Distrib. Comput. (2011) 24:165–175
DOI 10.1007/s00446-010-0116-2

Randomization can be a healer: consensus with dynamic omission failures

Henrique Moniz · Nuno Ferreira Neves · Miguel Correia · Paulo Veríssimo

Received: 31 October 2009 / Accepted: 30 July 2010 / Published online: 10 August 2010

Abstract Wireless ad-hoc networks are being increasingly used in diverse contexts, ranging from casual meetings to disaster recovery operations. A promising approach is to model these networks as distributed systems prone to dynamic communication failures. This captures transitory disconnections in communication due to phenomena like interference and collisions, and permits an efficient use of the wireless broadcasting medium. This model, however, is bound by the impossibility result of Santoro and Widmayer, which states that, even with strong synchrony assumptions, there is no deterministic solution to any non-trivial form of agreement if $n-1$ or more messages can be lost per communication round in a system with n processes. In this paper we propose a novel way to circumvent this impossibility result by employing randomization. We present a consensus protocol that ensures safety in the presence of an unrestricted number of omission faults, and guarantees progress in rounds where such faults are bounded by $f \leq \lceil \frac{n}{2} \rceil (n-k)+k-2$, where k is the number of processes required to decide, eventually assuring termination with probability 1.

1 Introduction

Wireless ad-hoc networks are being increasingly used in diverse contexts, ranging from casual meetings to disaster recovery operations. The ability of distributed processes to execute coordinated activities despite failures is important to distributed systems, including those based on wireless ad-hoc networks. Such coordination requires agreement among the processes, a problem that has taken many incarnations in the literature: consensus, Byzantine generals, and interactive consistency are just a few examples [13,18,24]. The prevalent aspect of these formulations is that at some point in their execution the processes involved have to agree on a common item of information.

Under the traditional models for distributed systems, faults are static and component-bound, i.e., a fault is associated to a particular component that is forever considered faulty. The faulty component can be a process or a communication link (e.g., [24,25]). These models are referred to as *component failure models*. For systems based on these models to operate correctly, a certain number of components must not exhibit failures during their entire operation time.

This approach, however, is not well adapted to wireless ad-hoc networks. First, in these environments, faults have a more dynamic and transient nature. The nodes are usually subject to momentary disconnection due to node mobility and other environmental phenomena such as electromagnetic interference, fading, collisions, etc. These events may result in message loss or corruption, but should not be sufficient to permanently assume a process or link as faulty, specially because they can possibly affect many processes during the lifetime of the system. Emergence of wireless networks, there is an increasing need for models that accurately capture the reality of these environments.

Second, the openness of wireless ad-hoc networks provides a natural broadcasting medium, where the cost of transmitting a message to multiple processes can be just the same of transmitting it to a single process, as long as they are within communication range. To take advantage of this

This work was partially supported by the FCT through the Multiannual and the CMU-Portugal Programmes, and the project PTDC/EIAEIA/100894/2008 (DIVERSE).

H. Moniz (✉) · N. F. Neves · M. Correia · P. Veríssimo
Faculty of Sciences, University of Lisboa, LASIGE,
Lisbon, Portugal
e-mail: hmoniz@di.fc.ul.pt

feature, it becomes necessary to depart from the common modeling assumption of reliable point-to-point channels, usually employed by the component failure models. Developing a system based on this assumption forces the implementation of end-to-end message delivery mechanisms (similar to TCP), which significantly increases the medium access contention, impairing the overall performance. The unreliability inherent to radio communications has to be dealt with in some other way. Models that assume unreliable communication links are more adjusted to wireless networking. Tolerance to message loss becomes integrated within the semantics of the algorithms, instead of being abstracted by typically inefficient implementations.

More adapted to the wireless ad-hoc environments is the *communication failure model* [28,29]. This model differs from the component failure models in the sense that it focuses on the effects of faults rather than their source. On message-passing systems, any failure, regardless of its nature, will ultimately manifest itself as transmission faults. For example, a process crash will manifest into a series of transmission omission faults with the crashed process as sender, and a process that is attacked and falls under the control of a maliciously adversary may manifest into a series of transmission corruption faults where the contents of the messages are modified relative to the original protocol. Such an approach implicitly allows every component of the system to eventually fail. The only restriction is placed on the number of faults that simultaneously manifest in the system.

Research in this model, however, has been limited mainly due to two fundamental reasons. When the model was introduced by Santoro and Widmayer in 1989, a stringent impossibility result came along with it [28]. This result applies to the *k-agreement* problem among n processes, in which k out of n processes must agree on a binary value $v \in \{0, 1\}$. The Santoro-Widmayer impossibility result applies to non-trivial agreement, i.e., for $k > \lceil n/2 \rceil$. It states that there is no finite time deterministic algorithm that allows n processes to reach *k-agreement* if more than $n - 2$ transmission failures occur in a communication step. This is a very discouraging result since the crashing of a single process necessarily results in $n - 1$ transmission failures, rendering this form of agreement impossible. Moreover, this result is produced under strong time assumptions where both the processes' relative processing times and communication delays are bound by known constants (i.e., a synchronous system).

The second reason has probably to do with some lack of practical interest in this model prior to the emergence of wireless ad-hoc communication. For distributed systems based on wired networks, it was safe and convenient to assume end-to-end reliable delivery mechanisms, since the implementation of such mechanisms did not represent a significant performance overhead. Interestingly, these models are also bound by an impossibility result: the FLP result [14]. It states that consensus is impossible to solve deterministically in asynchronous systems (i.e., where there are no assumptions about the processes' relative processing times and communication delays) if just a single process can fail.

Thus, on one hand we have asynchronous systems, bound by the FLP impossibility result, where agreement is impossible even if communication is reliable. On the other hand, we have systems that are synchronous but the communication is unreliable so they are bound by the Santoro-Widmayer impossibility result, also making agreement impossible. While several solutions have been proposed over the years to circumvent the FLP result (e.g., randomization [3,26], partial synchrony models [11], failure detection [8], wormholes [22]), the result of Santoro and Widmayer, for the reasons stated above, has not received comparable attention. Nevertheless, getting past the current upper bound of $n - 2$ transmission failures is paramount to the embracing of the communication failure model for emergent networking environments.

This paper presents a protocol that circumvents the Santoro-Widmayer impossibility result by employing *randomization*, which has never been applied before in the context of the communication failure model. The Santoro-Widmayer impossibility result rules out deterministic solutions to agreement in this model. Randomization takes a probabilistic approach to the problem, and has been used in the past to solve consensus in FLP-bound systems (starting with [3,26]). It overcomes previous limitations by supplying processes with access to random information (e.g., a coin flip) and combining this with a refinement of the problem statement where a decision is ensured with a probability of 1.

The proposed protocol is a binary *k-consensus* algorithm that employs randomization to tolerate omission transmission faults. The algorithm allows at least k processes to decide on a common binary value in a system with n processes such that $k > \frac{n}{2}$. The safety properties of consensus (i.e., validity and agreement) are ensured even with an unrestricted number of faults, while the liveness property (i.e., termination) is ensured if the number of faults per round does not exceed $\lceil \frac{n}{2} \rceil (n - k) + k - 2$. This algorithm is adequate for wireless ad-hoc networks because it allows one to take advantage of the broadcasting medium in an efficient way and, at the same time, ensures safety under severe communication problems that lead to many message losses. The termination is achieved with probability 1 when communication becomes stable, i.e., when the threshold above is satisfied. Furthermore, the algorithm is efficient in the sense that it is *fast-learning* [17], i.e., it terminates in 2 communication steps under favorable conditions (i.e., with no message losses, benign patterns of message losses, and/or all processes having the same initial value).

The remainder of the paper is organized as follows: Section 2 discusses the related work. Section 3 formalizes the

k-consensus problem, and the next section presents the system model. Section 5 describes the algorithm, and the correctness proofs are provided in the following section. Section 7 discusses extensions to the algorithm, and finally, Sect. 8 concludes the paper.

2 Related work

The problem of reaching agreement with unreliable communication links goes back to 1975 when Akkoyunlu et al. pointed out that an agreement between two processes connected by unreliable communication paths leads to an infinite exchange of messages [2]. In 1978 Gray identified essentially the same problem by formulating the *generals paradox* [15]. He showed that there is no deterministic protocol that allows agreement between two processes connected by an unreliable communication link. This problem is often referred to as the *coordinated attack problem* from the formalization of Lynch [19]. Varghese and Lynch later proposed a randomized solution to the coordinated attack problem where the protocol runs for a fixed number of rounds and agreement is reached with a probability proportional to the number of rounds [31].

The previous result was generalized to an arbitrary number of processes by Santoro and Widmayer [28,29]. Their contribution provides an important impossibility result. It states that there is no fixed-time solution to the problem of *k-agreement* (i.e., $k > \lceil \frac{n}{2} \rceil$ processes decide the same value 0 or 1) in a system with n processes if more than $n-2$ links are allowed to lose messages. Their problem statement represents a weaker form of agreement than ours. The definition of *k-agreement* allows processes to decide different values as long as k decide the same value, while in our definition (i.e., *k-consensus*) no process is allowed to decide a different value.

The work of Chockler et al. presents algorithms that solve consensus in systems where nodes fail only by crashing and messages can be lost due to collisions [10]. Their solution assumes that processes have access to a collision detector that determines when message collisions occur, which allows nodes to take recovery measures when messages are lost. Message omissions other than those due to collisions, however, are not covered by their model. By contrast, our model assumes message omissions regardless of their nature.

Two other works also solve consensus under dynamic communication failures. The work of Biely et al. does so by addressing the problem in the context of the *heard-of model* of Charron-Bost and Schiper [4,9]. This model permits a fine-grained specification of the fault patterns allowed in the system, thus being able to distinguish the cases where the fault pattern exceeds the lower bound of Santoro and Widmayer but is not harmful to the system as a whole (e.g., $n-1$ faults are harmful to the system if they originate at the same process, but may not be if they originate each one at a different process). The work of Schmid et al. presents an analogous contribution in the sense that it restricts the number of faults that each process may experience such that the harmful fault patterns are avoided [30]. None of these two contributions, however, deal with the problematic essence of the Santoro-Widmayer impossibility result, which is the failure of every transmission from a single process rendering consensus impossible. This implies that consensus remains unsolvable if, for instance, in a wireless ad-hoc network, a single node falls out of range of every other node for an unknown period of time.

The crash-recovery models combined with failure detection mechanisms can also be applied to wireless environments because of their ability to capture the disconnection and eventual reconnection of processes [1,12,16,23]. The granularity of these models, however, was not intended to capture connectivity scenarios likely to arise in wireless environments. For example, consensus can not be solved in scenarios where every *good* process (i.e., one that is not crashed) has some faulty link to another good process. Such configuration violates the eventual weak accuracy property required by failure detectors.

3 The *k*-consensus problem

The *k-consensus* problem considers a set of n processes where each process p_i proposes a binary value $v_i \in \{0, 1\}$, and at least $k > \frac{n}{2}$ of them have to decide on a common value proposed by one of the processes. The remaining $n-k$ processes do not necessarily have to decide, but if they do, they are not allowed to decide on a different value. Our problem formulation is designed to accommodate a randomized solution and is formally defined by the properties:

Validity. If all processes propose the same value v, then any process that decides, decides v.
Agreement. No two processes decide differently.
Termination. At least k processes eventually decide with probability 1.

4 System model

The system is composed by a fixed set of n processes $\Pi = \{p_0, p_1, \ldots, p_{n-1}\}$, where each process p_i has an unique identifier i. The timing model is assumed to be synchronous. This implies that (1) there is a known upper bound on time required by a process to execute a step, (2) there is a known upper bound on message transmission delays, and (3) every process has a local clock with a known bounded rate of drift

with respect to real-time. Later, in Sect. 7, it is discussed how the system model can be weakened to an asynchronous system.

The communication between processes proceeds in synchronous rounds. At each round, every process $p_i \in \Pi$ executes the following actions: (1) transmits a message m to every process $p_j \in \Pi$, including itself, by invoking `broadcast(`m`)`, (2) receives the messages broadcast in the current round by invoking `receive()`, and (3) performs a local computation based on its current state and the set of messages received so far. We should note that the assumption of a broadcast operation generating n transmissions arises from the necessity of modeling the possibility of non-uniform message delivery by the processes. In practice, this operation can still be implemented efficiently by transmitting a single message.

Processes are modeled so as not to exhibit faulty behavior, i.e., they correctly follow the protocol until termination. The notion of a faulty process is instead captured by the assumption of faulty message transmissions. For example, a crashed process can be expressed by the loss of every message transmitted by it. The model considers only omission transmission failures. A transmission between two processes p_i and p_j is subject to an omission failure if the message sent by p_i is not received by p_j.

In rounds where omission faults are $f \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$ out of the n^2 transmissions that occur (where k is the number of processes required to decide), the protocol necessarily makes some progress that eventually leads to a decision. Therefore, if enough of these rounds occur, then the protocol ensures termination with probability 1. Nevertheless, to simplify the correctness proofs we will assume that there is some unknown time after which at most f faulty transmissions occur at each round. The number of faults per round prior to this is unrestricted and can for instance match the total number of transmissions n^2.

Finally, every process $p_i \in \Pi$ has access to a local random bit generator acessible via a function `coin`$_i$`()` that returns unbiased bits observable only by p_i.

5 The algorithm

This section presents a k-consensus algorithm (Algorithm 1). The algorithm is tolerant to omission faults and relies on each process p_i having access to a *local coin*[1] mechanism that returns random bits observable only by p_i (e.g., [3,5]). The first local coin protocol was proposed by Ben-Or in 1983 [3], of which our protocol is reminiscent. Safety (i.e., the *validity* and *agreement* properties of consensus) is ensured by the algorithm regardless of the number of omission faults that occur per round, while liveness (i.e., the *termination* property) is ensured if, after some arbitrary number of rounds, the number of omission faults per round is $f \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$.

The internal state of a process p_i is comprised by three variables: (1) the *phase* $\phi_i \geq 1$, (2) the *proposal value* $v_i \in \{0, 1\}$, and (3) the *decision status* $status_i \in \{decided, undecided\}$. Each process starts the execution with $\phi_i = 1$, $status_i = undecided$, while v_i is set to the initial proposal value indicated by the input register $proposal_i$.

A round of the algorithm is executed as follows. Upon every clock tick (line 5), each process p_i broadcasts a message of the form $\langle i, \phi_i, v_i, status_i \rangle$ containing its identifier and the variables that comprise the internal state, and receives the messages broadcast by all processes (lines 6–7). Some of the messages that a process is supposed to receive may be lost. The messages that a process p_i receives at any round are accumulated in a set V_i (line 8). Based on its current internal state and the messages accumulated so far in set V_i, each process p_i performs a state transition (i.e., modifies ϕ_i, v_i or $status_i$).

Before continuing the explanation of the state transition, it is important to note the distinction between round and phase. The term *round* pertains to a periodic execution of the protocol activated by a synchronous event, a clock tick in this case. The term *phase* pertains to a monotonic variable ϕ_i that is part of the internal state of a process p_i, and whose value increases as p_i accumulates messages of a certain form in set V_i. How exactly ϕ_i is updated is explained below. For now, it is beneficial to retain that for any given round, any two processes p_i and p_j can have different phase values $\phi_i \neq \phi_j$.

A process p_i performs a state transition when one of two conditions occur:

1. The set V_i holds one message from some process p_j whose phase ϕ_j is *higher* than the phase ϕ_i of p_i.
2. The set V_i holds more than $\frac{n}{2}$ messages whose phase is *equal* to the phase ϕ_i of p_i.

The first case is straightforward (lines 9–13). As long as there is some message in V_i with a higher phase than that of p_i, the loop iterates until the state of p_i is matched with the state of the message with the highest phase value.

The second case involves the execution of more steps (lines 14–32). The way a process p_i updates its state depends on whether the current number of its phase ϕ_i is odd (i.e., $\phi_i \bmod 2 = 1$) or even (i.e., $\phi_i \bmod 2 = 0$). The odd phase essentially guarantees that if two processes set their proposal to a value 1 or 0, they do it for the same value. The even phase is where a process decides if it learns that a majority of processes have the same proposal value.

[1] As opposed to a *shared coin* that returns bits observable by all processes (e.g., [6,26]).

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

```
1  φ_i ← 1;
2  v_i ← proposal_i;
3  status_i ← undecided;
4  V_i ← ∅;
5  for each clock tick do
6      broadcast(⟨i, φ_i, v_i, status_i⟩);
7      M_i ← receive();
8      V_i ← V_i ∪ M_i;
9      while ∃⟨∗,φ,v,status⟩∈V_i : φ > φ_i do
10         φ_i ← φ;
11         v_i ← v;
12         status_i ← status;
13     end
14     if |{⟨∗, φ, ∗, ∗⟩ ∈ V_i : φ = φ_i}| > n/2 then
15         if φ_i mod 2 = 1 then                                   /* odd phase */
16             if ∃_{v∈{0,1}} : |{⟨∗, φ, v, ∗⟩ ∈ V_i : φ = φ_i}| > n/2 then
17                 v_i ← v;
18             else
19                 v_i ← ⊥;
20             end
21         else                                                    /* even phase */
22             if ∃_{v∈{0,1}} : |{⟨∗, φ, v, ∗⟩ ∈ V_i : φ = φ_i}| > n/2 then
23                 status_i ← decided;
24             end
25             if ∃_{v∈{0,1}} : |{⟨∗, φ, v, ∗⟩ ∈ V_i : φ = φ_i}| ≥ 1 then
26                 v_i ← v;
27             else
28                 v_i ← coin_i();
29             end
30         end
31         φ_i ← φ_i + 1;
32     end
33     if status_i = decided then
34         decision_i ← v_i;
35     end
36 end
```

Algorithm 1 k-consensus algorithm

If $\phi_i \bmod 2 = 1$ (lines 15–20), then the proposal value v_i is updated in the following way: if there are more than $\frac{n}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with $\phi = \phi_i$ and the same value $v \in \{0, 1\}$, then v_i is set to v (lines 16–17). Otherwise, it is set to a special value $\perp \notin \{0, 1\}$ indicating a lack of preference (lines 18–19).

If $\phi_i \bmod 2 = 0$ (lines 21–30), then the process sets $status_i$ to $decided$ if there are more than $\frac{n}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with the same value $v \neq \perp$ and $\phi = \phi_i$ (lines 22–24). The proposal value v_i is updated to v if there is at least one message of the form $\langle \phi, v, * \rangle$ in V_i with a value $v \neq \perp$ and $\phi = \phi_i$. Otherwise, v_i is set to the value of function $\texttt{coin}_i()$, which returns a random number 0 or 1, each with a probability $\frac{1}{2}$ (lines 25–29). Regardless of whether the phase ϕ_i is odd or even, its value is always incremented by one unit (line 31).

At the end of each round, a process p_i checks if $status_i$ has been set to $decided$. If so, it decides by setting the output variable $decision_i$ to the current proposal value v_i (lines 33–35). Any further accesses to this variable do not alter its value. Hence, they have no impact on the correctness of the algorithm.

The presented algorithm is not *quiescent*, i.e., processes do not voluntarily stop sending messages. They have continue their execution after deciding to help other processes decide. Section 7 discusses how the algorithm can be changed such that quiescence is achieved.

6 Correctness proof

The correctness proof of the algorithm is divided in two subsections. Section 6.1 is concerned with the safety properties of the algorithm: *validity* and *agreement*. Section 6.2 proves the liveness property: *termination*.

6.1 Safety

The *validity* and *agreement* properties are proven on the assumption that the system might be subject to an unbounded number of transmission omission faults per round.

The first lemma shows that if there is some phase where every process has the same proposal value, then at any subsequent phase they all have that same proposal value. This supports Lemma 2, which essentially states that whenever there is an odd phase ϕ where every process has the same proposal value, every process that reaches a phase higher than $\phi+1$ decides on that value. With these two lemmas, the validity property is easily proven (Theorem 1).

Lemma 1 *If every process p_i with phase value $\phi_i = \phi$ has the same proposal value $v_i = v \neq \perp$, then every process p_j that sets $\phi_j = \phi + 1$ also sets $v_j = v$.*

Proof The lemma is going to be proven by induction on the number of processes that reach phase $\phi + 1$. *Basis:* Without loss of generality, let p_1 be the first process that sets $\phi_1 = \phi+1$. In this case, process p_1 must have received more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ (line 14). Since every process p_i with $\phi_i = \phi$ has the same value $v_i = v \neq \perp$, every broadcast message of the form $\langle *, \phi, *, * \rangle$ carries the same proposal value v (line 6). This implies that the more than $\frac{n}{2}$ messages received by process p_1 have the form $\langle *, \phi, v, * \rangle$ with the same value v. Therefore, p_1 must set its proposal value to v (either on line 17 or 26). *Inductive step:* Assume that every process p_u with $1 \leq u \leq j-1$ has $\phi_u = \phi + 1$ and $v_u = v$. Now we want to demonstrate that when p_j sets $\phi_j = \phi + 1$ it will also set $v_j = v$. In order for process p_j to set $\phi_j = \phi + 1$ it must have in set V_j (1) more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ (line 14) or (2) at least a message of the form $\langle *, \phi + 1, *, * \rangle$ (line 9). Condition (1) corresponds to the basis case, and therefore it has already been shown that p_j sets $v_j = v$. Condition (2) also results in the same outcome, since by hypothesis message $\langle *, \phi + 1, *, * \rangle$ must have been transmitted by one of the p_u processes, and therefore p_j also sets $\phi_j = \phi + 1$ and $v_j = v$ (lines 10–11). □

Lemma 2 *Let ϕ be some odd phase (i.e., $\phi \bmod 2 = 1$). If every process with phase value ϕ has the same proposal value v, then every process that sets its phase to any value $\phi' > \phi + 1$ decides v.*

Proof Since every process with odd phase value ϕ has the same proposal value v, by Lemma 1, every process that reaches even phase $\phi+1$ also has proposal value v (either on lines 10–11 or lines 17 and 31). Let p_i be the first process to set phase value $\phi_i = \phi+2$. Since there is no other process p_j with phase value $\phi_j > \phi + 1$, the only way for p_i to go from phase $\phi + 1$ to $\phi + 2$ is to receive more than $\frac{n}{2}$ messages of the form $\langle *, \phi+1, *, * \rangle$ (line 14). Since $\phi+1$ is even and all these messages carry the same proposal value v, this implies that p_i sets $status_i = decided$, $v_i = v$ and $\phi_i = \phi + 2$ (lines 23, 26, 31). Consequently, process p_i can now decide v (line 34).

The next process that sets its phase value to $\phi + 2$ also decides v because it either accumulates more than $\frac{n}{2}$ messages with phase value $\phi + 1$ and same proposal value v (lines 23, 26, 31 and 34), or receives a message from p_i of the form $\langle *, \phi + 2, v, decided \rangle$ (lines 10–12 and 34). This reasoning can be applied recursively to any other process that sets its phase value to $\phi + 2$. It follows that any process that sets its phase value to $\phi' \geq \phi + 2$ must either have been at phase $\phi + 2$, and hence decided, or it must have received some message from a process that went through phase $\phi+2$, and thus also decided. Therefore, every process that sets its phase to any value $\phi' > \phi + 1$ decides v. □

Theorem 1 *If all processes propose the same value v, then every process that decides, decides v.*

Proof If every process has the same initial proposal value v, then they all set proposal value to v in odd phase 1 (lines 1–2). Therefore, by Lemma 1, every process p_j that sets phase $\phi_j = 2$ also has proposal value $v_j = v$. Moreover, by Lemma 2, every process p_i that sets its phase to $\phi_i > 2$, decides v. □

The following lemma proves that any two processes in the same even phase can not have different proposal values 0 and 1. In other words, if some process has proposal value $v \in \{0, 1\}$, then any other process can only have a proposal value of v or $\perp$. This lemma is essential to the *agreement* property in Theorem 2, which shows that when the first process decides, every process proposes the same value at the following phase. By lemma 2, this inevitably leads to a decision on the same value.

Lemma 3 *In any even phase ϕ, there are no two processes p_i and p_j that receive messages of the form $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$, respectively.*

Proof Suppose otherwise. Then p_i and p_j are two processes with phase value ϕ that, respectively, receive a message $\langle *, \phi, 0, * \rangle$ from p_u and a message $\langle *, \phi, 1, * \rangle$ from p_w. This implies that process p_u has set $v_u = 0$ either because on odd phase $\phi - 1$ it accumulated more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, 0, * \rangle$ (lines 16–17, 31), or because it received a message $\langle *, \phi, 0, * \rangle$ (lines 10–11) from a process that had accumulated that majority of $\langle *, \phi - 1, 0, * \rangle$ messages. Using a similar reasoning, in order for process p_w to have set $v_w = 1$, some process must have received on odd phase $\phi-1$ more than $\frac{n}{2}$ messages of the form $\langle *, \phi-1, 1, * \rangle$. But this is a contradiction because only one of the proposal values 0 and 1 can be in a majority of the messages broadcast for any particular phase number. □

Theorem 2 *No two processes decide differently.*

Proof Let p_i be the first process to decide, and do so when phase $\phi_i = \phi$ (line 34). Without loss of generality, let the decision value be 1. Then, set V_i must contain more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, 1, undecided \rangle$, and $\phi - 1$ must be even (to allow the execution of lines 23, 26, and 31). By Lemma 3, no other process p_j can receive a message of the form $\langle *, \phi - 1, 0, * \rangle$. Therefore, every other process p_j with phase $\phi_j = \phi$ has proposal value $v_j = 1$ either because it accumulates more than $\frac{n}{2}$ messages with at least one being of the form $\langle *, \phi - 1, 1, * \rangle$ (line 26), or because it receives a message $\langle *, \phi, 1, * \rangle$ (line 11) transmitted by process p_i (or another process that sets its proposal value to 1). Additionally, since all processes with phase ϕ have proposal value 1, then by Lemmas 1 and 2, every process that decides in phase $\phi' > \phi$ will do it for value 1. □

6.2 Liveness

The remainder of the proof handles the *termination* property of consensus (Theorem 3). For this part we work on the assumption that the message scheduling falls under the control of an *adversary* that can cause no more than f faults per round, with $f \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$.

The rationale for the *termination* property is based on the idea that as long as processes keep increasing their phase values, a decision is eventually reached. As we have seen from the safety proofs, if there is unanimity at some odd phase ϕ, then every process that reaches any phase higher than $\phi + 1$ decides. The idea is to show that k processes can reach any arbitrarily high phase value, and that unanimity eventually happens.

The following two lemmas dictate how processes increment their phase values in tandem. Lemma 5, in particular, states that for any process with phase value ϕ, eventually k processes must have a phase value equal or higher than $\phi - 1$.

Lemma 4 *If some process p_i has some phase value $\phi_i > 1$, then there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi_i - 1$ and $|S| > \frac{n}{2}$.*

Proof Given a phase number $\phi > 1$, then there must be some process p_i that is the first to set its phase to $\phi_i = \phi$. In order to do this, p_i must have more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, *, * \rangle$ in set V_i (line 14). It follows that there are more than $\frac{n}{2}$ processes that were at some point in time in phase $\phi - 1$. □

Lemma 5 *If some process p_i has phase value $\phi_i = \phi$, then eventually there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi - 1$ and $|S| \geq k$.*

Proof Suppose otherwise. By Lemma 4, if some process p_i has $\phi_i = \phi > 1$, then there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi - 1$ and $|S| > \frac{n}{2}$. Let $R^+ = S$ where $\frac{n}{2} < |R^+| < k$, and R^- be the set of remaining processes, i.e., $\forall_{p_u \in R^-} : \phi_u < \phi - 1$ where $n - k < |R^-| < \frac{n}{2}$.

By assumption, the adversary can create at most $f = f_1 + f_2$ message omissions per round, where $f_1 = \lceil \frac{n}{2} \rceil (n-k)$ and $f_2 = k - 2$. In order to prevent processes in R^- from reaching $\phi_u \geq \phi - 1$, the adversary must omit every message from processes of R^+ to R^- (due to lines 9–13). Since, $|R^+| > \frac{n}{2}$ and $|R^-| > n - k$, this implies the omission of more than $\lceil \frac{n}{2} \rceil (n-k)$ messages. It is clear that after consuming f_1 faults in this manner, there are at most $n-k$ processes in R^- that do not receive any message from R^+. Since by definition $|R^-| - (n-k) = k - |R^+| > 0$, there must be $k - |R^+|$ processes in R^- that could still receive messages from every process in R^+. Let R^-_* denote the set of processes in this situation. To prevent every process p_u in R^-_* from reaching $\phi_u \geq \phi - 1$, the adversary must create $|R^+||R^-_*|$ omissions, where $|R^+| + |R^-_*| = k$. However, the adversary only has $f_2 = k - 2 = |R^+| + |R^-_*| - 2$ faults available. This creates a contradiction because $|R^+||R^-_*| > |R^+| + |R^-_*| - 2$, for all $|R^+| \geq 1$ and $|R^-_*| \geq 1$. This implies that some process in $|R^-|$ always increases its phase value when $\frac{n}{2} < |R^+| < k$. □

The following lemma is central to this part of the proof. It shows that in any communication round where the number of omission faults is not higher than $\lceil \frac{n}{2} \rceil (n - k) + k - 2$, some process increases its phase value.

Lemma 6 *Let R^+ represent the set of processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi$, with $|R^+| = k + \alpha$ and $0 \leq \alpha \leq n - k$. Let α or more processes in R^+ have phase ϕ and the remaining processes of R^+ have phase $\phi + 1$. Let R^- be the set of processes such that $\forall_{p_j \in R^-} : \phi_j < \phi$, with $|R^-| = n - k - \alpha$. Whenever a round has such configuration, some process increases its phase value.*

Proof Suppose otherwise. Then, under the lemma conditions, there must be a message schedule where at some round no process increases its phase value.

In order to prevent every process in R^- from increasing its phase value, the adversary must omit every message from R^+ to R^- (due to lines 9–13). This requires that $|R^+||R^-|$ faults must be spent. Since $|R^+||R^-| = (k + \alpha)(n - k - \alpha)$ and the total number of omissions per round is $f = \lceil \frac{n}{2} \rceil (n - k) + k - 2$, then the adversary is left with no more than $f - |R^+||R^-| \leq (\alpha + \lceil \frac{n}{2} \rceil + k - n)\alpha + k - 2$ faults.

In order to block each of the α processes in R^+ with phase ϕ, the adversary must omit all messages from processes in R^+ with phase $\phi + 1$ (line 9) and it must prevent the reception of more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ also from processes in R^+ (line 14). This implies that each of the α processes with phase ϕ can receive the $n - k - \alpha$

messages from processes in R^- and at most $\lfloor \frac{n}{2} \rfloor$ messages from processes in R^+. Therefore, the adversary must create at least $\left[n - (\lfloor \frac{n}{2} \rfloor + n - k - \alpha)\right]\alpha$ faults to stop the progression of the α processes. Since $\left[n - (\lfloor \frac{n}{2} \rfloor + n - k - \alpha)\right]\alpha = (\alpha + \lceil \frac{n}{2} \rceil + k - n)\alpha$, the adversary is left with no more than $k - 2$ faults.

For the remaining k processes in R^+, there are two possible cases:

1. First consider the two extreme situations, where all k processes either have phase value ϕ or $\phi + 1$. Since the adversary only has $k - 2$ faults left, some process has to receive more than $\frac{n}{2}$ messages with the same phase ϕ or $\phi + 1$. Therefore, some process increases its phase value (line 14).
2. Second consider that some of the k processes have phase value $\phi + 1$ and the others have phase value ϕ. Let H be the set of processes with $\phi + 1$ and L the set of processes with ϕ, such that $|H| + |L| = k$. To block the processes in L, the adversary has to omit $|H||L|$ messages (due to line 9). Since the adversary only has $k-2 = |H|+|L|-2$ faults left, it can not prevent some process from increasing its phase because $|H||L| > |H| + |L| - 2$ for all $|H| \geq 1$ and $|L| \geq 1$. □

Lemma 7 *Let $\phi_{init} = 1$ be the initial phase value for all processes. Some process p_i eventually sets $\phi_i > \phi_{init}$.*

Proof If every process has the same phase value ϕ_{init}, then according to the conditions of Lemma 6, this is equivalent to having every process in set R^+ with phase ϕ_{init}, such that $|R^+| = n$. Therefore, by Lemma 6, some process has to increase its phase value and set $\phi_i > \phi_{init}$. □

The following lemma wraps up the progress of phase values by stating that some process can reach any arbitrarily high phase value.

Lemma 8 *If some process has phase value ϕ, then eventually some process must have phase value $\phi + 1$.*

Proof If some process has phase value ϕ, then by Lemma 5, eventually there is a set R^+ of k or more processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi - 1$. This implies that the system must reach a configuration where there are two sets of processes R^+ and R^- according to the conditions of Lemma 6. When this happens, by the same lemma, some process will increase its phase. This process can be in one of three possible cases: (1) a process of R^-; (2) a process with phase number $\phi - 1$ of R^+; or (3) a process with phase number ϕ of R^+. The system configuration resulting from cases (1) and (2) falls under the conditions of Lemma 6, and therefore more processes will continue to increase their phase. Consequently, in the most extreme scenario, the system will evolve to a configuration where all process are in phase number ϕ, and case (3) will necessarily have to occur, and some process p_i will set its phase number to $\phi_i = \phi + 1$. □

Finally, the *termination* property is proven by showing that as long as processes keep increasing their phase value, then unanimity eventually happens and, consequently, a decision by k processes.

Theorem 3 *At least k processes eventually decide with probability 1.*

Proof The proof is organized in two parts. First, we show that as messages are received, processes make progress on the protocol execution and continue to increase their phase number. Second, we demonstrate that due to this progression, eventually the system will reach to a configuration where at least k processes decide with probability 1.

First part: By Lemma 7, some process p_i eventually increases its phase number from the initial phase number, i.e., $\phi_i = \phi > \phi_{init}$. Then, by Lemma 8, some process will eventually set its phase number to $\phi + 1$. Moreover, by Lemma 5, k or more processes set their phase value to at least ϕ. Since these lemmas can be applied repeatedly, this ensures that for any arbitrary phase value ϕ', there is some subset S of at least k processes such that every process $p_j \in S$ eventually has phase value $\phi_j \geq \phi'$.

Second part: By Lemma 3, no two processes with the same even phase value ϕ can receive messages $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$. Therefore, any process p_i that enters the **if** condition of line 14, and sets $\phi_i = \phi + 1$ (line 31), must set its proposal value v_i either to a common value v (line 26) or to a random value 1 or 0 (lines 28). This implies that the probability of every process with odd phase value $\phi + 1$ having the same proposal value v is $p \geq 2^{-\gamma}$, where γ is the number of processes with phase $\phi + 1$.

As the protocol progresses, and the phase number of processes increases, the probability of not existing an odd phase where every process proposes the same value v is $\lim_{\phi \to \infty}(1 - p)^{\phi} = 0$. Thus, eventually there will be an odd phase ϕ_t where every process p_i with phase $\phi_i = \phi_t$ has the same proposal value v. According to Lemma 2, every process p_i that sets its phase value to $\phi_i > \phi_t + 1$ decides v. By the discussion in the first part of the proof there is a subset S of k processes such that every process $p_j \in S$ eventually has phase value $\phi_j > \phi_t + 1$. Consequently, at least k processes eventually decide. □

7 Performance and optimizations

The algorithm guarantees the termination property of consensus in a probabilistic fashion. Since the execution of the algorithm may need to extend for any number of rounds and

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

1 $\phi_i \leftarrow 1$;
2 $v_i \leftarrow proposal_i$;
3 $status_i \leftarrow undecided$;
4 $V_i \leftarrow \emptyset$;
5 **for each** *clock tick* **do**
6 broadcast($\langle \phi_i, v_i, status_i \rangle$);
7 $M_i \leftarrow$ receive();
8 $V_i \leftarrow V_i \cup M_i$;
9 **if** $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| = n$ **then** /* early decision */
10 $status_i \leftarrow decided$;
11 **end**
12 **while** $\exists_{\langle \phi, v, status \rangle \in V_i} : \phi > \phi_i$ **do**
13 $\phi_i \leftarrow \phi$;
14 $v_i \leftarrow v$;
15 $status_i \leftarrow status$;
16 **end**
17 **if** $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
18 **if** $\phi_i \pmod 3 = 1$ **then** /* phase $\phi_i \pmod 3 = 1$ */
19 $v_i \leftarrow$ majority value v in messages with phase $\phi = \phi_i$;
20 **else if** $\phi_i \pmod 3 = 2$ **then** /* phase $\phi_i \pmod 3 = 2$ */
21 **if** $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
22 $v_i \leftarrow v$;
23 **else**
24 $v_i \leftarrow \perp$;
25 **end**
26 **else** /* phase $\phi_i \pmod 3 = 0$ */
27 **if** $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
28 $status_i \leftarrow decided$;
29 **end**
30 **if** $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| \geq 1$ **then**
31 $v_i \leftarrow v$;
32 **else**
33 $v_i \leftarrow \texttt{coin}_i()$;
34 **end**
35 $\phi_i \leftarrow \phi_i + 1$;
36 **end**
37 **if** $status_i = decided$ **then**
38 $decision_i \leftarrow v_i$;
39 **end**
40 **end**

Algorithm 2 Optimized k-consensus algorithm

any process may reach an arbitrarily high phase, eventually there will be a phase where all processes flip the same coin value v and decide (Theorem 3). The expected number of rounds for this to happen is $O(2^n)$ after the system stabilizes at the upper bound of f faults per round.

Nevertheless, a simple inspection of the protocol suffices to observe that the algorithm is fast-learning, i.e., it can decide within two communication rounds in runs with no faults or with certain fault patterns. This is true even if processes have different initial proposal values. As long k processes see the majority of one value during the first phase, they propose the same value for the second phase and then decide.

There are two optimizations, however, that can make the algorithm perform even better in practical scenarios. The first allows termination in one communication round, and the second improves the performance against certain fault patterns. These are discussed below and the modified algorithm is presented in Algorithm 2.

7.1 One-round decision

The algorithm can be modified such that processes decide if they receive, for their current phase ϕ_i, n messages with the same value v (lines 9–11 of Algorithm 2). One-round decision, in particular, can be achieved if every process proposes the same initial value and every message is delivered to at least k processes in the first communication round, then k processes decide by the end of the round. It can be easily

seen by lemma 2 why this does not affect the correctness of the algorithm.

7.2 Three-step variant

When the initial proposal values of the processes are not unanimous, there are certain fault patterns that can make Algorithm 1 terminate in the worst case expected number of phases. The precise conditions for this to happen are: (1) the fault pattern is such that in every round there is no communication from one fixed subset of k processes to a fixed subset of the other $n - k$ processes, and (2) the subset of k processes does not have unanimous proposal values. Under this scenario the algorithm is forced into the worst expected number of phases until a decision is reached, which is $O(2^n)$.

This issue can be overcome by introducing an extra phase in the algorithm, following an approach similar to the local coin protocol of Bracha [5] (lines 18–19 of Algorithm 2). This way, phase $\phi \bmod 3 = 1$ is our new extra phase, phase $\phi \bmod 3 = 2$ is equal to the previous odd phase, and phase $\phi \bmod 3 = 0$ is equal to the previous even phase. In the new phase, processes simply set their proposal value to the value present in a majority of the received messages. The majority can be biased for a predefined proposal value in order to overcome potential ties. This way, the set of k processes converges to the same proposal value in phase $\phi \bmod 3 = 1$, deciding by the end of phase $\phi + 2$ under the considered scenario. It has been demonstrated that the presence of an adversary that enforces a worst-case scheduling against this algorithmic construction is very unlikely to happen in practice [20,21].

8 Conclusions

Despite its usefulness to represent wireless ad-hoc communication environments, research on the communication failure model has been limited. This is related to an associated impossibility result, which states that no agreement is possible in a synchronous system if at every communication round more than $n - 2$ messages can be lost [28,29]. This paper presents a *k-consensus* algorithm tolerant to transmission omission faults, the first to circumvent the Santoro-Widmayer impossibility result using randomization. In a system with n processes, our algorithm makes consensus possible among $k > \frac{n}{2}$ processes. It maintains safety despite an unrestricted number of faults and ensures liveness if the number of omission faults does not exceed $\lceil \frac{n}{2} \rceil (n - k) + k - 2$. Furthermore, the algorithm can be fast learning in the sense that it terminates within two communication rounds under favorable conditions.

Acknowledgments The authors would like to thank the anonymous referees for the constructive criticism provided, which helped to further strengthen the contributions of this paper. A special acknowledgement goes to the insightful reviewer that suggested a solution for the quiescence of the algorithm.

References

1. Aguilera, M., Chen, W., Toueg, S.: Failure detection and consensus in the crash-recovery model. Distrib. Comput. **13**(2), 99–125 (2000)
2. Akkoyunlu, E.A., Ekanadham, K., Huber, R.V.: Some constraints and tradeoffs in the design of network communications. In: Proceedings of the 5th ACM Symposium on Operating Systems Principles, pp. 67–74 (1975)
3. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, pp. 27–30 (1983)
4. Biely, M., Widder, J., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A.: Tolerating corrupted communication. In: Proceedings of the 26th ACM Symposium on Principles of Distributed Computing, pp. 244–253, (2007)
5. Bracha, G.: An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In: Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pp. 154–162 (1984)
6. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. J. Cryptol. **18**(3), 219–246 (2005)
7. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pp. 42–51 (1993)
8. Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
9. Charron-Bost, B., Schiper, A.: The heard-of model: Computing in distributed systems with benign failures. Technical Report LSR-REPORT-2007-001, EPFL (2007)
10. Chockler, G., Demirbas, M., Gilbert, S., Lynch, N., Newport, C., Nolte, T.: Consensus and collision detectors in radio networks. Distrib. Comput. **21**(1), 55–84 (2008)
11. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchronism needed for distributed consensus. J. ACM **34**(1), 77–97 (1987)
12. Dolev, D., Friedman, R., Keidar, I., Malkhi, D.: Failure detectors in omission failure environments. In: Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, pp. 286–295 (1997)
13. Fischer, M.J.: The consensus problem in unreliable distributed systems (A brief survey). In: Karpinsky, M. (ed.) Foundations of Computing Theory, Lecture Notes in Computer Science, vol. 158, pp. 127–140, Springer (1983)
14. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
15. Gray, J.: Notes on data base operating systems. In: Bayer, R., Graham, R.M., Seegmüller, G., (eds) Operating Systems: An Advanced Course, Lecture Notes in Computer Science, vol. 60. Springer (1978)
16. Hurfin, M., Mostefaoui, A., Raynal, M.: Consensus in asynchronous systems where processes can crash and recover. In: Proceedings of the the 17th IEEE Symposium on Reliable Distributed Systems, pp. 280–286 (1998)
17. Lamport L. (2006) Lower bounds for asynchronous consensus. Distrib. Comput. 19(2):104–125
18. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
19. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, (1997)

20. Moniz, H., Neves, N.F., Correia, M., Veríssimo, P.: Experimental comparison of local and shared coin randomized consensus protocols. In: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems, pp. 235–244 (2006)
21. Moniz, H., Neves, N.F., Correia, M., Veríssimo, P.: RITAS: Services for randomized intrusion tolerance. IEEE Transactions on Dependable and Secure Computing, (to appear) (2010)
22. Neves, N.F., Correia, M., Veríssimo, P.: Solving vector consensus with a wormhole. IEEE Trans. Parallel. Distrib. Syst. **16**(12), 1120–1131 (2005)
23. Oliveira, R., Guerraoui, R., Schiper, A.: Consensus in the crash-recover model. Technical Report, pp. 97–239, EPFL (1997)
24. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
25. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Softw. Eng. **12**(3), 477–482 (1986)
26. Rabin, M.O.: Randomized Byzantine generals. In: Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, pp. 403–409 (1983)
27. Raynal, M., Roy, M.: A note on a simple equivalence between round-based synchronous and asynchronous models. In: Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 387–392 (2005)
28. Santoro, N., Widmayer, P.: Time is not a healer. In: Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science, pp. 304–313 (1989)
29. Santoro, N., Widmayer, P.: Agreement in synchronous networks with ubiquitous faults. Theor. Comput. Sci. **384**(2–3), 232–249 (2007)
30. Schmid, U., Weiss, B., Keidar, I.: Impossibility results and lower bounds for consensus under link failures. SIAM J. Comput **38**(5), 1912–1951 (2009)
31. Varghese, G., Lynch, N.A.: A tradeoff between safety and liveness for randomized coordinated attack. Inf. Comput. **128**(1), 57–71 (1996)