcast to the start of the next one, p satisfied $\texttt{Mes}_p[p] = \bot$.

  By induction assumption, when the k^{th} broadcast invoked by p starts, $\forall j \in$ *p-Player*(p,k), either (a) j is crashed or (b) $\texttt{Flag}_p[p][j] = (k-1) \bmod 2$, $\texttt{Flag}_j[p][j] = (k-1) \bmod 2$, and $\texttt{Flag}_j[p][q] = (k-1) \bmod 2$. Due to the quality of `FD`, **(ii)** the broadcast then terminates when at least $\forall j \in$ *p-Player*$(p,k+1)$, either j is crashed or $\texttt{Flag}_p[p][j] = k \bmod 2$. In this case, **(iii)** these later values remains constant until at least p starts its $(k+1)^{th}$ broadcast.

  By induction assumption, **(iv)** when the k^{th} broadcast invoked by p starts, $\forall j \in$ *p-Player*(p,k) $\setminus \{p\}$, either j is correct and, as a consequence, j is waiting for the k^{th} broadcast of p or j is faulty and is waiting for the k^{th} broadcast of p, still executing the $(k-1)^{th}$ broadcast of p, or is crashed. Now, p waits that either j crashes or j acknowledges with a $\langle p\text{-}\mathtt{ACK}, k \bmod 2\rangle$ message. If j crashes, then either p suspects j before receiving the valid acknowledgement, or p eventually receives the valid acknowledgment, or p eventually flushes the link (q,p) from any p-ACK until it suspects q. If j does not crash and is not suspected by p during the broadcast, p eventually receives a valid acknowledgement from j otherwise p never terminates the k^{th} broadcast and as a consequence never starts the $(k+1)^{th}$ (a contradiction). Hence, **(v)** when p starts its $(k+1)^{th}$ broadcast, it received a valid acknowledgement ($\langle \mathtt{ACK}, k \bmod 2\rangle$) from any $j \in p\text{-}Player(p, k+1) \setminus \{p\}$ and, as a consequence, any j receive a $\langle \mathtt{BRD}, m, k \bmod 2\rangle$ from p.

To send such an acknowledgement, any $j \in p\text{-}Player(p, k+1) \setminus \{p\}$ must first wait for the k^{th} broadcast of p and then receives a $\langle \mathtt{BRD}, -, k \bmod 2\rangle$ message. The first of these processes, say j_0, to receive such a message can only receives it from p due to the induction assumption. Then, when j_0 receives such a message it sets $\mathtt{Mes}_{j_0}[p]$ to m (due to (i) and the induction assumption) and starts sending $\langle \mathtt{BRD}, m, k \bmod 2\rangle$ messages. Then, the second process to receive the message, receives it from p or j_0, and so on. Hence, when p a $\langle \mathtt{ACK}, k \bmod 2\rangle$ message from j, it has the guarantee that j starts participating to the k^{th} broadcast of p. Finally, any j sends it only when they are waiting for the $(k+1)^{th}$ broadcast of p. As previously, **(vi)** every process j waits for the $(k+1)^{th}$ broadcast of p until at least one receives a $\langle \mathtt{BRD}, -, (k+1) \bmod 2\rangle$ messages from p, *i.e.*, until at least p starts its $(k+1)^{th}$ broadcast. Another consequence is that **(vii)** any j satisfies $\mathtt{Mes}_j[p] \in \{\perp, m\}$ until at least p starts its $(k+1)^{th}$ broadcast.

To sum up, when p starts its $(k+1)^{th}$ broadcast, Point (1) of the induction holds thanks to **(ii)**, **(iii)**, and **(vi)**; Points (2) and (3) holds thanks to **(iv)**, **(v)**, **(vi)**, and the FIFO property; and Points (4) holds thanks to **(i)** and **(vii)**.

- $q \in correct(F) \setminus \{p\}$. Similary to the previous case, we can show $\forall j \in q\text{-}Player(p, k+1)$, Point (1) to (4) are satisfied for any j when p starts its $(k+1)^{th}$ broadcast.

Hence, the induction holds for $i = k+1$. □

In our algorithm, the broadcasts are executed in sequence. We now show that every broadcast terminates in finite time if the broadcastor process does not crash. Hence, the current broadcast cannot prevent forever the next one to be performed.

Lemma 4 *If a process does not crash during the broadcast it starts, then the broadcast terminates in finite time.*

Proof. Assume, by the contradiction, that a broadcast started by the process p never terminates despite p never crashes, *i.e.*, p is correct. When p starts the broadcast, $\mathtt{Flag}_p[p][p] = (k-1) \bmod 2$ by Lemma 3. Then, $\mathtt{Flag}_p[p][p]$ is incremented to $k \bmod 2$ and becomes constant.

As the broadcast never terminates, the test of Line 23 is never satisfied at p for $i = p$. Now, p never suspects any correct process and FD eventually outputs the exactly list of the correct process. So, eventually $\Pi \setminus \mathtt{FD}$ becomes constant and as the number of processes is finite, there exists a correct process $q \neq p$ such that $\mathtt{Flag}_p[p][p] \neq \mathtt{Flag}_p[p][q]$ holds each time p executes Line 23 with $i = p$. In the algorithm, once $\mathtt{Flag}_p[p][p] = \mathtt{Flag}_p[p][q]$ holds, $\mathtt{Flag}_p[p][p] = \mathtt{Flag}_p[p][q]$ holds continuously until p terminates its current broadcast. Hence, $\mathtt{Flag}_p[p][p] \neq \mathtt{Flag}_p[p][q]$ holds forever and, as a consequence, p sends infinitively many $\langle p\text{-}\mathtt{BRD}, -, k \bmod 2\rangle$ messages to q. As the link from p to q is fair and q tries to receive p-BRD message from p infinitively often (due to the quality of FD and as p

never crashes, p is never suspected by q), q receives infinitively many $\langle p$-BRD,$-$,k mod 2$\rangle$ messages from p.

By Lemma 3, when p starts the broadcast, q is waiting for the k^{th} broadcast message of p. As q eventually received a $\langle p$-BRD,$-$,k mod 2$\rangle$ from p, $\texttt{Flag}_q[p][p]$ and $\texttt{Flag}_q[p][q]$ are eventually set to k mod 2. From this point on, $\texttt{Flag}_q[p][p]$ and $\texttt{Flag}_q[p][q]$ remains constant until q waits the $(k+1)^{th}$ broadcast message of p.

If q is eventually waiting for the $(k+1)^{th}$ broadcast message of p (recall p never starts its $(k+1)^{th}$ broadcast since we assume that the k^{th} broadcast of p never terminates), then q acknowledges any $\langle p$-BRD,$-$,k mod 2$\rangle$ messages it receives from p with $\langle p$-ACK,k mod 2$\rangle$ messages. Hence, q sends infinitively many $\langle p$-ACK,k mod 2$\rangle$ messages to p and as the links are fair and p tries to receive p-ACK messages from q infinitively often (due to the quality of FD and as q is correct, q is never suspected by p), p eventually receives such a message from q and $\texttt{Flag}_p[p][p] = \texttt{Flag}_p[p][q]$ eventually holds, a contradiction.

Assume now that q is never waiting for the $(k+1)^{th}$ broadcast message of p. Then, q executes the k^{th} broadcast of p forever. As previously, there exists a process j in $correct(F) \setminus \{p,q\}$ such that $\texttt{Flag}_q[p][j] = (k-1)$ mod 2 forever. In this case, q sends infinitively many $\langle p$-BRD,$-$,k mod 2$\rangle$ messages to j. As the links are fair and j tries to receive p-BRD from q infinitively often (due to the quality of FD and as q is correct, q is never suspected by j), j receives infinitively many $\langle p$-BRD,$-$,k mod 2$\rangle$ messages from q. As a consequence, j sends infinitively many $\langle p$-ACK,k mod 2$\rangle$ to q (at the beginning j was waiting for the k^{th} broadcast of p by Lemma 3 and j never starts the $(k+1)^{th}$ broadcast of p because p never does it too) and as the links are fair, q eventually receives such a message from j and so $\texttt{Flag}_q[p][j] = k$ mod 2 eventually holds, a contradiction. □

If a process p does not crash during the broadcast of the message m it starts, then p executes DELIVER(m) once during the broadcast: at its termination. Hence, we can deduce the two following corollary from Lemma 4:

Corollary 4 (Validity) *Let p be a correct process. In any* BROADCAST(m) *invoked by p, p executes* DELIVER(m).

Corollary 5 *Let p be a correct process. In any* BROADCAST(m) *invoked by p, p executes* DELIVER(m) *at most once.*

Lemma 5 (Agreement) *If a process executes* DELIVER(m)*, then all correct processes eventually execute* DELIVER(m).

Proof. Let m be the k^{th} message broadcast by some process p. Assume then by the contradiction that some process j eventually executes DELIVER(m) but a correct process q does not.

First, by Lemma 3, q is waiting for the k^{th} broadcast of p when p starts its broadcast. Then, j is either waiting for the k^{th} broadcast of p or still executing the $(k-1)^{th}$ broadcast of p by Lemma 3. By assumption, j is eventually waiting for the k^{th} broadcast of p and eventually receives a $\langle p$-BRD,m,k mod 2$\rangle$ message from a process in j-$Player(p,k)$. From that point, j starts sending $\langle p$-BRD,m,k mod 2$\rangle$ messages and eventually receives an acknowledgment of q for the reception of one of its $\langle p$-BRD,m,k mod 2$\rangle$ message, otherwise j never executes DELIVER(m).

After receiving the first $\langle p$-BRD,m,k mod 2$\rangle$ message, q starts sending $\langle p$-BRD,m,k mod 2$\rangle$ messages until it receives a $\langle p$-ACK,k mod 2$\rangle$ message from any process i in $\Pi \setminus$ FD. Due to the quality of FD, $\Pi \setminus$ FD is eventually equal to $correct(F)$. So, there is a correct process v from which q never receives an acknowledgment for m: $\texttt{Flag}_q[p][v] \neq k$ mod 2 holds forever. As a consequence, (*) q

sends infinitively many $\langle p$-BRD,m,k mod 2$\rangle$ to v. By Lemma 3, (**) v is waiting for the k^{th} broadcast of p when p starts its broadcast. By (*), (**), and the fact that the links are fair, v eventually receives a $\langle p$-BRD,m,k mod 2$\rangle$ message and then acknowledges any p-BRD message it receives from q with a $\langle p$-ACK,k mod 2$\rangle$ message until it starts participating to the $(k+1)^{th}$ broadcast of p. Now, as q never executes DELIVER(m), q is never waiting for the $(k+1)^{th}$ broadcast of p and, as a consequence, q never acknowledges the $\langle p$-BRD,m,k mod 2$\rangle$ messages it receives from p. Hence, p never starts its $(k+1)^{th}$ broadcast. So, by the fair property of the links, v receives infinitively many $\langle p$-BRD,m,k mod 2$\rangle$ messages from q and as a consequence send infinitively many $\langle p$-ACK,k mod 2$\rangle$ messages. Yet by the fairness property of the links, q eventually receives a $\langle p$-ACK,k mod 2$\rangle$ message from v, a contradiction. □

Lemma 6 (Integrity) *For every message* m*, (1) every process executes* DELIVER(m) *at most once, and (2) only if sender*(m) *previously invokes* BROADCAST(m)*.*

Proof.

- **Proof of Claim (1):** Let m by the k^{th} message broadcasted by p. Consider a process q.

  If $q = p$, then q executes DELIVER(m) at most once by Corollary 5.

  Assume now that $q \neq p$ and q executes DELIVER(m). When this event occurs, q sets Mes[p] to $\perp$ and q is waiting for the $(k+1)^{th}$ broadcast of p: the next message q will DELIVER will be a message encapsulated into a p-BRD message marked with the value $(k+1)$ mod 2. By Lemma 3, q will receive such a p-BRD only after p starts its $(k+1)^{th}$ broadcast. Once p starts its $(k+1)^{th}$ broadcast, q ignores any p-BRD marked with the value k mod 2 until it DELIVER the $(k+1)^{th}$ broadcast message of p. Lemma 3 implies that if q eventually DELIVER the $(k+1)^{th}$ broadcast message of p, then it previously received acknowledgment for any other process in q-$Player(p, k+1)$ meaning that they are participating to the $(k+1)^{th}$ broadcast message of p. At this time, p receives no p-BRD message contaning m forever and, as a consequence, never more executes DELIVER(m). Hence, if $q \neq p$, then q executes DELIVER(m) at most once.

- **Proof of Claim (2):** (contraposition) Assume that *sender*(m) never invokes BROADCAST(m). Then, MesToBrd never returns m and, as a consequence, $\texttt{Mes}_{sender(m)}[sender(m)]$ is never set to m. So, *sender*(m) never executes DELIVER(m) and never sends any $\langle p$-BRD,m,$-\rangle$ message to any other process. As a consequence, any process $q \neq sender(m)$ also never set $\texttt{Mes}_q[p]$ to m and, as a consequence, never executes DELIVER(m).

□

Proof of Theorem 2: By Corollary 4, Lemma 5, and 6, any run of Algorithm $\mathcal{B}$ satisfies reliable-broadcast. □

D Proof of Theorem 3

We now show that Algorithm $\mathcal{CS}$ solves the reliable broadcast problem (using a failure detector of type $\mathcal{P}^-$ and finite local memories on processes) in any system $\Phi^{\mathcal{F}}$.

To prove that the Algorithm $\mathcal{CS}$ (Figure 3) is a consensus algorithm in $\Phi^{\mathcal{F}}$ with $\mathcal{S}$, we first have to prove that all correct processes decide, then the proof is exactly the same as the proof of the Chandra and Toueg algorithm [6].

Lemma 7 *For every correct process p that propose v, p terminates* $\texttt{PROPOSE}(v)$.

Proof. Assume by contradiction that some correct process doesn't decide. In this case every correct process that never decides loops forever in one of its n asynchronous rounds. Consider two cases:

- One correct process p decides: There is a time after which p sends only $\langle \texttt{Decide}, - \rangle$ messages and so, by FIFO property there is a time τ, after which every process can only receive $\langle \texttt{Decide}, - \rangle$ messages from p. Consider a correct process, say x, that never decides. It loops forever in some round, say rx. After time τ, each time process x executes this round rx, it sends a $\langle \texttt{R-}rx, - \rangle$ message to every others processes and in particular to p and it can only send this message. By fairness of the link the link from p to x, p receives an infinity of this message Line 61 and sends an infinity of $\langle \texttt{Decide}, - \rangle$ to x Line 62. By fairness of the links too, x receives at least one of this message Line 44 and decides.

- No correct process decides: Let r be the lowest rounds in which a least one correct process loops forever. Let pr be one of these processes. By FIFO property of the links, there is a time τ, after which no message of lower rounds than rounds r are received. In round r, pr waits forever some message $\langle \texttt{R-}r, - \rangle$ of some process q trusted by its $\texttt{FD}$. By completeness of $\texttt{FD}$, q is correct. After time τ, there is two cases:
  - q loops forever in round r: In this case, each time q executes its loop it sends $\langle \texttt{R-}r, - \rangle$ to pr. As we are after time τ, process q does not receive any message of a process in a lower round. So q does not send any other messages than $\langle \texttt{R-}r, - \rangle$ to pr. By the fairness of the link, pr receives this message.
  - q loops forever in a bigger round: In this case, each time pr executes its loop it sends $\langle \texttt{R-}r, - \rangle$ to q. As we are after time τ, process pr does not receive any message of process q in a lower round. So pr does not send any other messages than $\langle \texttt{R-}r, - \rangle$ to q. By the fairness of the link, q receives an infinity of this message Line 28 and sends an infinity of $\langle \texttt{R-}r, - \rangle$ to pr. As we are after time τ, process q can only receive message of process pr in round r. Again by the fairness of the link pr receives this message.

$\square$

Proof of Theorem 3: Immediat from Lemma 7 and [6]. $\square$

E Proof of Theorem 5

We now show that Algorithm $\mathcal{RCP}$ (Figures 5 and 6) solves the repeated consensus problem (using a $\mathcal{P}^-$ and finite local memories on processes).

We begin by a technical lemma:

Lemma 8 *For every integer i, for every process p, for every process q*

- *(part 1) Messages $\langle \texttt{R-}x, - \rangle$ sent by p to q while executing i^{th} $\texttt{R-PROPOSED}(-)$ can be only received (1) Line 30 of the $(i-1)^{th}$ $\texttt{R-PROPOSED}(-)$ if $(i > 1)$ or (2) Lines 14 of the consensus function in the i^{th} $\texttt{R-PROPOSED}(-)$*

- *(part 2) Messages $\langle \texttt{Decide}, - \rangle$ sent by p to q while executing i^{th} $\texttt{R-PROPOSED}(-)$ can be only received by q (1) Line 39 of the consensus function in the i^{th} $\texttt{R-PROPOSED}(-)$ or (2) Line 15 in the i^{th} $\texttt{R-PROPOSED}(-)$*

- *(part 3) Messages* $\langle$`Start`$\rangle$ *sent by p to q while executing* i^{th} `R-PROPOSED`$(-)$ *can be only received by q (1) Line 15 in the* i^{th} `R-PROPOSED`$(-)$ *or (2) Line 30 in the* i^{th} `R-PROPOSED`$(-)$

Proof. The proof is by induction on integer i

- (Case $i = 1$) To show the first part of the lemma, assume that some message $\langle R\text{-}x, -\rangle$ sent by process p while executed the first `R-PROPOSED`(v), is received by a process q while executing the j^{th} `R-PROPOSED`(v) with $j > 1$.

  Process q has ended the i^{th} `R-PROPOSED` and has received a message $\langle$`Decide`$, -\rangle$ or $\langle$`Start`$\rangle$ of every process that its failure detector module does not suspect (Line 20, Figure 6). There is two cases: (1) process q has received a message $\langle$`Decide`$, -\rangle$ or $\langle$`Start`$\rangle$ from p, by FIFO property process q can't receive a message $\langle R\text{-}x, -\rangle$ sent by process p while executed the i^{th} `R-PROPOSED`(v), (2) the failure detector at q suspects p. In this case, p will be never trusted again by the failure detector. And so after that q does not receive any message from p.

  The second and the third parts follow the same lines.

- (induction) Assume the lemma is true until the $i-1$ with $i > 1$. To show the first part assume that a message $\langle R\text{-}x, -\rangle$ sent by process p to q, while executing i^{th} `R-PROPOSED`$(-)$ is received by q in j^{th} `R-PROPOSED`$(-)$ with $(i > j)$ and not in Line 30 of the $(i-1)^{th}$ `R-PROPOSED`$(-)$.

  Before executing the i^{th} `R-PROPOSED`$(-)$, process p has executed the $(i-1)^{th}$ `R-PROPOSED`$(-)$ and terminates it before time t. And so, it has suspected q or it has wait a $\langle$`Decide`$, -\rangle$ message or a $\langle$`Start`$\rangle$ message from q. If q is suspected in the $(i-1)^{th}$ `R-PROPOSED`$(-)$ by the failure detector of p, q remains suspected in the i^{th} `R-PROPOSED`$(-)$ and so p does not send messages to q. If it has wait a $\langle$`Decide`$, -\rangle$ message or a $\langle$`Start`$\rangle$ message from q, by induction assumption part 2 and 3 this message can only send by q during the $(i-1)^{th}$ `R-PROPOSED`$(-)$. And so q has finished all j^{th} `R-PROPOSED` with $j < i-1$ and has executed the $i-1^{th}$ `R-PROPOSED`$(-)$ until at least Line 13, contradicting the assumption.

  To conclude the proof of the first part we have to show that no message $\langle R\text{-}x, -\rangle$ sent by process p while executed the first `R-PROPOSED`(v), is received by a process q while executing the j^{th} `R-PROPOSED`(v) with $j > 1$. The proof is the same that in case $i = 1$.

  The proof of the second and the third parts follows the same lines.

□

Lemma 9 *For every integer i, for processes p, q:*
(1) if process p is correct, the i^{th} `R-PROPOSED`$(-)$ *of process p terminates.*
(2) the i^{th} *decision value of process p and process q are the same*
(3) the i^{th} *decision value of process p is the* i^{th} *value proposed by some process*

Proof. The proof is by induction on integer i

- case $i = 1$: By Lemma 8, while executing the *consensus*() function, processes only receives messages sent by others processes while executing this first call. The proof that every correct process terminates *consensus*() function is similar to the Lemma 7. By a proof similar of theorem 3, we prove items (2) and (3) of the lemma.

  After the end of the *consensus*() function, each correct process p repetitively sends `Decide` message until it receives `Decide` message or `Start` message (Lines 12-22, Figure 6).

Assume that all correct processes are blocked forever in this loop. During this loop, p doesn't send any other message. So by the fairness of the links, each correct process receives `Decide` message from every other correct processes and goes to the next loop contracting the assumption.

Assume now that at least one correct process p is blocked in the first repeat loop and a least one process q has finished the second repeat loop (Lines 27-37, Figure 6). This case is impossible because to finish the second loop, by the accuracy of the failure detector q has to receive a `Start` message or $\langle R\text{-}1, -\rangle$ message from p. By Lemma 8, these messages can only be sent by q in the first `R-PROPOSED`$(-)$. And in our case p has never sent `Start` messages. Moreover, by the part 1 of the this lemma, $\langle R\text{-}1, -\rangle$ message from p can't be receive by q at this point.

So assume now that at least one correct process p is blocked in the first repeat loop and at least one correct process is blocked in the second repeat loop (the other correct processes are blocked either in the first repeat loop or in the second) Every correct process sends either `Decide` message or `Start` message to p infinitively often and only this message. By the fairness of the links, process p receives these messages and ends the first repeat loop.

So all correct processes end the first repeat loop.

In the same way we show that every correct process ends the second repeat loop. Notice that the consensus function works with `FD` in $\mathcal{P}^-$ and so every process waits the asynchronous rounds message following $\mathcal{P}^-$. And so the behaviour of `Start` and $\langle R\text{-}1, -\rangle$ messages is the same in this repeat loop that the behaviour of `Decide` and `Start` messages in the first repeat loop.

- Induction case

  By induction assumption every correct process terminates the j^{th} `R-PROPOSED`$(-)$ with $j < i$. So every correct process begins the i^{th} `R-PROPOSED`. The proof follows the same line that in the case $i = 1$.

□

Proof of Theorem 5: Immediat from Lemma 9. □

F Proof of Proposition 2

Note first that when the memory is infinite, fair links and reliable links are equivalent.

Proposition 3 *Provided reliable links and processes with an infinite amount of memory, if there is an algorithm $\mathcal{A}$ with failure detector $\mathcal{D}$ solving the consenbsus, then there exists an algorithm solving repeated consensus with $\mathcal{D}$.*

Proof. As the memory is infinite, we can use an infinite number of integer. From $\mathcal{A}$, it is possible to obtain a infinite number of instance of $\mathcal{A}$ named $\mathcal{A}_x$ where x is an integer so that the code of $\mathcal{A}_x$ is exactly the same as the code of $\mathcal{A}$, except that, each time a process sends a message m in $\mathcal{A}$, it sends now the message *x:m* and it receives only the messages prefixing by *x:*.

The failure detector $\mathcal{D}$ can be query by any process p at any time τ. Its output depends only of p, τ, and the failure pattern. So, every algorithm $\mathcal{A}_x$ solves consensus.

The repeated consensus algorithm simply uses the algorithm $\mathcal{A}_i$ when the i-th `R-PROPOSE`$(-)$, is invoked. The `DECIDE`$(-)$ of this algorithm gives the i-th `R-DECIDE`$(-)$.

□

Proof of Proposition 2: Immediat from Proposition 3. □