Consensus in the Crash-Recover Model

Rui Oliveira   Rachid Guerraoui   André Schiper

{oliveira,guerraoui,schiper}@lse.epfl.ch

Phone: +41 21 6934248   Fax: +41 21 6936770

Ecole Polytechnique Fédérale

Département d'Informatique

CH-1015 Lausanne, Switzerland

Summary

This paper presents a deterministic algorithm that solves consensus in asynchronous distributed systems where processes may crash and recover, messages may be lost, and failure detections may be inaccurate. Our algorithm has an early delivery property: in runs where no process crashes or is suspected to have crashed, consensus is reached only after two communication steps.

The paper shows that to solve consensus with processes that may crash and recover, more knowledge about failures is required than if we assume that crashed processes never recover.

Key words: Distributed algorithms - Asynchronous system - Unreliable failure detector - Crash-recovery model - Consensus

1 Introduction

Defining the conditions under which the consensus problem can be solved is a fundamental issue in fault-tolerant distributed systems, as many typical problems underlying reliable distributed applications can be viewed as variations of consensus. In particular, both the atomic broadcast [5] and the non-blocking (weak) atomic commitment problems [9] are shown to be solvable under exactly the same conditions as consensus.

In 1985, Fischer, Lynch and Paterson [8] proved an important impossibility result, stating that, even if one single process can crash, no deterministic algorithm can solve consensus in an *asynchronous* system, where no assumption is made about message delays and process relative speeds. More recently, Chandra, Hadzilacos and Toueg showed that the minimal conditions for solving consensus can be expressed in terms of knowledge about process failures, expressed by two properties: *eventual weak accuracy*, i.e., eventually some correct process (one which never crashes) is never suspected (to have crashed), and *weak completeness*, i.e., eventually every faulty process (that crashes) is suspected by every correct process [4]. Chandra and Toueg presented an algorithm that solves consensus assuming a majority of correct processes and a failure detector that guarantees *eventual weak accuracy* and *weak completeness* properties [5].

The work on failure detectors is a fundamental step towards defining a rigorous framework for fault-tolerant distributed computing. From a practical point of view however, the assumptions made in [4, 5] about the underlying system model may be viewed as too constraining, as the communication channels are considered to be reliable (no omission failures), and processes are supposed to never recover after a crash (what is called hereafter the *crash-stop* model). Transforming *real-world* communication channels into reliable channels is indeed feasible by masking omission failures, but very expensive because every message may need to be kept and retransmitted until it is acknowledged [10].

In this paper, we present a consensus algorithm that tolerates general omission failures and takes into account process recovery. As in [12], our algorithm has a low latency degree, as processes reach consensus within two communication steps, in runs where no process crashes or is suspected to have crashed (the most frequent runs in practice). This *early-delivery* property, together with the tolerance of process recovery and omission failures, makes our algorithm a good candidate for solving consensus related problems in practice.

For presentation generality, we describe our algorithm in an abstract model, called g/r-model (*green/red*), composed of *green* processes, *red* processes and *fair lossy* communication channels. The intuition behind the *green/red* distinction is that *green* processes are *well behaved*, whereas *red* processes are not (but do not behave maliciously). We introduce the notion of stubborn channels which abstract the minimal message loss fairness property of fair lossy communication channels. Stubborn channels provide precise semantics which help and ease the presentation and proof of our consensus algorithm. Our algorithm guarantees consensus agreement and validity (safety), no matter how the system behaves, and ensures consensus termination (liveness), with a majority of *green* processes, *stubborn* channels, and a *red detector of class* $\diamond\mathcal{S}_r$. This class, which we prove is the weakest for which consensus can be solved in the g/r-model abstract model, is defined by the properties (1) *eventual weak accuracy*, i.e., eventually some green process is never suspected, and (2) *strong completeness*, i.e., eventually every red process is suspected by every green process.

We compare the knowledge about process failures that is needed to solve consensus in the crash-recover model, with the knowledge about failures that is needed to solve consensus in the crash-stop model. Interestingly, whereas in the latter model, *eventual weak accuracy* and *weak completeness* are sufficient to solve consensus, the crash-recover model requires *eventual weak accuracy* and *strong completeness*. This highlights the difference between both models, and reflects the intuition that it is harder to solve consensus in the crash-recover model than in the crash-stop model.

The remainder of the paper is organized as follows. Section 2 formally defines the abstract g/r-model. In Section 3, we define stubborn communication channels. In Section 4, we describe our consensus algorithm and we prove its correctness. In Section 5, we evaluate the knowledge about failures that is needed to solve consensus in the g/r-model. Section 6 mentions some related work. Finally, Section 7 concludes the paper by discussing the practical applicability of our algorithm.

2 Green/Red process model

We consider an asynchronous distributed system composed of a set of processes that communicate by message passing. Asynchrony means that there is no bound on communication delays, nor on process relative speeds. Processes may crash, and later recover. We do not consider Byzantine failures. In a model in which processes do not recover after a crash, a correct process is defined as a process that does not crash. In a crash/recovery model, "correct" processses have to be defined differently. To avoid ambiguity, we introduce the notion of "green" and "red" processes, instead of correct/incorrect: a green process is a "correct" process in the crash/recovery model, while a "red" process is an incorrect process. Green and red processes are defined below.

Similarly to [5], we assume the existence of a distributed oracle, called *red detector* in our model, which provides information about the processes that are suspected to be red.

2.1 Processes and communication channels

We consider a set of n processes $\Sigma = \{p_1, p_2, \ldots, p_n\}$, completely connected through a set of unreliable communication channels. The channel connecting process p_i to process p_j is denoted c_{ij}. Process p_i, in order to send a message m to p_j, *hands over* the message to the channel c_{ij}. If channel c_{ij} does not lose m, then eventually the channel *hands over* m to process p_j. At that point, either message m is received by p_j[1], or p_j is subject to a *receive-omission* failure, in which case m is lost.

2.1.1 Fair Lossy channels

We assume the channels connecting our processes to be *fair lossy* channels. A fair lossy channel is defined by the following two properties:

Property 2.1 (No Creation) *If a channel* c_{ij} hands over *a message* m *to process* p_j*, then some process* p_i handed over m *to the channel* c_{ij}*.*

Property 2.2 (Fair Loss) *If a process* p_i hands over *an infinite number of messages to a channel* c_{ij}*, then the channel* c_{ij} *eventually* hands over *an infinite subset of these messages to process* p_j*.*

Property 2.1 states that no spurious or corrupt messages are created (which would be a source of Byzantine failures which our model explicitly does not cover). Property 2.2 is similar to the *Weak Loss Limitation* in [11] and the *Fair Lossy* channel in [3]. This property reflects the usefulness of the channel. Without such a property, any interesting distributed problem would be trivially impossible to solve.

2.1.2 Green and Red processes

Computation proceeds in steps of an *algorithm* [4]. An algorithm is a collection A of n deterministic automata $A(p_i)$ (one per process). In each (non null) step of an algorithm A, a process p_i performs the following actions: (1) p_i receives a message handed over by one of the channels c_{xi} ($x \in \{1..n\}$), or a "null" message

[1] A process receives a message only if it is handed over by a channel.

λ; (2) p_i queries and receives a value d from its *red* detector module D_i; (3) p_i changes its state and sends a message (possibly null) to some process in Σ. Action (3) is performed according to (a) the automaton $A(p_i)$, (b) the state of p_i at the beginning of the step, (c) the message received in action 1, and (d) the value d of action 2. A process p_i *takes a non null step* s, if p_i completely executes the three actions of s.

To model crashed processes, we introduce "null" steps. A null step is a step in which a process p_i suffers from receive-omission failures (no message is received by a process taking a null step), does not change its state, and does not send any message.

All processes execute an infinite number of steps (some of which can be null steps). A process that crashes and later recovers, executes only a finite number of null steps between its crash and its recovery. A process that crashes and never recovers executes an infinite number of null steps. Based on this model, green and red processes are defined as follows:

- *green* processes take only a finite number of null steps.
- *red* processes take an infinite number of null steps.

This definition implies that green processes crash only a finite number of times, and always recover after a crash. Red processes either (1) crash and recover an infinite number of times, or (2) never recover after a crash.

2.1.3 Green processes and fair lossy channels

The definition of green processes together with the fair loss property (Property 2.2) of the fair lossy channels leads to the following proposition:

Proposition 2.3 *Let p_i and p_j be two green processes. If p_i* sends *an infinite number of messages to p_j, then p_j eventually* receives *an infinite subset of these messages.*

PROOF: By Property 2.2, if p_i sends (hands over to the channel) an infinite number of messages to p_j, an infinite subset of these messages are eventually handed over to p_j. Since p_j is a green process, p_j takes an infinite number of non null steps, i.e., p_j can be subject to only a finite number of receive-omission failures. Therefore p_j receives an infinite subset of the messages sent by p_i. □

Notice that, if one of the processes is red, nothing can be ensured. If p_i is red, p_i might not be able to send an infinite number of messages to p_j, and if p_j is red, p_j might be subject to an infinite number of receive-omission failures. Nevertheless, nothing precludes red processes from communicating: a message sent by a red process might always be received by some (green or red) process, as well as a red process might always receive a message sent by some process.

2.2 Red detectors

In [5], Chandra and Toueg have introduced the notion of *failure detectors*. We call them here *red detectors*, as in the g/r-model their responsibility is to detect red processes. From the failure detector classes defined by Chandra and Toueg, we focus in this paper on the classes $\diamond\mathcal{S}$ and $\diamond\mathcal{W}$, and we make "syntactic" changes to define what we denote the $\diamond\mathcal{S}_r$ and the $\diamond\mathcal{W}_r$ classes of red detectors. Class $\diamond\mathcal{S}_r$ is the set of red detectors that satisfy the following two properties:

Strong completeness. Eventually every red process is permanently suspected by every green process.

Eventual weak accuracy. Eventually some green process is never suspected by any green process.

Class $\diamond\mathcal{W}_r$ is the set of red detectors that satisfy *eventual weak accuracy*, and the following property:

Weak completeness. Eventually every red process is permanently suspected by some green process.

3 Stubborn communication channels

3.1 The abstraction of stubborn channels

We introduce the concept of stubborn channels as an abstraction that adequately defines the communication semantics required by our consensus algorithm (Section 4), and therefore also eases its presentation. Stubborn channels can be obtained as a simple transformation (Section 3.2) of fair lossy channels in the g/r-model.

Stubborn communication channels are defined by the two primitives *stb-send* and *stb-receive*, and characterized by the no creation property (Property 2.1), and the following additional properties:

Property 3.1 (No Duplication) *Every message m that is* stb-sent *to a process p_i is* stb-received *by p_i at most once.*

Property 3.2 (k-Stubbornness) *Let p_i and p_j be two green processes, and $k > 0$ any constant. If p_i* stb-sends *messages $m_\psi, \ldots, m_{\psi+k-1}$ to p_j, and p_i indefinitely delays* stb-sending *any further message to p_j, then p_j eventually* stb-receives *messages $m_\psi, \ldots, m_{\psi+k-1}$.*

Properties 2.1 and 3.1 define a communication channel similar to the *non-duplication* channel of [1], which is a natural abstraction of the service provided by a connectionless network layer.

Property 3.2 slightly constrains the unreliability of the communication channels. It basically says that, if a green process p_i stb-sends a sequence $[m_\psi, \ldots, m_{\psi+k-1}]$ of messages to a green process p_j, and p_i afterwards is able to indefinitely delay the stb-sending of any subsequent message to p_j, then p_j eventually stb-receives

$[m_\psi, \ldots, m_{\psi+k-1}]$. Property 3.2 does not mean that, in order to ensure the reception of $[m_\psi, \ldots, m_{\psi+k-1}]$, process p_i is not allowed to stb-send any new message $m_{\psi+k}$ to p_j, after having stb-sent $m_{\psi+k-1}$. The property actually means that there is no fixed delay after which $m_{\psi+k}$ can be stb-sent without compromising the reception of m_ψ. This is abstractly expressed by the indefinite delay requirement.

Note that stubborn channels may lose an infinite number of messages (as long as the sender does not delay subsequent messages for enough long time) and may also reorder messages.

3.2 Transforming fair lossy into stubborn channels

3.2.1 Overview of the transformation

Figure 1 depicts a possible (not optimized) implementation of stubborn channels over fair lossy channels (defined by the *send* and *receive* primitives).

The transformation is given through two procedures *stb-send* and *stb-receive* and two concurrent tasks labeled *sender* and *receiver*. We consider (Section 4) that the state variables, the procedures and the concurrent tasks that form the transformation are part of the algorithm that uses them.

Basically, satisfying the k-stubbornness property is achieved by buffering and periodically retransmitting (using send) the k most recent messages stb-sent by p to q. Any new message that is stb-sent by p to q overwrites the oldest of the previous buffered k messages, which stops being retransmitted. Satisfying the no duplication property is achieved by pairing every message that is stb-sent with a strictly increasing sequence number. The sequence number of every received message is then compared with the smallest sequence number among the messages currently buffered.

Notice that the transformation into k-stubborn channels is restricted to the point-to-point retransmission of k messages. Transforming fair lossy channels into reliable channels would require broadcast communication, having each process p buffering all messages p sends, and still having p buffering and relaying all messages received by p but addressed to other processes [3].

3.2.2 Description of the transformation

Process p_i manages two buffers for each process p_j: $InBuf_i[j]$ and $OutBuf_i[j]$. Buffer $InBuf_i[j]$ (resp. $OutBuf_i[j]$) contains the k latest messages received from (resp. stb-sent to) p_j by p_i. The value $SN_i[j]$ denotes the sequence number of the latest message stb-sent by p_i to p_j. This sequence number is used to achieve the no duplication property (Property 3.1).

When p_i stb-sends a message m to p_j (line 5), m is paired with a strictly increasing sequence number $SN_i[j]$, and put in $OutBuf_i[j]$. If $OutBuf_i[j]$ is not full (i.e., does not yet contain k non-null messages) then m is added to $OutBuf_i[j]$. Otherwise m replaces the message with the smallest sequence number in $OutBuf_i[j]$. Periodically (task at line 17), each message in $OutBuf_i[j]$ is sent to p_j.

Upon the reception of a message (cnt, m) from p_j (line 23), p_i checks whether (1) (cnt, m) is currently in $InBuf_i[j]$, or (2) there is a message (cnt', m') in $InBuf_i[j]$ such that $cnt < cnt'$. In cases (1) or (2), (cnt, m) is discarded. Otherwise, (cnt, m) is added to $InBuf_i[j]$ and marked as *stb-unreceived* (i.e., (cnt, m) has not been yet stb-received at line 12). If $InBuf_i[j]$ is full, (cnt, m) replaces the message in $InBuf_i[j]$ with the smallest sequence number.

When p_i executes procedure stb-receive (m), some $stb - unreceived$ message in $InBuf_i[j]$ is assigned to m (the return parameter), and marked as *stb-received* in $InBuf_i[j]$. If no such message exists then $m = \lambda$.

```
variables
  OutBuf_i[n][k] ← [[λ, ... λ], ..., [λ, ... λ]];                { outgoing messages }
  InBuf_i[n][k] ← [[λ, ..., λ], ..., [λ, ... λ]];                 { incoming messages }
  SN_i[k] ← [0, ..., 0];                  { sequence number counters for outgoing messages }
procedure stb-send(m) to p_j                                      { stubborn send primitive }
  SN_i[j] ← SN_i[j] + 1;
  if |OutBuf_i[j]| < k then
    add (SN_i[j], m) to OutBuf_i[j];
  else
    replace the message in OutBuf_i[j] with the smallest sequence number by (SN_i[j], m);
end send ;
procedure stb-receive(m) from p_j                              { stubborn receive primitive }
  (cnt, m) ← some stb-unreceived message in InBuf_i[j];
  mark (cnt, m) in InBuf_i[j] as stb-received;
end receive ;
cobegin
|| task sender                                             { task executed by the sender }
periodically execute
  foreach j ∈ Σ
    foreach m ∈ OutBuf_i[j]
      send(m) to p_j;
|| task receiver                                         { task executed by the receiver }
  upon receive(cnt, m) from p_j :
    smallestSN ← smallest sequence number in InBuf_i[j];
    if (cnt, m) ∉ InBuf_i[j] and cnt > smallestSN then
      if |InBuf_i[j]| < k then
        add (cnt, m) to InBuf_i[j] and mark it stb-unreceived;
      else
        replace the message in InBuf_i[j] with smallest sequence number, by (cnt, m) mark it stb-unreceived;
coend
```

Figure 1: Transforming Fair Lossy into Stubborn channels (code for process p_i)

3.2.3 Correctness of the transformation

Theorem 3.3 *The algorithm of Figure 1 transforms a fair lossy channel into a stubborn channel.*

PROOF: The channels defined by the primitives *stb-send* and *stb-receive* of Figure 1 satisfy the no creation, no duplication and stubbornness properties.

No Creation (Property 2.1): Follows directly from the algorithm (Figure 1) and from the no creation property of the fair lossy channels.

No Duplication (Property 3.1): Assume by contradiction that some message m stb-sent by a process p_j is stb-received twice by a process p_i. Message m can only be stb-received twice if (c, m) is received twice, and put twice in $InBuf_i[j]$ at line 27 or 29 (and marked *stb-unreceived*). Consider the second time (c, m) is received, and let this time be denoted by t_{sec}. If at time t_{sec} the message (c, m) is in $InBuf_i[j]$ (marked as *stb-received* or *stb-unreceived*), then (c, m) is discarded. If at time t_{sec} the message (c, m) is not in $InBuf_i[j]$ then there is a time $t_{repl} < t_{sec}$ at which (c, m) has been replaced in $InBuf_i[j]$ by some message (c', m') with a sequence number $c' > c$. Moreover, at time t_{repl} all messages (c'', m'') is $InBuf_i[j]$ have a sequence number $c'' > c$. So, when (c, m) is received at time $t_{sec} > t_{repl}$, the condition of line 25 evaluates to false and (c, m) is discarded. A contradiction.

k-stubbornness (Property 3.2): Consider the n ($n \leq k$) most recent messages stb-sent by p_j to p_i, and assume that p_j stb-sends no further messages to p_i. Since these n messages have the n largest sequence numbers among all the messages stb-sent by p_j to p_i, these n messages are kept in $OutBuf_j[i]$ and are infinitely often sent to p_i over the fair lossy channel (lines 20 and 21). By Proposition 2.3 of the fair lossy channels, p_i eventually receives these n messages. As these n messages carry the n largest sequence numbers among all the messages stb-sent by p_j to p_i, each of these messages is eventually put in $InBuf_i[j]$, marked as *stb-unreceived* and never discarded. Thus these n messages are eventually stb-received by p_i. □

4 Solving consensus in the g/r-model

This section describes our consensus algorithm. We first rephrase the consensus problem in terms of green and red processes, then we describe our algorithm and we prove its correctness.

4.1 The Consensus problem

We define the consensus problem in the g/r-model over a set $\Pi \subset \Sigma$ of processes. Every process $p \in \Pi$ starts with an initial value v_p, and the processes of Π have to decide on a common value v, such that the following properties are satisfied [2]:

Uniform Validity. A process decides v only if v is the initial value of some process.

Uniform Agreement. No two processes decide differently.

Termination. Every green process eventually decides.

4.2 The consensus algorithm

The algorithm of Figure 2 guarantees the safety properties of consensus, (i.e., agreement and validity), no matter how many processes are red, and how the communication channels and the red detector behave. The liveness property of consensus (i.e., termination) is guaranteed if, every pair of processes is connected through 2-stubborn communication channels, a majority of the processes are green, and the red detector is of class $\diamond\mathcal{S}_r$.

[2] More accurately, we consider here the *uniform consensus* problem [9].

1 **function** consensus (v_i) { algorithm for a process p_i }
2 $r_i \leftarrow 0$; $estimate_i \leftarrow (i, v_i)$ { current round and estimate: $(estimate.first = i,\ estimate.second = v_i)$ }
3 $phase_i$; $coord_i$; { phase (1 or 2) and coordinator of $round_i$ }
4 $currentRoundTerminated_i$; $coordSuspected_i$; $msgCounter_i$; $suspCounter_i$;
5 **cobegin**
6 || **upon reception of** $(p_j, r_j, v_j, decide)$ from p_j;
7 send $(p_i, r_j, v_j, decide)$ to all;
8 **return** v_j;
9 || **loop**
10 $phase_i \leftarrow 1$; $currentRoundTerminated_i \leftarrow false$; $coordSuspected_i \leftarrow false$;
11 $coord_i \leftarrow (r_i \textbf{ mod } n) + 1$;
12 **if** $i = coord_i$ **then** send $(p_i, r_i, 1, estimate_i)$ to all; { p_i is the coordinator for the current round }
13 **while not** $currentRoundTerminated_i$
14 **select** { select one of the branches starting at lines 15,22,25,30,38 }
15 **upon reception of** $(p_j, r_i, 1, estimate_j)$ from p_j **when** $phase_i = 1$:
16 **first reception** : $msgCounter_i \leftarrow 1$;
17 **if** $i \neq coord_i$ **then** $estimate_i \leftarrow estimate_j$;
18 send $(p_i, r_i, 1, estimate_i)$ to all;
19 **other receptions** : $msgCounter_i \leftarrow msgCounter_i + 1$;
20 **if** $msgCounter_i = \lceil \frac{(n+1)}{2} \rceil$ **then** send $(p_i, r_i, 1, estimate_i.second, decide)$ to all;
21 **return** $estimate_i.second$;
22 **upon** $coord_i \in \mathcal{D}_{r_i}$ **when not** $coordSuspected_i$ **and** $phase_i = 1$:
23 send $(p_i, r_i, 1, suspicion)$ to all;
24 $coordSuspected_i \leftarrow true$;
25 **upon reception of** $(p_j, r_i, 1, suspicion)$ from p_j :
26 **first reception** : $suspCounter_i \leftarrow 1$;
27 **other receptions** : $suspCounter_i \leftarrow suspCounter_i + 1$;
28 **if** $suspCounter_i = \lceil \frac{(n+1)}{2} \rceil$ **then** $phase_i \leftarrow 2$;
29 send $(p_i, r_i, 2, estimate_i)$ to all;
30 **upon reception of** $(p_j, r_i, 2, estimate_j)$ from p_j :
31 **first reception** : $msgCounter_i \leftarrow 1$;
32 **if** $phase_i = 1$ **then** $phase_i \leftarrow 2$;
33 send $(p_i, r_i, 2, estimate_i)$ to all;
34 **other receptions** : $msgCounter_i \leftarrow msgCounter_i + 1$;
35 **if** $estimate_j.first = coord_i$ **then** $estimate_i \leftarrow estimate_j$;
36 **if** $msgCounter_i = \lceil \frac{(n+1)}{2} \rceil$ **then** $currentRoundTerminated_i \leftarrow true$;
37 $estimate_i.first \leftarrow i$; $r_i \leftarrow r_i + 1$;
38 **upon reception of** $(p_j, r_j > r_i, phase_j, estimate_j)$ from p_j :
39 $phase_i \leftarrow 1$; $r_i \leftarrow r_j$; $coordSuspected_i \leftarrow false$; $coord_i \leftarrow (r_i \textbf{ mod } n) + 1$;
40 $estimate_i \leftarrow estimate_j$; { adopt the "caller's" estimate }
41 **if** $phase_j = 1$ **then goto** 16;
42 **else goto** 31; { $phase = 2$ }
43 **end select**
44 **end while**
45 **end loop**
46 **coend**

Figure 2: Consensus with 2-stubborn channels and $\diamond\mathcal{S}_r$ (code for process p_i)

Our consensus algorithm is an extension of Schiper's algorithm [12]. We will discuss in Section 5 the reason why we have chosen to extend Schiper's algorithm, rather than the original algorithm of Chandra and Toueg [5].

The only change to Schiper's algorithm is the addition of lines 38 to 42 to overcome the omission failures of the communication channels. The reader familiar with [12] may directly proceed to Section 4.2.3 which discusses lines 38 to 42.

4.2.1 Overview of the algorithm

The consensus algorithm is based on the rotating coordinator paradigm and proceeds in asynchronous rounds. Every process p_i manages a variable $estimate_i$, which is p_i's current estimation of the decision value, and a variable r_i representing p_i's current round number. In every round, there is one process that plays the role of the coordinator, and this process is known a priori to all processes, e.g., in round 0 the coordinator is p_1, in round 1 the coordinator is p_2, etc. In every round r, the coordinator p_c tries to impose its $estimate_c$ as the decision value, by sending $estimate_c$ to all processes. When a process p_i receives $estimate_c$ from the coordinator, p_i forwards $estimate_c$ to all processes. A process decides on $estimate_c$ as soon as it has received $estimate_c$ from a majority of processes.

The protocol terminates in the first round if the first coordinator, p_1, is not suspected. Otherwise (i.e., if p_1 is suspected), the processes proceed to the second round, and so on. Before proceeding from a round to another, the estimates of the processes are updated, so as to satisfy the following invariant:

> *If some process has decided on $estimate_c$ in round r, then any process that proceeds to round $r+1$, starts round $r+1$ with $estimate_c$ as its current estimate.*

This invariant ensures the uniform agreement property of consensus: if some process decides $estimate_c$ in round r, then in any round $r' > r$, no decision can be different from $estimate_c$.

4.2.2 Description of the algorithm

Every process p_i invokes the function $consensus()$ of Figure 2, with its initial value v_i as a parameter. The function terminates when p_i executes the instruction **return** (either at line 8, or at line 21): we say that p_i decides on a value v exactly when p_i executes **return** v. Function $consensus()$ uses the stubborn communication primitives given in Figure 1, and runs concurrently with the two *sender* and *receiver* tasks [3]. The function *consensus* consists of two concurrent tasks: the first task executes lines 6-8, while the second task executes lines 9-45.

The first task handles the reception of the decision message $(p_j, r_j, v_j, decide)$ (line 6), and reissues the message to all (line 7) before returning. The fact that this *decision* message is the last message sent by p_i, together with the 2-stubbornness property[4] of the communication channels (Property 3.2), ensures that if a green process decides, then every green process eventually decides.

The second task is the central part of the algorithm. Every message sent by a process p_i includes the current round number r_i of p_i (e.g., line 12), and every message received by p_i at round r_i carries a round number equal or larger than r_i (e.g., lines 15 and 38). Each round of the consensus algorithm is divided in two phases, numbered 1 and 2:

- in phase 1 of every round r, the *consensus* algorithm tries to decide on the estimate value of the coordinator p_c of round r.

- if the coordinator of round r is suspected to be red, then phase 2 of round

[3]Note that in the algorithm we use the primitives *send* and *receive* instead of the defined stubborn primitives $stb-send$ and $stb-receive$. Since hereafter we will only use stubborn channels, this substitution does not arise any ambiguity and helps both the algorithmic and textual presentations.

[4]$k = 1$ would be sufficient here.

r is used to define a new *consensus try*, to be performed in round $r+1$. The initial value of process p_i for the *consensus try* of round $r+1$ is the estimate of p_i at the end of phase 2 of round r.

Phase 1 (lines 10-21). At line 12, the coordinator sends its current estimate to all processes (message $(p_i, r_i, 1, estimate_i)$). The estimate is a pair (*process number, initial value*) (see line 2) where *process number* indicates the identity of the process proposing that estimate. The third field in the message $(p_i, r_i, 1, estimate_i)$ indicates that the message is sent during phase 1. The message $(p_i, r_i, 1, estimate_i)$ is received by a process p_j[5] at line 15: p_j adopts (at line 17) the estimate received from p_i, and p_j forwards (at line 18) this estimate to all. Note that lines 17 and 18 are not performed by the coordinator, because it does not need to adopt its own estimate, and it has already sent its estimate to all.

Process p_i decides at line 21 on $estimate_i.second$, as soon as p_i has received $(p_j, r_i, 1, estimate_i)$ from a majority of processes. At line 20, process p_i sends its decision to all and does not send any further message. This ensures, along with the 2-stubbornness property of the communication channels (Property 3.2) that, if p_i is a green process, then every green process eventually also decides.

From phase 1 to phase 2 (lines 22-29). If no process suspects the coordinator in phase 1, then the decision value is the estimate of the coordinator. If a process p_i suspects the coordinator at line 22 (notation: $coord_i \in \mathcal{D}_{r_i}$), then p_i sends $(p_i, r_i, 1, suspicion)$ to all (line 23), indicating that p_i suspects the coordinator of round r_i. Once a process p_i knows that the coordinator is suspected by a majority of processes, then p_i proceeds to phase 2 (line 28). Moreover, upon proceeding to phase 2, p_i sends $(p_i, r_i, 2, estimate_i)$ to all (line 29). The reception of this message at line 30 forces a process to move to phase 2 (line 32). Upon

[5]A message sent by p_i to all is also received by p_i.

proceeding to phase 2, every process p_i similarly sends $(p_i, r_i, 2, estimate_i)$ to all (line 33). The condition "$phase_i = 1$", at line 34, prevents a process that has already sent $(p_i, r_i, 2, estimate_i)$ to all, at line 29, from sending this message again.

Phase 2 (lines 30-37). In phase 2, process p_i receives messages $(p_j, r_i, 2, estimate_j)$ (line 30). Upon each reception of such a message, p_i adopts the $estimate_j$ value, if and only if $estimate_j.first = coord_i$ (line 35), i.e., if and only if the estimate is that of the coordinator of the current round. Once p_i has received the message $(p_j, r_i, 2, estimate_j)$ from a majority of processes, p_i can switch to phase 1 of the next round (lines 36 and 37).

4.2.3 Handling messages from larger rounds

The relevant additions to the algorithm of Schiper [12] are at lines 38 to 42, where the reception of messages from larger rounds is handled. If p_i, at any phase of round r_i, receives a message $(p_j, r_j, phase_j, estimate_j)$ from a process p_j in a round $r_j > r_i$ (line 38), p_i adopts $estimate_j$ as its own estimate and directly joins p_j in round r_j (lines 39 to 42). In round r_j, process p_i carries on its execution where p_i would have received the $(p_j, r_j, phase_j, estimate_j)$ message in round r_j (i.e., at lines 16 or 31).

4.3 The proofs

We prove the correctness of the algorithm of Figure 2 under the assumption of a majority of green processes and a red detector of class $\diamond\mathcal{S}_r$.

In this section, we say that a process p_i *reaches round* ρ, when the current round r_i of p_i is such that $r_i \geq \rho$, and we say that p_i *reaches phase 2 of round* ρ if either (1) $r_i = \rho$ and $phase_i = 2$, or (2) $r_i > \rho$.

4.3.1 Preliminary lemmas

Our proof structure follows that of [12]. We first introduce five lemmas that are related to the termination property of consensus, then we introduce three other lemmas that are related to the validity and agreement properties[6].

Lemma 4.1 (Termination-1) *If one green process decides, then every green process eventually decides.*

PROOF: Let p_i be a green process that decides. Process p_i can do so either at line 8 or at line 21. In both cases, p_i sends the decision to all (message $(p_i, r_j, v_j, decide)$ sent at line 7, message $(p_i, r_i, estimate_i.second, decide)$ sent at line 20) as p_i's last message before returning. By the 2-stubbornness property (Property 3.2) of the communication channels, each green process that has not yet decided, eventually receives $(p_j, r_j, v_j, decide)$ (line 6), and also decides. □

Lemma 4.2 (Termination-2) *Let r be any round such that $r > 0$. The first process to reach round r leaves round $r-1$ at line 37.*

PROOF: Assume that p_i is the first process to reach round r. To reach round r, either (1) p_i executes line 37 of round $r-1$, or (2) in a round smaller than r process p_i receives some estimate message at line 38 from some process at round r. Case (2) is in contradiction with the assumption that p_i is the first process to reach round r. □

Lemma 4.3 (Termination-3) *There is a round ρ^* such that no process reaches any round $r > \rho^*$.*

[6] The reader familiar with [12] may notice that Lemmas 4.1, 4.5, 4.6, 4.7, and 4.8 match respectively Lemmas 4.1, 4.2, 4.3, 4.4, and 4.5 in [12].

PROOF: By the *eventual weak accuracy* property of $\Diamond\mathcal{S}_r$, there is a time t after which some green process p_c is no longer suspected by any green process. Let r be the largest round that some process has reached at time t, and $\rho^* \geq r$ the smallest round in which p_c is the coordinator. The proof is by induction on the round number r.

Base step: No process reaches round $\rho^* + 1$.
The proof is by contradiction. Let p_i be the first process to reach round $\rho^* + 1$. By Lemma 4.2, process p_i can only proceed to round $\rho^* + 1$ if p_i executes line 37 at round ρ^*. To do so, p_i must have received a majority of $(p_j, \rho^*, 2, estimate_j)$ at line 30. These $(p_j, \rho^*, 2, estimate_j)$ messages can only be sent if some process has received a majority of $(p_j, \rho^*, 1, suspicion)$ messages at line 25, which in turn implies that a majority of processes suspected p_c at line 22 in round ρ^*. However round ρ^* happens after t and, by assumption, a majority of processes are green and no green process suspects p_c after time t: a contradiction.

Induction step: If no process reaches some round $r > \rho^*$, then no process reaches round $r + 1$.
By Lemma 4.2, the first process to reach round $r + 1$ leaves round r at line 37. By the induction hypothesis, no process reaches round r, and thus no process reaches round $r + 1$.

□

Lemma 4.4 (Termination-4) *For any to consecutive messages m and m' that a process p sends to q, m and m' cannot both be suspicion messages.*

PROOF: A process p_i can only send a suspicion message once per round (line 23) because once p_i sends the suspicion message, $coordSuspected_i$ becomes true (line 24) and can only be assigned false if p_i enters another round (lines 10 or 39).

Consider that process p_i sends $(p_i, \rho, 1, suspicion)$ at line 23 of round ρ. Process p_i can only send another suspicion message if p_i reaches some round $\rho' > \rho$. We prove that before sending another suspicion message, p_i needs firstly to send some estimate message. Process p_i can reach round ρ' either 1) at line 37 (in which case $\rho' = \rho + 1$), or 2) at line 39.

In case 1, p_i reached phase 2 of round ρ either at line 28, or at line 32. In both cases, p_i sends a $(p_i, \rho, 2, estimate_i)$ message (line 29 or 33).

In case 2, p_i receives a $(p_j, \rho', phase_j, estimate_i)$ message at line 38. If $phase_j = 1$ then p_i moves to line 16 and round ρ', and sends a $(p_i, \rho', 1, estimate_i)$ at line 18 since p_i cannot be the coordinator of round ρ'. If $phase_j = 2$ then p_i moves to line 31 of round ρ', and sends a $(p_i, \rho', 2, estimate_i)$ at line 33 since p_i has $phase_i = 1$. □

Lemma 4.5 (Termination-5) *For any round ρ, if no green process decides in a round $r \leq \rho$, then every green process eventually reaches round $\rho + 1$.*

Proof: Let ρ be the smallest round for which the lemma does not hold: no green process decides in round ρ, and some green process never reaches round $\rho + 1$. As ρ is the smallest of such rounds, each green process eventually reaches round ρ. We will contradict the hypothesis by proving the following successive results:

i) At least one green process eventually reaches phase 2 of round ρ.

ii) Each green process eventually reaches phase 2 of round ρ.

iii) Each green process eventually reaches round $\rho + 1$.

Proof of (i): Assume that no green process decides in round ρ, and no green process reaches phase 2 of round ρ. We consider two cases: the coordinator of round ρ, process p_c, 1) is a green process, or 2) is a red process.

In case 1, process p_c is a green process and sends $(p_c, \rho, 1, estimate_c)$ to all at line 12. As by hypothesis, p_c does not reach phase 2 (and does no suspect itself), this message is the last message p_c ever sends. By the 2-stubbornness property of the communications channels all green processes eventually receive p_c's message at line 15. All green processes (except p_c) send then a $(p_i, \rho, 1, estimate_i)$ message to all at line 18. Since no green process leaves phase 1 of round ρ, each green process can only send one further message (possibly a $(p_i, \rho, 1, suspicion)$ message at line 23) after sending the $(p_i, \rho, 1, estimate_i)$ message. By the 2-stubbornness property of the communication channels all green processes eventually receive a majority of $(p_j, \rho, 1, estimate_j)$ messages at line 15. Thus the condition $msgCounter = \lceil \frac{(n+1)}{2} \rceil$ of line 20 eventually becomes true for every green process, i.e., every green process eventually decides in round ρ: a contradiction with the fact that no green process decides.

In case 2, process p_c is a red process. By the eventual strong completeness property of $\Diamond\mathcal{S}_r$, p_c is eventually suspected by every green process. Every green process that suspects p_c sends a $(p_i, \rho, 1, suspicion)$ message to all at line 23. As by hypothesis, no green process reaches phase 2, all green processes eventually receive a majority of $(p_j, \rho, 1, suspicion)$ messages at line 25 and reach phase 2: a contradiction with the fact that no process reaches phase 2 of round ρ.

Proof of (ii): By (i), at least one green process, say p_k, eventually reaches phase 2 of round ρ. Process p_k sends $(p_k, \rho, 2, estimate_k)$ to all at line 29. This is the last message p_k may send in round ρ.

Assume that there is some green process p_l that does not leave phase 1 of round ρ. We consider two cases: 1) $(p_k, \rho, 2, estimate_k)$ is the last message p_k ever sends, or 2) p_k sends some further estimate message in a round $\rho' > \rho$.

In case 1, by the stubbornness property of the communication channels, process p_l eventually receives the $(p_k, \rho, 2, estimate_k)$ message at line 30, and reaches phase 2 at line 32.

In case 2, by Lemma 4.3, no process reaches any round larger than ρ^*,and hence p_k's last message must be sent in a round $\rho < \rho' \leq \rho^*$. As a process only sends suspicion messages or estimate messages, by Lemma 4.4, one of the two last messages of p_k is an estimate message, i.e., a $(p_k, \rho', -, estimate_k)$ message. By the 2-stubbornness property of the communication channels, p_l eventually receives this message at line 38 and moves to round ρ', thus leaving phase 2 of round ρ.

Both cases contradict the fact that there is some green process that does not leave phase 1 of round ρ.

Proof of (iii): By (ii) all green processes reach phase 2 of round ρ. In case 2 of (ii), all green processes eventually reach some round $\rho < \rho' \leq \rho^*$ and so reach round $\rho+1$. Therefore we need only to prove that all green processes leave round ρ when $(p_i, \rho, 2, estimate_i)$ is the last estimate message any green process ever sends.

As all green processes reach phase 2 of round ρ and send a $(p_i, \rho, 2, estimate_i)$ message either at line 29 or at line 33, as their last message. By the 2-stubbornness property of the communication channels, all green processes eventually receive a majority of such messages at line 30. Then, condition $msgCounter = \lceil \frac{(n+1)}{2} \rceil$ of line 36 becomes true and each green process proceeds to round $\rho + 1$. □

Lemma 4.6 (Validity) *All messages $(p_i, \rho, 1, estimate_i)$ sent during phase 1 of round ρ carry the $estimate_c$ value of the coordinator p_c of round ρ.*

PROOF: The proof is by induction on the length of the *send-receive* chain of messages $(p_i, \rho, 1, estimate_i)$. The messages $(p_i, \rho, 1, estimate_i)$ are numbered as follows:

- the message $(p_i, \rho, 1, estimate_i)$ sent by the coordinator at line 12 is numbered 0;

- if the message $(p_j, \rho, 1, estimate_j)$ received by p_i at line 15 is numbered k, then the message $(p_i, \rho, 1, estimate_i)$ sent by p_i at line 18 is numbered $k+1$.

Base step. Trivially, for the message $(p_i, \rho, 1, estimate_i)$ number 0, $estimate_i$ is the estimate of the coordinator of round ρ.

Induction step. Consider a message $(p_i, \rho, 1, estimate_i)$ number $k+1$. This message is sent by some process p_i at line 18, after having received, at line 15, the message $(p_j, \rho, 1, estimate_j)$ number k. By induction hypothesis, this message carries the $estimate_c$ value of the coordinator of round ρ. Therefore, because of line 17, the message $(p_i, \rho, 1, estimate_i)$ also carries the $estimate_c$ value of the coordinator of round ρ. □

Lemma 4.7 (Agreement-1) *If a process (green or red) decides v in round ρ, then v is the "$estimate_c.second$" value of the coordinator p_c of round ρ.*

PROOF: A process p_j can decide in round ρ either at line 8, or at line 21. However, process p_j can decide at line 8 of round ρ if and only if there is a process p_i that has decided at line 23, as it is not possible for all processes that decide in round ρ to do so at line 8. Consider thus the decision of p_i at line 21. If p_i is the coordinator of round ρ, then $estimate_i.second$ is trivially the $estimate_c.second$ value of the coordinator of round ρ. Otherwise, by line 17, the decision is on the first estimate value received by p_i at line 15. By Lemma 4.6, the value received is the estimate of the coordinator of round ρ. □

Lemma 4.8 (Agreement-2) *If a process (green or red) decides v in round ρ, then every process p_i in any round $r > \rho$ has $estimate_i.second = v$.*

PROOF: If every process p_i that leaves round ρ does so with $estimate_i.second = v$ then the result holds for any $r > \rho$.

We prove by contradiction that every process p_i that leaves round ρ does so with $estimate_i.second = v$. Let p_i be the first process to leave round ρ such that

$estimate_i.second \neq v$. Process p_i cannot have left round ρ upon the reception of a message at line 38, because in this case p_i would have received a message from some process p_j in a round larger than ρ such that $estimate_j \neq v$, which would imply that some process had already left round ρ with $estimate \neq v$, a contradiction with the fact that p_i is the first of such processes. So p_i must have left round ρ at line 37.

A process p_j can decide in round ρ either at line 8, or at line 21. However, process p_j can decide at line 8 of round ρ if there is a process p_i that has decided at line 21, as it is not possible for all processes that decide in round ρ to do so at line 8. Consider thus the decision at line 21.

Let p_i be a process that decides v in round ρ at line 21. By Lemma 4.7, v is the $estimate_c.second$ value of the coordinator p_c of round ρ. Moreover, because of line 20, when p_i decides at line 21, a majority of processes in phase 1 have sent $(p_j, \rho, 1, estimate_j)$ to all. By Lemma 4.6, every estimate sent in phase 1 is the estimate of the coordinator p_c of round ρ. Thus when p_i decides at line 21, a majority of processes (including p_i itself) have their estimate equal to the estimate of p_c. Let us call this set $\mathsf{CoordEstimateSet}_\rho$: we have $|\mathsf{CoordEstimateSet}_\rho| > n/2$.

Consider now a process p_k that leaves round ρ at line 37. This is only possible after p_k has received the message $(p_j, \rho, 2, estimate_j)$ from a majority of processes, including from itself (line 36). Let us call this set $\mathsf{AuthorizationSet}_k$: we have $|\mathsf{AuthorizationSet}_k| > n/2$. Altogether we have $|\mathsf{AuthorizationSet}_k| > n/2$ and $|\mathsf{CoordEstimateSet}_\rho| > n/2$, therefore $\mathsf{AuthorizationSet}_k \cap \mathsf{CoordEstimateSet}_\rho \neq \emptyset$. This means that p_k receives the message $(p_j, \rho, 2, estimate_j)$ at line 30 from at least one process in $\mathsf{CoordEstimateSet}_\rho$, and at line 35, process p_k sets $estimate_k.second$ to v. Thus, when p_k proceeds to round $\rho + 1$, we have $estimate_k.second = v$. A contradiction. $\square$

4.3.2 Correctness proof of the consensus algorithm

We now prove, based on the previous lemmas, that the consensus algorithm of Figure 2 satisfies the Termination, Validity and Uniform Agreement properties.

Proposition 4.9 (Termination) *The consensus algorithm of Figure 2 satisfies the Termination property.*

PROOF: By the eventual weak accuracy property of $\diamond\mathcal{S}_r$, there is a time t after which some green process p_k is not suspected by any green process. Let ρ be a round such that (i) p_k is the coordinator of ρ, and (ii) every green process enters round ρ after t (if such a round does not exist, then by Lemma 4.5 one green process has decided in a round $\rho' < \rho$, and so, by Lemma 4.1, every green process decides, and the Termination property holds). As a majority of processes are green and no green process suspects p_k in round ρ, the condition $suspCounter_i = \lceil\frac{(n+1)}{2}\rceil$ at line 28 remains false forever for every p_i in round ρ. In round ρ, process p_k sends $(p_k, \rho, 1, estimate_k)$ to all (line 12). As p_k is a green process and p_k either decides or $(p_k, \rho, 1, estimate_k)$ is the last message p_k sends to all processes, by the 2-stubbornness property of communication channels, each green process eventually receives $(p_k, \rho, 1, estimate_k)$ (line 15). Moreover, at line 18, each green process sends $(p_i, \rho, 1, estimate_i)$ to all. As a majority of processes are green, and each green process p_i either decides or $(p_i, \rho, 1, estimate_i)$ is its last messages, by the 2-stubbornness property of communication channels, any green process that does not decide by receiving a "decide" message (line 6), will receive enough estimates to decide at line 21. □

Proposition 4.10 (Validity) *The consensus algorithm of Figure 2 satisfies the Validity property.*

PROOF: Assume by contradiction that Validity does not hold. Then some process p_i sets $estimate_i.second$ to a value that is not the proposal of any process. Let ρ be the smallest round in which this happens.

Case 1. Assume that this happens in phase 1 of round ρ, i.e., at line 17. By Lemma 4.6 every estimate sent in phase 1 of round ρ is the estimate of the coordinator p_c of round ρ. If $\rho = 0$, $estimate_c.second$ is the proposal of p_c. A contradiction. If $\rho > 0$, then $estimate_c.second$ is the estimate of p_c at the end of round $r < \rho$. As by hypothesis ρ is the earliest round in which some process p_i sets $estimate_i.second$ to a value that is not the proposal of some process, the value $estimate_c.second$ is the proposal of some process. A contradiction.

Case 2. Assume that this happens in phase 2 of round ρ: some process sets for the first time $estimate_i.second$ to a value that is not the proposal of some process in phase 2 of round ρ. This can only occur at line 35, where $estimate_j$ is the estimate received at line 30. Any estimate received at line 30 is sent by some process either at line 29, or at line 33. In both cases, the estimate sent is the estimate of some process in phase 1 of round ρ. Thus some process must have set, in phase 1, its estimate to a value that is not the proposal of some process, in contradiction with the result established by Case 1.

Case 3. Assume that this happens at line 38 (either in phases 1 or 2): some process p_i sets for the first time $estimate_i.second$ to a value that is not the proposal of some process. Process p_i must receive a $(p_j, \rho', phase_j, estimate_j)$ with $\rho' > \rho$ and such that $estimate_j.second$ is not the proposal of some process. Since ρ is the smallest round in which some process adopts such an estimate, some process must have adopted $estimate_j.second$ in round ρ either in case 1 or case 2 (above), which have been proven to be impossible. A contradiction. □

Proposition 4.11 (Uniform Agreement) *The consensus algorithm of Figure 2 satisfies the Uniform Agreement property.*

PROOF: Assume that a process p_i (green or red) decides v in round ρ. We prove that no other process p_j can decide on a different value.

Assume that p_j decides in round ρ'. If $\rho' = \rho$, by Lemma 4.7, p_i and p_j both decide on the $estimate_c.second$ value of the coordinator p_c of round ρ, i.e., on the same value.

Consider the case $\rho' > \rho$ (if $\rho' < \rho$, rename p_i to p_j, and p_j to p_i). By Lemma 4.8, every process p_i that begins round $\rho' > \rho$, does so with $estimate_i.second = v$. By Lemma 4.7, the value decided by p_j in round ρ' is also v. $\square$

Propositions 4.9, 4.10, and 4.11 prove that the algorithm of Figure 2 solves consensus in the g/r-model.

5 The weakest red detector for solving consensus

We prove in this section that, as in the crash-stop model $\Diamond\mathcal{S}$ is the weakest class of failure detectors for solving consensus [4], in the crash-recover model $\Diamond\mathcal{S}_r$ is the weakest class of red detectors for solving consensus. We show however that, unlike in the crash-stop model, the class $\Diamond\mathcal{W}_r$ is strictly weaker than the class $\Diamond\mathcal{S}_r$ (in the crash-stop model, the classes $\Diamond\mathcal{S}$ and $\Diamond\mathcal{W}$ are equivalent). Hence, $\Diamond\mathcal{W}_r$ does not solve consensus (i.e., $\Diamond\mathcal{W}_r$ is too weak to solve consensus). Informally, the proof of equivalence between $\Diamond\mathcal{S}$ and $\Diamond\mathcal{W}$ in the crash-stop model uses the fact that eventually all faulty processes crash, and therefore stop sending and receiving messages. In the g/r-model, this is no longer true: at any time, red processes might send and receive messages.

5.1 Definitions

We first introduce some definitions that are patterned after the model of [4], and that are required in the following sections.

Failure patterns Processes can fail by crashing. After a crash processes may or may not recover. We assume a discrete global clock to which processes do not have access[7], and Φ, the range of the clock ticks, is the set of natural

[7] This global clock is a fictional device helping the presentation of our model.

numbers. A *failure pattern* is a function $F : \Phi \to 2^{\Sigma}$, where $F(t)$ denotes the set of crashed processes at time t. We say that a process p *is green in* F if $\exists t, \forall t' \geq t : p \notin F(t')$, and that a process p *is red in* F if p is not green in F. We assume that in any failure pattern, there is at least a green process.

Red detector histories and classes. A *red detector history* is a function $R : \Sigma \times \Phi \to 2^{\Sigma}$, where $R(p, t)$ denotes the set of processes suspected to be red by process p at time t. A *red detector* is a function $\mathcal{D}$ that maps each failure pattern F to a set of red detector histories. A red detector class is a set of red detectors. We say that a red detector class $\mathcal{D}_r$ *solves* a problem P, if there is an algorithm that solves P with any red detector of class $\mathcal{D}_r$. Given two red detector classes $\mathcal{D}_r$ and $\mathcal{D}'_r$, $\mathcal{D}_r$ is said to be *weaker than* $\mathcal{D}'_r$, if there is an algorithm that transforms any red detector of class $\mathcal{D}_r$ into some red detector of class $\mathcal{D}'_r$; $\mathcal{D}_r$ and $\mathcal{D}'_r$ are said to be *equivalent* if $\mathcal{D}_r$ is weaker than $\mathcal{D}'_r$ and $\mathcal{D}_r$ is weaker than $\mathcal{D}'_r$; $\mathcal{D}_r$ is said to be *strictly weaker* than $\mathcal{D}'_r$ if $\mathcal{D}_r$ is weaker than $\mathcal{D}'_r$, but $\mathcal{D}_r$ and $\mathcal{D}'_r$ are not equivalent.

Configurations, schedules and runs. A *configuration* of the system is a pair (I, M) where I is a function mapping each process p to its local state, and M is the set of messages sent by every process. A configuration (I, M) is an initial configuration if $M = \emptyset$. A non null step is a tuple $s = (p, m, d, A)$ uniquely defined by the identity of the process p that takes the step, the message m received by p, the *red* detector value d during the step, and the algorithm A. A non null step $s = (p, m, d, A)$ is *applicable to a configuration* (I, M) if and only if $m \in M \bigcup \{\lambda\}$. The *unique* configuration that results from applying $s = (p, m, d, A)$ to $C = (I, M)$ is noted $s(C)$. A null step $s = \lambda$ is applicable to any configuration $C = (I, M)$, and $s(C) = C$.

A *schedule* of an algorithm A is a (possibly finite) sequence of steps of A and null steps, noted $S = S[1]; S[2]; \ldots S[k]; \ldots$. A schedule S is applicable to a configuration C if (1) S is the empty schedule, or (2) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$, etc.

A *partial run* of A using a red detector $\mathcal{D}_r$, is a tuple $\Upsilon =< F, R, C, S, T >$ where, F is a failure pattern, R is a red detector history such that $R \in \mathcal{D}_r(F)$, C is an initial configuration of A, T is a finite sequence of increasing time values, and S is a finite schedule of A, such that: (1) $|S| = |T|$, (2) S is applicable to C, (3) for all $i \leq |S|$ where $S[i] = (p, m, d, A)$, we have $p \notin F(T[i])$ and $d = R(p, T[i])$, and (4) for all $i \leq |S|$ where $S[i] = \lambda$, we have $p \in F(T[i])$.

A *run* of an algorithm A using a red detector $\mathcal{D}_r$, is a tuple $\Upsilon =< F, R, C, S, T >$ where F is a failure pattern, R is a red detector history such that $R \in \mathcal{D}_r(F)$, C is an initial configuration of A, S is an infinite schedule of A, T is an infinite sequence of increasing time values, and in addition to the conditions of a partial run ((1), (2) and (3) above), the following condition is satisfied: (4) messages that are sent are received according to the fairness proposition (Proposition 2.3) of the communication channels.

Let $\Upsilon =< F, R, C, S, T >$ be a partial run of some algorithm A. We say that $\Upsilon' =< F', R', C', S', T' >$ is *an extension of* Υ, if Υ' is either a run or a partial run of A, and $F' = F$, $R' = R$, $C' = C$, $\forall i$, $T[1] \leq i \leq T[|T|]$: $S'[i] = S[i] \wedge T'[i] = T[i]$.

5.2 $\diamond\mathcal{S}_r$ is the weakest class for solving consensus

This result follows that of Chandra, Hadzilacos and Toueg [4], which states that, in the crash-stop model, $\diamond\mathcal{S}$ is the weakest class for solving consensus.

Proposition 5.1 *$\diamond\mathcal{S}_r$ is the weakest class for solving consensus in the g/r-model, with a majority of green processes and fair lossy channels.*

PROOF: The proof is by contradiction. Assume that there is a red detector class, $\mathcal{D}_r$ strictly weaker than $\diamond\mathcal{S}_r$, which allows for solving consensus in the g/r-model. The crash-stop model satisfies the definitions of the g/r-model, and thus $\mathcal{D}_r$ allows for solving consensus in the crash-stop model which is in contradiction with the result of [4] (i.e., $\diamond\mathcal{S}$ is weakest class that allows for solving consensus). $\square$

5.3 $\diamond\mathcal{W}_r$ is not sufficient for solving consensus

In the following, we show that in a system with at least three processes, $\diamond\mathcal{S}_r$ is strictly stronger than $\diamond\mathcal{W}_r$ [8]. As $\diamond\mathcal{S}_r$ is the weakest class that solves consensus (Proposition 5.1), then, in the g/r-model, $\diamond\mathcal{W}_r$ does not solve consensus.

As $\diamond\mathcal{W}_r$ is obviously weaker than $\diamond\mathcal{S}_r$, showing that $\diamond\mathcal{W}_r$ is strictly weaker than $\diamond\mathcal{S}_r$ comes down to show that there is a red detector of class $\diamond\mathcal{W}_r$, that no algorithm can transform into a red detector of class $\diamond\mathcal{S}_r$. In the following, we define a specific red detector, that we note $\mathcal{D}_r(p_i,p_j)$, of class $\diamond\mathcal{W}_r$, and we prove that if an algorithm $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ transforms $\mathcal{D}_r(p_i,p_j)$ into some red detector Δ that satisfies strong completeness, then Δ does not satisfy eventual weak accuracy (i.e., Δ cannot be of class $\diamond\mathcal{S}_r$). We note *output*(Δ) the variable used by $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ to simulate the behavior of Δ.

Red detector $\mathcal{D}_r(p_i,p_j)$. The red detector $\mathcal{D}_r(p_i,p_j)$ we use in our proof is defined using two processes p_i and p_j of Σ, and the notion of (i,j)-*pattern*: we say that a failure pattern F is a (i,j)-*pattern*, if either p_i or p_j is red in F (only one of them can be red). The red detector $\mathcal{D}_r(p_i,p_j)$ is specified as follows: (a) In any (i,j)-*pattern*, (a.1) p_i and p_j permanently suspect every process, and (a.2) $\forall p_k, p_k \neq p_i \wedge p_k \neq p_j$, p_k permanently suspects every other process except p_i and p_j; (b) In every run in which F is not a (i,j)-*pattern*, every red process is permanently suspected by every green process, and no green process is ever suspected by any green process.

[8] In a system with two processes, strong and weak completeness properties are similar, hence the two classes are obviously equivalent.

Lemma 5.2 *$\mathcal{D}_r(p_i, p_j)$ is of class $\diamond\mathcal{W}_r$.*

Proof: The red detector $\mathcal{D}_r(p_i, p_j)$ satisfies weak completeness: as (1.1) in any (i, j)-*pattern*, either p_i or p_j is green and permanently suspects every red process (since it permanently suspects every process), and (1.2) in any other pattern, every red process is permanently suspected by every green process. The failure detector $\mathcal{D}_r(p_i, p_j)$ also satisfies eventual weak accuracy as (2.1) in any (i, j)-*pattern*, either p_i or p_j is green and is not suspected by any green process, and (2.1) in any other pattern, no green process is suspected by any green process. Hence $\mathcal{D}_r(p_i, p_j)$ is of class $\diamond\mathcal{W}_r$. □

Lemma 5.3 *Consider a system with at least three processes among which, one may be red. Let $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ be any algorithm that transforms $\mathcal{D}_r(p_i, p_j)$ into some red detector Δ. Let $\Upsilon =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S, T >$ be any partial run of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ where F is a (i, j)-*pattern *in which all processes of Σ, except either p_i or p_j, are green. If Δ satisfies strong completeness, then there is an extension $\Upsilon_\Sigma =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S_\Sigma, T_\Sigma >$ of Υ, where for every green process $p \in \Sigma$, there is a time t, $T[|T|] \leq t \leq T_\Sigma[|T_\Sigma|]$, such that for every green process $q \in \Sigma$, $p \in output(\Delta, t)_q$ in Υ_Σ.*

Proof: Consider the partial run $\Upsilon =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S, T >$ of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ where F is a (i, j)-*pattern* in which all processes, except either p_i or p_j, are green. Let p be any process in Σ. Let $\Upsilon' =< F', R_{\mathcal{D}_r(p_i,p_j)}, C, S, T >$ be a partial run of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ such that F' is a (i, j)-*pattern* similar to F, except that in F', p is red and behaves like in F until at least time $T[|T|]$). As $\mathcal{D}_r(p_i, p_j)$ provides the same values both for F and F', Υ' is a partial run of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$. By the strong completeness property of Δ, there is an extension $\Upsilon'_p =< F', R_{\mathcal{D}_r(p_i,p_j)}, C, S_p, T_p >$ of Υ', such that $p \in output(\Delta, T_p[|T_p|])_q$ for every green process $q \in \Sigma$. Let $S_{Susp(p)}$ be the schedule of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ such that $S_p(C) = S_{Susp(p)}(S(C))$. The schedule $S_{Susp(p)}$ can be viewed as the schedule needed to put p into the $output(\Delta)_q$ of every green process q.

Consider now the partial run $\Upsilon =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S, T >$. As $\mathcal{D}_r(p_i,p_j)$ provides the same values both for F and F', and S_p is applicable to C, then $\Upsilon_p =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S_p, T_p >$ is an extension of Υ, and for every green process q, $p \in output(\Delta, T_p[|T_p|])_q$. By iteratively applying the construction of the partial run Υ_p to every process $p \in \Sigma$, the partial run Υ can be extended to a partial run $\Upsilon_\Sigma =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S_\Sigma, T_\Sigma >$ where *every* process p is put in $output(\Delta)_q$ for every green process q. $\square$

Lemma 5.4 *Consider a system with at least three processes among which one may be red. Let $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$ be any algorithm that transforms $\mathcal{D}_r(p_i,p_j)$ into some red detector Δ. If Δ satisfies strong completeness, then there is a run of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$, where Δ does not satisfy eventual weak accuracy.*

PROOF: Consider the partial run $\Upsilon =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S, T >$ where F is a (i,j)-*pattern* in which all processes, except either p_i or p_j, are green. By Lemma 5.3, there is an extension $\Upsilon_\Sigma =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S_\Sigma, T_\Sigma >$ of Υ, where for every green process $p \in \Sigma$, there is a time t, $T[|T|] \leq t \leq T_\Sigma[|T_\Sigma|]$, such that for every green process $q \in \Sigma$, $p \in output(\Delta, t)_q$ in Υ_Σ.

Let (I, M) be the configuration $S_\Sigma(C)$. Consider now a schedule S_{Msg} defined by: the reception, according to Proposition 2.3, of all messages in M not received in S_Σ , then the reception by every green process of the null message λ. The schedule S_{Msg} is by construction applicable to $S(C)$, and we write $S_\Omega(C) = S_{Msg}(S_\Sigma(C))$. There is a sequence of increasing time values T_Ω, such that $\Upsilon_\Omega =< F, R_{\mathcal{D}_r(p_i,p_j)}, C, S_\Omega, T_\Omega >$ is an extension of Υ. In Υ_Ω, all messages sent to every green process before time $T[|T|]$ are received by p according to Proposition 2.3 before $T_\Omega[|T_\Omega|]$, and every green process takes at least one step between $T[|T|]$ and $T_\Omega[|T_\Omega|]$.

Therefore, given any partial run Υ of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$, where F is a (i,j)-*pattern* in which all processes, except either p_i or p_j, are green, we can extend Υ to a partial run Υ_Σ where every process is suspected at some green process by Δ.

We note $\Upsilon_\Sigma^0 = \Upsilon_\Sigma$, Υ_Σ^1 an extension of Υ obtained by applying the construction above to Υ_Σ^0, Υ_Σ^i an extension of Υ obtained by applying the construction above to Υ_Σ^{i-1}, etc., and $\Upsilon_\Sigma^\infty = lim_{i\to\infty}\Upsilon_\Sigma^i$. Υ_Σ^∞ does exist since red processes may take an infinite number of non null steps (even if red processes also take an infinite number of null steps in F).

In Υ_Σ^∞, the properties of a partial run are satisfied, and as every message sent to a process is eventually received, Proposition 2.3 holds, hence Υ_Σ^∞ is a run of $A_{\mathcal{D}_r(p_i,p_j)\to\Delta}$. Furthermore, for any time t and any process p, there is a time $t' \geq t$, such that $p \in output(\Delta, t')_q$ of every green process q. Hence Δ does not satisfy eventual weak accuracy in Υ_Σ^∞. □

Proposition 5.5 *In a system with at least three processes among which, one may be red, the red detector class $\diamond\mathcal{W}_r$ is strictly weaker than the red detector class $\diamond\mathcal{S}_r$.*

PROOF: By Lemma 5.2 and Lemma 5.4, no algorithm can transform any red detector of $\diamond\mathcal{W}_r$, into some red detector of $\diamond\mathcal{S}_r$. In other words, $\diamond\mathcal{S}_r$ is not weaker than $\diamond\mathcal{W}_r$. As $\diamond\mathcal{W}_r$ is weaker than $\diamond\mathcal{S}_r$, then $\diamond\mathcal{W}_r$ is strictly weaker than $\diamond\mathcal{S}_r$. □

6 Related work

In this section we compare our work with the following references [5, 12, 7, 6, 2, 10]. We start by a comparison of the system models. While our system model is an extension of that considered in [5, 12] and it is clearly more general, the differences to the models considered in [7, 6] are less obvious but fundamental. We compare then our consensus algorithm with those presented in [5, 12, 7, 10].

6.1 Model

In the model considered in [5, 12], processes that fail do so by crashing and, afterwards do not take any further step (crash-stop semantics). With respect to process failure semantics, our abstract g/r-model is more general as processes can crash-stop, and still crash-recover.

To our knowledge, only [7, 6] address the problem of solving consensus in an asynchronous system, tolerating processes crash-recovery failures and omission failures. In [7], the authors consider a crash-recover model with network partitions, where eventually (1) there is a *connected* majority of processes that never crash, and (2) this majority is *disconnected* from the rest of the processes. In [6], these two conditions appear in the *majority-stable* predicate formulated according to the *timed* asynchronous model's timeliness criterium.

While in our model assumption (1) exists as the requirement of a *majority of green processes*, we make no assumption (2) on the red processes inability to communicate (eg. with green processes). That is in the g/r-model, at any time, messages sent by green processes may be received by red processes, and messages sent by red processes may be received by green processes. As discussed in Section 5, it turns out that in the model of [7], $\diamond\mathcal{W}$ is the weakest failure detector for solving consensus while in the g/r-model $\diamond\mathcal{W}_r$ is too weak.

Our definition of the red detector class $\diamond\mathcal{S}_r$ generalizes the definition of failure detector class $\diamond\mathcal{S}$ [5]. While in $\diamond\mathcal{S}$ suspicions are specified in terms of *crashed* processes, in $\diamond\mathcal{S}_r$ suspicions are specified in terms of red processes. This generalization is similar to that found in [2], where the *crashed* processes is replaced by *faulty* process in order to encompass *crashed* and *unreachable* processes. Our generalization is a mere syntactic transformation, however.

6.2 Algorithm

As we mentioned in Section 4, our algorithm is an extension of Schiper's algorithm [12], which like the original algorithm of Chandra and Toueg [5], considers reliable channels and process crash-stop semantics. Beside the fact that Schiper's algorithm has an *early delivery* property, an important reason why we have chosen to extend that algorithm (instead of Chandra-Toueg's one), is that it has the property that a process p does not leave a round r for round $r+1$, unless p knows that a majority of processes suspect the coordinator in round r. This is an important property as red processes may always suspect the coordinator and keep sending their suspicions to all. The requirement of collecting a majority of suspicions in each round before leaving it, actually precludes red processes from indefinitely carrying all processes from one round to another. We could have added this property to the original Chandra-Toueg's algorithm, but this would have ended in a very complex algorithm.

Beside this difference, our consensus algorithm tolerates omission failures, and has the important practical advantage of having a memory requirement that increases logarithmically with the round number instead of linearly. Assuming reliable channels over a fair lossy channels requires each message to be buffered for retransmission until the message is acknowledge, which implies buffering k messages per round r. Using 2-stubborn channels instead, allows to reduce buffering to two messages at every time. In both cases, every message carries the round number in which it has been sent, and thus its storage requirement is $O(log_2 r)$. With the algorithms of [5, 12], the storage requirements are $O(k \cdot r \cdot log_2 r)$ while in ours (similarly to [10, 7]) it is $O(2 \cdot log_2 r)$.

7 Concluding remarks

By stating the conditions under which the consensus problem can be solved in terms of failure detector properties, Chandra, Hadzilacos and Toueg, defined a rigorous framework for reliable distributed computing [4, 5]. However, the assumptions made on the reliability of the communication channels and the process crash-stop semantics, make the framework of limited use in practical distributed systems, where messages may indeed be lost and processes may recover after a crash. The motivation of this paper was to precisely extend that framework, by alleviating the need for those assumptions.

We considered a system model where a process is said to be *green*, when it eventually is *never crashed*, and said to be *red* otherwise. We presented an algorithm that solves consensus in a system where, (1) there is a majority of *green* processes, (2) communication channels preserve a minimal fairness property regarding message loss, and (3) the failure detector ensures *eventual weak accuracy* (i.e., eventually, there is a green process that is never suspected by any green process), and *(strong completeness)* (i.e., eventually, every red process is suspected by every green process). In runs where assumptions (1), (2) or (3) do not hold, our algorithm still ensures consensus safety (e.g., no two processes will ever reach different decisions).

Although assumption (1) above formally means that eventually a majority of processes never crash, in fact, the assumption is required to hold only enough time to allow some green process to decide (at most the time for n rounds of the consensus algorithm). Assumption (2) intuitively states that a message sent by a green process to a green process must have a non null probability of being received. Without such a property, any interesting distributed problem would be trivially impossible to solve. Finally, assumption (3) is for example satisfied whenever the processes in the *green majority* communicate in a timely manner (given a failure detector implemented with time-outs for instance).

We have shown that assumption (3) above is actually required for any algorithm that solves consensus in the crash-recover model. Interestingly, in contrast to the crash-stop model, *eventual weak accuracy* and *weak completeness* (i.e., eventually, every red process is suspected by *some* green process) are not sufficient to solve consensus. This shows that the crash-recover model is *strictly weaker* than the crash-stop model, as more knowledge about failures is required to solve consensus.

Our algorithm has the nice property that, in runs where no process crashes or is suspected to have crashed (the most frequent runs in practice), consensus is reached after (only) two communication steps. This early delivery property [12], together with the relatively weak assumptions we make on the underlying system, make our algorithm adequate for solving consensus in practice. Furthermore, our algorithm can easily be extended to solve consensus related problems, such as atomic broadcast [5] and non-blocking (weak) atomic commitment [9].

References

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, and L. Zuck. Reliable communication over unreliable channels. *Journal of the ACM*, 41(6), July 1994.

[2] Ö. Babaoğlu, R. Davoli, and A. Montresor. Failure detectors, group membership and view-synchronous communication in partitionable asynchronous systems. Technical Report UBLCS-95-18, University of Bologna, November 1995.

[3] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *10th Intl. Workshop on Distributed Algorithms (WDAG'96)*. Springer Verlag, LNCS 1151, October 1996.

[4] T. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving Consensus. *Journal of the ACM*, 43(4), July 1996.

[5] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(1), March 1996.

[6] F. Cristian and C. Fetzer. The Timed Asynchronous system model. Technical Report CSE97-519, Department of Computer Science, University of California San Diego, 1997.

[7] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi. Failure detectors in omission failure environments. Technical Report TR96-1608, Cornell University, 1996.

[8] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed Consensus with one faulty process. *Journal of the ACM*, 32(2), April 1985.

[9] R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and Consensus. In *9th Intl. Workshop on Distributed Algorithms (WDAG'95)*. Springer Verlag, LNCS 972, September 1995.

[10] R. Guerraoui, R. Oliveira, and A. Schiper. Stubborn communication channels. Technical report, LSE, Département d'Informatique, Ecole Polytechnique Fédérale de Lausanne, Switzerland, December 1996.

[11] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[12] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3), April 1997.

Author's brief biography

Rui Oliveira is a PhD student at the EPFL (Federal Institute of Technology in Lausanne) on leave from Universidade do Minho. Rui Oliveira received a MS in computer science from the Universidade do Minho in 1994. His current research interests are fault-tolerant distributed algorithms and systems. Rui Oliveira is a member of the Association for Computing Machinery.

Rachid Guerraoui is a lecturer and research associate at the EPFL (Federal Institute of Technology in Lausanne). Rachid Guerraoui received a PhD in computer science from University of Orsay in 1992. His current research interests are fault-tolerant distributed algorithms and systems, as well as object-oriented computing.

André Schiper has been a professor of Computer Science at the EPFL (Federal Institute of Technology in Lausanne) since 1985, leading the Operating Systems laboratory. During the academic year 1992-93 he was on sabbatical leave at Cornell University, Ithaca (NY). He was the program chair of the 1993 International

Workshop on Distributed Algorithm (WDAG-7), and co-organizer of the International Workshop "Unifying Theory and Practice in Distributed Systems" (Schloss Dagstuhl, Germany, September 1994). He is member of the ESPRIT Basic Research Network of Excellence in Distributed Computing Systems Architectures (CaberNet). His current research interests are in the areas of fault-tolerant distributed systems and group communication, which has led to the development of the *Phoenix* group communication middleware.