Theoretical Computer Science 709 (2018) 80–97

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Randomized k-set agreement in crash-prone and Byzantine asynchronous systems [☆]

Achour Mostéfaoui [a], Hamouma Moumen [b], Michel Raynal [c,d,*]

[a] LINA, Université de Nantes, 44322 Nantes, France
[b] University of Batna, Algeria
[c] Institut Universitaire de France, France
[d] IRISA, Université de Rennes, 35042 Rennes, France

ARTICLE INFO

Article history:
Received 21 April 2016
Received in revised form 17 March 2017
Accepted 23 March 2017
Available online 25 March 2017

Keywords:
Asynchronous system
Broadcast abstraction
Byzantine process
Coin
Crash failure
Distributed algorithm
k-Set agreement
Message-passing system
Randomized algorithm
Signature-free algorithm

ABSTRACT

k-Set agreement is a central problem of fault-tolerant distributed computing. Considering a set of n processes, where up to t may commit failures, let us assume that each process proposes a value. The problem consists in defining an algorithm such that each non-faulty process decides a value, at most k different values are decided, and the decided values satisfy some context-depending validity condition. Algorithms solving k-set agreement in synchronous message-passing systems have been proposed for different failure models (mainly process crashes, and process Byzantine failures). Differently, k-set agreement cannot be solved in failure-prone asynchronous message-passing systems when $t \geq k$. To circumvent this impossibility an asynchronous system must be enriched with additional computational power.

Assuming $t \geq k$, this paper presents two distributed algorithms that solve k-set agreement in asynchronous message-passing systems where up to t processes may commit crash failures (first algorithm) or more severe Byzantine failures (second algorithm). To circumvent k-set agreement impossibility, this article considers that the underlying system is enriched with the computability power provided by randomization. Interestingly, the algorithm that copes with Byzantine failures is signature-free, and ensures that no value proposed only by Byzantine processes can be decided by a non-faulty process. Both algorithms share basic design principles.

1. Introduction

Distributed agreement in the presence of process failures. The world is distributed and more and more applications are now distributed. Moreover, when considering the core of non-trivial distributed applications, it appears that the computing entities (processes) have to agree in one way or another, for example to take a common decision, execute specific actions, or validate some commitment. Said another way, agreement problems lie at the core of distributed computing.

The most famous distributed agreement problem is the *consensus* problem. Let us consider a set of processes, where some of them may commit failures. Assuming each process proposes a value, the consensus problem is defined by the

[☆] Parts of the work presented in this paper appeared in the ACM conferences SPAA 2001 and ICDCN 2016 [24,27].
[*] Corresponding author.
E-mail address: raynal@irisa.fr (M. Raynal).

http://dx.doi.org/10.1016/j.tcs.2017.03.018
0304-3975/© 2017 Elsevier B.V. All rights reserved.

following properties: each non-faulty process must decide a value (termination), such that the same value is decided by the non-faulty processes (agreement), and this value satisfies some validity condition, which depends on the proposed values and the considered failure model [13,29].

The k-set agreement problem is a natural weakening of consensus [10]. It allows the non-faulty processes to decide different values, as long as no more than k values are decided (the problem parameter k can be seen as the coordination degree imposed to processes). Hence, consensus is 1-set agreement. Let us notice that k-set agreement can be easily solved in crash-prone systems where k (the maximal number of different values that can be decided) is greater than t (the maximal number of processes that may be faulty). The k-set agreement problem has applications, e.g., to compute a common subset of wavelengths (each process proposes a wavelength and at most k of them are selected), or to duplicate k state machines where at most one is required to progress forever [16,35].

Crash and Byzantine failures. A process crash failure occurs when a process stops prematurely. After it crashed, a process never recovers; moreover it behaves correctly (i.e., according to its code) before crashing. A crash failure can be seen as a benign failure, as a crashed process did not pollute the computation before crashing (e.g., by disseminating fake values).

The situation is different with Byzantine failures. This failure type has been introduced in the context of synchronous distributed systems [21,29,33], and then investigated in the context of asynchronous distributed systems [2,22,34]. A process has a *Byzantine* behavior when it arbitrarily deviates from its intended behavior. We then say that it "commits a Byzantine failure" (otherwise we say the process is *non-faulty* or *correct*). This bad behavior can be intentional (malicious) or simply the result of a transient fault that altered the local state of a process, thereby modifying its behavior in an unpredictable way. Let us notice that, from a failure hierarchy point of view, process crashes (unexpected halting) constitute a strict subset of Byzantine failures. As asynchronous message-passing systems are more and more pervasive, the assumption "no process has a bad behavior" is no longer sensible. Hence, agreement in asynchronous Byzantine message-passing systems is becoming a more and more important issue of fault-tolerance.

An impossibility result and how to cope with it. Let us consider a system made up of n processes, where up to t may be faulty. Whatever the value of k (with respect to t), k-set agreement can always be solved if the system is synchronous [33]. The situation is different in asynchronous systems where k-set agreement is impossible to solve in the process crash failure model when $k \leq t$ [5,19,37]. As Byzantine failures are more severe than crash failures, this impossibility remains true in asynchronous Byzantine systems.

It follows from this impossibility that, when $k \leq t$, either the space of values that can be proposed must be restricted [14,25], or the underlying asynchronous distributed system must be enriched with additional computational power for k-set agreement to be solved. Such an additional computational power can be provided with partial synchrony assumptions (e.g., [11,39] which consider $k = 1$), minimal synchrony assumptions (e.g., [6] which considers $k = 1$ and Byzantine failures), appropriate failure detectors (e.g., [9,26] which consider $k = 1$ and crash failures, and [15] which considers $k = 1$ and Byzantine failures), or randomization (e.g., [3] which considers $k = 1$ and crash failures, [31] which considers $k = 1$ and Byzantine failures, [8] which considers $k \leq t$ and crash failures in read/write shared memory systems, and [27] which considers $k \leq t$ and crash failures in message-passing systems).

Intrusion-tolerant agreement with respect to Byzantine processes. The validity property associated with a distributed agreement problem relates its outputs to its inputs. As no process creates fake values in a crash-prone system, the k-set agreement validity property is easy to state, namely, a decided value must be a value proposed by a process. In a system where processes may commit Byzantine failures, there is no way to direct a Byzantine process to decide some specific value. Consequently the k-set agreement validity property can only be on the values decided by the correct processes. Moreover, the notion of a "value proposed by a faulty process" is dubious.

A classical validity property for Byzantine consensus (see, e.g., [22]) states that, if all the non-faulty processes propose the same value, they must decide it. Hence, as soon as two non-faulty processes propose different values, any value can be decided by the correct processes, even a value "proposed" by a Byzantine process. (Let us observe that a Byzantine process can appear as proposing different values to different correct processes.) More generally, and as noticed and deeply investigated in [30], it follows that the solvability of Byzantine k-set agreement is sensitive to the particular validity property that is considered.

This paper considers the following validity property (introduced in [28] where it is called *intrusion-tolerance*): no value proposed only by Byzantine processes can be decided by a non-faulty process. One way to be able to design a k-set algorithm providing this property, consists in allowing a non-faulty process to decide a default value $\perp$, except (to prevent triviality) when the non-faulty processes propose the same value. (The $\perp$ decision at some non-faulty processes can occur for example in the adversary scenario where the non-faulty processes propose different values, while the Byzantine processes propose the same value.) Another way to design a k-set algorithm providing intrusion-tolerance consists in adding a constraint on the total number of different values that can be proposed by the non-faulty processes. Let $m \geq 2$ be this number. It is shown in [18] that, in an n-process system where up to t processes may commit Byzantine failures, such a constraint is $n - t > mt$ (i.e., there is a value proposed by at least $(t + 1)$ non-faulty processes).

Content of the paper. This paper is on k-set agreement in n-process asynchronous message-passing systems, where $k \leq t$. It presents two algorithms. The first is a k-set agreement algorithm for asynchronous message-passing systems where up to

$t < n/2$ processes may commit crash failures. This algorithm, which is relatively simple, is based on the reliable broadcast abstraction and randomization (local random coins with several sides). For the reasons explained in Section 3, this algorithm assumes also $t < n - k\lfloor \frac{n}{k+1} \rfloor$.

The second algorithm (which constitutes the main part of the paper) is a signature-free intrusion-tolerant k-set agreement algorithm for asynchronous message-passing systems where up to $t < n/3$ processes may commit Byzantine failures.

When focusing on k-set agreement in the context of Byzantine failures, the paper has two main contributions.

- The first is a pair of all-to-all communication abstractions. The first one, called MV-broadcast (where MV stands for "Multivalued Validated"), allows the non-faulty processes to exchange values in such a way that all the non-faulty processes eventually obtain the same set of values, and none of these values is from Byzantine processes only. The second one, called SMV-broadcast (where S stands for "Synchronized") is built on top the first one, and is such that, if a non-faulty process obtains a set with a single value, the set obtained by any other non-faulty process contains this value. The important point is that these communication abstractions allow the processes to exchange values while eliminating the values sent only by Byzantine processes. They generalize to the "multivalue" case the communication abstractions introduced in [23], where the set of values that the processes exchange is limited to two values.
  Independently from their use in this paper, these all-to-all communication abstractions are interesting on their own, and could be used to solve other problems.
- The second is the k-set agreement algorithm for asynchronous message-passing systems where processes may commit Byzantine failures. This algorithm, which is round-based, is built in a very modular way. It relies on the previous SMV-broadcast abstraction, and on the additional computational power supplied by local multi-sided random coins. As far as we know, this is the first randomized k-set agreement algorithm for asynchronous Byzantine message-passing systems. The fact that this algorithm is also signature-free has a strong consequence: the "Byzantine adversary" is not required to be computationally bounded.
  This algorithm assumes $t < n/(m+1)$, where m is the number of different values that can be proposed. As shown in [18,28], this is a necessary and sufficient condition when one wants that a correct process always decides a value proposed by at least one correct process.

Roadmap. The paper is composed of two parts. The first part (Sections 2 and 3) addresses crash failures. Section 2 presents the crash failure model, the reliable broadcast abstraction, the notion of a local random coin, and a definition of k-set agreement suited to this model. Assuming $t < n/2$, Section 3 presents a k-set agreement algorithm suited to this asynchronous model.

The second part (Sections 4, 5, and 6) addresses Byzantine failures. Section 4 presents the Byzantine failure model, the no-duplicity broadcast abstraction, and a definition of intrusion-tolerant k-set agreement suited to this model. Section 5 introduces two new all-to-all communication abstractions (MV-broadcast and SMV-broadcast), which are at the core of the Byzantine-tolerant k-set agreement algorithm presented in Section 6. Finally, Section 7 concludes the paper.

Remark on the reading of this paper. The variables with the same meaning in both k-set agreement algorithms (the one suited to crash failures and the one suited to Byzantine failures) have been given the same names. Nevertheless, to facilitate the understanding of each algorithm independently from the other, some discussions are "repeated" in the presentation of each algorithm (e.g., the presentation of the constants W and R).

2. Asynchronous model with crashes failures, and definitions

2.1. Computation model

Asynchronous processes. The system is made up of a finite set Π of $n > 1$ asynchronous sequential processes, namely $\Pi = \{p_1, \dots, p_n\}$. "Asynchronous" means that each process proceeds at its own pace, which may vary arbitrarily with time, and remains always unknown to the other processes.

Communication network. The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. "Asynchronous" means that a message is eventually received by its destination process, i.e., there is no bound on message transfer delays. "Reliable" means that the network does not loss, duplicate, modify, or create messages. "Point-to-point" means that there is a bi-directional communication channel between each pair of processes. Hence, when a process receives a message, it can identify its sender.

A process p_i sends a message to a process p_j by invoking the primitive operation send TAG(m) to p_j, where TAG is the type of the message and m its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process receives a message by executing the primitive "receive()".

The operation broadcast TAG(m) is a macro-operation which stands for "**for each** $j \in \{1, \dots, n\}$ send TAG(m) to p_j **end for**". This operation is usually called *unreliable* broadcast (if the sender crashes while executing the **for** loop, it is possible that only an arbitrary subset of correct processes receives the message).

Failure model. Up to t processes may crash during an execution. As already indicated in the Introduction, before a process (possibly) crashes, it executes its code as defined by its local algorithm, and no crashed process recovers. A crash is consequently a definitive halting.

Given an execution, a process that crashes is said to be *faulty* in this execution, otherwise it is *correct* or *non-faulty*. Hence, before a process crashes, no one knows if it is correct or faulty.

Random multi-sided local coin. Each process p_i is endowed with an operation denoted random(). Each invocation of this operation takes a non-empty set X as input parameter and returns a value of X with probability $1/|X|$. As we will see in Section 3, equipping each process with such a local random coin provides an additional computational power that allows k-set agreement to be solved. (For the interested reader, a discussion on local coins vs global coins in Byzantine Agreement is presented in [20].)

Notation. This computation model is denoted $\mathcal{CAMP}_{n,t}[\emptyset]$ (CAMP stands for "Crash-prone Asynchronous Message Passing"). In the following, this model is both restricted with a constraint on t and enriched with random multi-sided local coins, which provide the processes with additional computational power. More precisely, $\mathcal{CAMP}_{n,t}[t < n/\alpha]$ (where α is a positive integer) denotes the model $\mathcal{CAMP}_{n,t}[\emptyset]$ where the maximal number of faulty processes is smaller than n/α. $\mathcal{CAMP}_{n,t}[t < n/\alpha, \text{LRC}]$ denotes the model $\mathcal{CAMP}_{n,t}[t < n/\alpha]$ where each process is enriched with a local multi-sided random coin. Let us notice that, as LRC belongs to the model, it is given for free in $\mathcal{CAMP}_{n,t}[t < n/\alpha, \text{LRC}]$.

Time complexity. When computing the time complexity, we ignore local computation time, and consider the longest sequence of causally related messages $m_1, m_2, \ldots, m_z$ (i.e., for any $x \in [2..z]$, the reception of m_{x-1} is a requirement for the sending of m_x). The size of such a longest sequence defines the time complexity.

2.2. Reliable broadcast abstraction

This reliable broadcast communication abstraction (in short R-Broadcast) provides the processes with two operations, denoted R_broadcast() and R_deliver(). When a process invokes R_broadcast TAG(m), we say that it "r-broadcasts" the message whose type is TAG and value is m. Similarly, when a process returns from the invocation of R_deliver() we say that it "r-delivers" a message. Reliable broadcast is defined by the following properties [7,17].

- R-Validity. If a process r-delivers TAG(m) from a process p_j, p_j invoked R_broadcast TAG(m).
- R-Integrity. A process r-delivers at most once a message TAG(m) from a sender p_i.
- R-Termination. If a correct process r-broadcasts a message TAG(m), or a correct process r-delivers the message TAG(m), then all correct processes r-deliver the message TAG(m).

Validity relies the outputs to the inputs (no spurious messages). Assuming no process r-broadcasts several times the same message (which can be easily implemented by associating a new sequence number with each message r-broadcast by a process), Integrity states there is no duplication. Finally, Termination states the conditions under which a message must be r-delivered by all correct processes, namely, either when its sender is correct, or when at least one correct process r-delivered it.

It is easy to see that, all correct processes r-deliver the same set of messages M, and this set contains all the messages they r-broadcast. Moreover, a faulty process r-delivers a subset of M, but two faulty processes can r-deliver (before crashing) two sets of messages $M1$ and $M2$ such that none of $M1$ and $M2$ contains the other set.

Implementations of R-Broadcast can be easily designed in $\mathcal{CAMP}_{n,t}[\emptyset]$. A very simple (but inefficient) one is the following. When, at the implementation level, a process receives for the first time a copy of the message TAG(m), it first forwards it to all the other processes, and only then r-delivers it. According to the underlying topology and the way message identifiers are built, more efficient implementations can be designed (e.g., [32,36]).

2.3. k-Set agreement

The k-agreement problem was introduced in [10] in the context of the model $\mathcal{CAMP}_{n,t}[\emptyset]$. It consists in implementing an operation denoted propose$_k$() satisfying the properties stated below. This operation takes an input parameter, and returns a value. When a process invokes propose$_k(v)$, we say that it "proposes value v". When a process returns from propose$_k$() with the value w, we say that it "decides w". It is assumed that at least the correct processes invoke propose$_k$(). The properties defining k-set agreement are the following.

- C-KS-Validity. It a process decides v, there is a process that proposed v.
- C-KS-Agreement. At most k different values are decided.
- C-KS-Termination. Any correct process decides a value.

As before, Validity relies the outputs to the inputs. Agreement defines a coordination constraint on the processes. Termination states that at least the processes that do not crash decide.

```
operation propose$_k(v_i)$ is
(1)  $val_i \leftarrow [\bot, \ldots, \bot]$; $r_i \leftarrow 0$; $est_i \leftarrow v_i$; R_broadcast VAL$(v_i)$;
(2)  while $true$ do $r_i \leftarrow r_i + 1$; % round $r_i = r$ %
// ————- phase 1 of round $r_i$: From up to $n$ values to up to $k$ values plus possibly $\bot$ ———-
(3)    broadcast PHASE1$(r_i, est_i)$;
(4)    wait (PHASE1$(r_i, -)$ received from $R = k\lfloor \frac{n}{k+1} \rfloor + 1$ processes);
(5)    if $(\exists v \mid W = \lfloor \frac{n}{k+1} \rfloor + 1$ PHASE1$(r_i, v)$ messages have been received)
(6)                then $ph2\_est_i \leftarrow v$ else $ph2\_est_i \leftarrow \bot$ end if;
// ————- phase 2 of round $r_i$: Try to decide on one of at most $k$ values —————————
(7)    broadcast PHASE2$(r_i, ph2\_est_i)$;
(8)    wait (PHASE2$(r_i, ph2\_est)$ received from $maj = \lfloor \frac{n}{2} \rfloor + 1$ processes);
(9)    let $ph2\_rec_i = \{$ $ph2\_est$ such that PHASE2$(r_i, ph2\_est)$ has been received $\}$;
(10)   case $ph2\_rec_i = \{\bot\}$            then $est_i \leftarrow val_i$[random([1..n])]
(11)        $\bot \notin ph2\_rec_i$            then let $v$ be any value $\in ph2\_rec_i$; R_broadcast DEC$(r_i, v)$
(12)        $ph2\_rec_i = \{\bot, v, \ldots\}$  then $est_i \leftarrow$ any non-$\bot$ value $\in ph2\_rec_i$
(13)   end case
(14) end while.

(15) when VAL$(v)$ is r-delivered from $p_j$ do $val_i[j] \leftarrow v$.

(16) when DEC$(r, v)$ is r-delivered from $p_j$ do return$(v)$.
```

Fig. 1. Solving k-set agreement in $\mathcal{CAMP}_{n,t}[t < \min(n/2, n - k\lfloor \frac{n}{k+1} \rfloor)$, LRC] (Algorithm 1).

3. Crash model: a randomized k-set agreement algorithm

This section presents an algorithm which solves the k-set agreement problem in the system model $\mathcal{CAMP}_{n,t}[t < \min(n/2, n - k\lfloor \frac{n}{k+1} \rfloor)$, LRC]. Algorithm 1 is a round-based algorithm, which means that the processes execute a sequence of asynchronous rounds.[1]

As we are interested in a randomized algorithm to solve k-set agreement, the Termination property is weakened as follows [3,31]: any correct process decides with probability 1. In the context of round-based algorithms, this property can be re-stated as follows, where p_i is any correct process:

$$\text{C-KS-P-Termination:} \quad \lim_{r \to +\infty} \big(\text{Probability } [p_i \text{ decides by round } r]\big) = 1.$$

3.1. Description of the algorithm

Each process p_i starts Algorithm 1 by invoking propose$_k(v_i)$, where v_i is the value it proposes. It decides a value when it executes the statement return(v); v is then the value it decides. Moreover, when it executes return(), a process terminates its participation to the algorithm. $\bot$ denotes a default value that no process can propose. It is used during each round to restrict the set of proposed values to a set of at most k values.

Algorithm *1* is described in Fig. 1. Each process manages a local variable est_i, which represents the current estimate of its decision value. Initially, est_i is set to v_i (the value proposed by p_i). Process p_i manages also a local array $val_i[1..n]$, initialized to $[\bot, ..., \bot]$.

Dissemination of the proposed values. When, it starts, a process p_i first r-broadcasts the value it proposes (line 1). When, it r-delivers the value proposed by p_j, p_i saves it in $val_i[j]$ (line 16). Let us notice that, due to the Validity and Termination properties of R-broadcast, the arrays $val[1..n]$ of the correct processes eventually (a) contain at least the values proposed by each correct process, and (b) become equal.

A sequence of asynchronous rounds. The processes execute a sequence of asynchronous rounds to converge to a set of at most k values. Each round is made up of two communication phases (hence it costs two communication steps). The aim of the first phase (lines 3–6) is to force each process to adopt either a value from a set of at most k different values, or the default value $\bot$. The aim of the second phase (lines 7–13) is to allow processes to decide non-$\bot$ values that have been previously adopted, while ensuring that (if processes decide during distinct rounds) no more than k different values will eventually be decided (i.e., the Agreement property is not violated).

Let us notice that, differently from the R-broadcast used at lines 1 and 11, the broadcast operation used at lines 3 and 7 is the unreliable macro-operation multi-send defined in Section 2.1.

[1] Differently from round-based synchronous algorithms where the progress from a round to the next one is a built-in property provided by the model, in an asynchronous system it is to the processes to implement the progress of a round to the next one.

First phase of a round r. The processes first exchange their current estimate values (lines 3–4). Let us note that, as far the round r is concerned, a message PHASE1(r, v) can be interpreted as a vote for the value v. Accordingly, a process p_i adopts a value if has received enough votes for it, say W votes. If, among the values it has received, none has enough votes to be adopted, p_i adopts the default value $\bot$. The adopted value is kept in $ph2\_est_i$ (line 6).

The aim is to have at most k different values adopted by the processes at the end of the first phase. In order to attain this goal, we must have $(k+1)W > n$ (as there are only n processes, $k+1$ values cannot each obtain W votes). This means that $W = \lceil \frac{n+1}{k+1} \rceil = \lfloor \frac{n}{k+1} \rfloor + 1$.

Let us now examine how many messages PHASE1(r, v) a process has to wait for (at line 4) before adopting a value (line 6) in order to have a chance to adopt a value initially proposed by a process (i.e., a value different from $\bot$). Let R be this number. Considering the case where p_i adopts a non-$\bot$ value, let us examine the worst situation: p_i can receive $(W-1)$ votes for $(k-1)$ different values, and only then receive W votes for the value v it adopts. Hence, $R = (W-1)(k-1) + W = (W-1)k + 1$. Moreover, in order that no process blocks at line 4, we must have $R \le n - t$ which is equivalent to $t < n - k\lfloor \frac{n}{k+1} \rfloor$.

Hence, at the end of the first phase, the set of the local variables $ph2\_est_i$ contains at most k values, plus possibly $\bot$. The aim of the second phase is to allow each process to decide one of these non-$\bot$ values in such a way that the Agreement property be not violated even if processes decide during different rounds.

Second phase of a round r. During the second phase, the processes exchange the values they have previously adopted. A process p_i waits for messages PHASE2() from a majority of processes (lines 7–8). As shown at line 9, $ph2\_rec_i$ is the set of values received by p_i. Let us notice that if v $(\neq \bot)$ belongs to $ph2\_rec_i$, then v was the estimate of at least W processes at the beginning of the current round. There are three cases determined by the content of $ph2\_rec_i$.

- If $\bot \notin ph2\_rec_i$, p_i can decide any value v of this set (line 11). It then r-broadcasts the message DEC(v). If p_i does not crash, this message will be r-delivered at all the non-crashed processes, which (if they do not have yet decided) will decide v at line 15.
- If $ph2\_rec_i$ contains both $\bot$ and non-$\bot$ values, p_i updates its estimate est_i to any non-$\bot$ value of $ph2\_rec_i$, and proceeds to the next round.
- If $ph2\_rec_i$ contains only the default value $\bot$, p_i updates its current estimates est_i to a randomly chosen value (line 10), and then proceeds to the next round. Actually, p_i selects randomly a process identity (say x) and sets est_i to $val_i[x]$. Let us note that $val_i[x]$ is equal to the value proposed by p_x or $\bot$. The randomness of the choices guarantees that eventually there are rounds during which p_i selects non-$\bot$ entries of its array $val_i[1..n]$.

It is important to observe that, as soon as a process returned from the R-broadcast of line 11, all correct processes will eventually return a value. Said, differently, no deadlock is possible as soon as a process has executed line 11.

Deterministic behavior. Let us point out a nice feature of Algorithm 1. When processes collectively propose at most k values, or said differently when the cardinal of the set of all proposed values is at most k, the algorithm terminates deterministically at the first round without executing the random statement line 10. Indeed, in such a situation, there exists at least one value that satisfies the predicate of line 5 and all the processes that do not crash execute line 11 as no process keeps the default value $\bot$.

3.2. Proof of the algorithm

As announced previously, the proof assumes $t < n/2$ and $t < n - k\lfloor \frac{n}{k+1} \rfloor$.

Lemma 1. *If no process decides during a round $r' \le r$, all correct processes will start round $r+1$.*

Proof. The proof is by contradiction. Let r be the first round during which a correct process p_i blocks forever. It does it in a wait() statement at line 4 or 8.

Due to the assumption, all the correct processes start round r, and consequently send a message PHASE1$(r, -)$. As there are at least $(n-t)$ correct processes, and $n - t \ge R$, it follows that p_i receives at least R messages PHASE1$(r, -)$, and p_i cannot block forever at line 4. Moreover, it follows that each correct process sends a message PHASE2$(r, -)$ at line 7.

As $\forall k > 0 : (n - k\lfloor \frac{n}{k+1} \rfloor) \le \frac{n}{2}$, and $R = k\lfloor \frac{n}{k+1} \rfloor + 1$, it follows that $n - (R-1) \le \frac{n}{2}$, from which $R > \frac{n}{2}$ follows. Hence, every correct process p_i receives a message PHASE2$(r, -)$ from a majority of processes. It follows that it cannot block forever at line 8. □

Lemma 2. *Let $EST[r]$ be the set of the estimate values of the processes that start round r. If $\bot \notin EST[r]$ and $|EST[r]| \le k$, any process that starts round r decides during this round, unless it crashes.*

Proof. As by assumption there are no more than k different estimates values (all different from $\bot$) at the beginning of r, the messages PHASE1$(r, -)$ carry at most k different values. As $R = (W-1)k + 1$, it follows that any process p_i (that executes line 5 during r) selects one of these values (say v) to update $ph2\_est_i$ at line 6. Said in another way, no local variable $ph2\_est$ is set to $\perp$. It follows that $\forall i$: $\perp \notin ph2\_rec_i$. Consequently, any process p_i can only execute line 11 and decide. □

Lemma 3. *Let $PH2\_EST[r]$ be the set including the values of all the $ph2\_est_i$ local variables at the end of the first phase of r (i.e., just after line 6). $PH2\_EST[r]$ contains at most k values, plus possibly the default value $\perp$.*

Proof. Let us assume that $PH2\_EST[r]$ contains $(k+1)$ non-$\perp$ values. If a value belongs to this set (hence, is the value of a local variable $ph2\_est_i$), it has been received (by some p_i) from at least W processes (see line 5). Moreover, each process sends only one message PHASE1() per round, which carries a single value (line 3). It follows that $(k+1)W$ processes have sent messages PHASE1(). As $(k+1)W > n$, this is impossible. □

Lemma 4. *A decided value is a proposed value.*

Proof. The proof follows from the observation that a decided value is an estimate different from $\perp$, and it follows from the code that any estimate variable est_i can only contain a proposed value or $\perp$. □

Lemma 5. *Every correct process eventually decides with probability* 1.

Proof. Let us remark that, if a process decides (executes return() at line 16), all correct processes decide. This follows from the Termination property of the R-broadcast abstraction used to disseminate decided values (lines 11 and 16).

The proof is by contradiction. Let us assume that no process decides. There is a time τ after which:

(H1) There are only correct processes executing the algorithm, and
(H2) the arrays $val[1..n]$ of the correct processes are equal.

This is a direct consequence of the fact that these arrays are filled in with values that are disseminated with the R-broadcast abstraction. If both p_i and p_j are correct, then if the value v_k is r-delivered by p_i, it is also r-delivered by p_j. Hence after τ, $val_i[k] = v_k$ implies $val_j[k] = v_k$.

Let us first note that, as no process decides, no correct process blocks forever in a round (Lemma 1). Moreover, no process executes line 11. Hence, at every round r after τ, a process executes line 10 or line 12. We consider three cases.

- Case 1: All the processes that execute r, execute line 12.
  So, all the processes set their estimates to a non-$\perp$ value. Due to Lemma 3, there are no more than k different estimate values. Hence, all the processes that start the round $(r+1)$ do it with at most k different estimates, no one being equal to $\perp$. Due to Lemma 2, they decide.
- Case 2: During r at least one process (but not all) executes line 12.
  In this case, due to Lemma 3, each process p_i that executes line 12 sets its local variable est_i to a non-$\perp$ value taken from a set (namely, $PH2\_EST[r]$) that includes at most k non-$\perp$ values. The other processes execute the line 10. There is a probability (> 0) such that each of these processes sets its estimate variable to a non-$\perp$ value $\in PH2\_EST[r]$.
- Case 3: During round r no process executes line 12.
  In this case, all the processes execute line 10. There is a probability (> 0) that they get no more than k different estimate values (all different from $\perp$).

In Case 1, the termination is obtained. Let us consider Case 2 and Case 3. During any round after τ, there is a probability $p > 0$ that there are at most k estimate values, each different from $\perp$. Hence, there is a probability $P(\alpha) = p + p(1-p) + p(1-p)^2 + \cdots + p(1-p)^{\alpha-1} = 1-(1-p)^\alpha$ that, after at most α rounds, the processes have no more than k estimate values, each different from $\perp$. As $\lim_{\alpha\to\infty} P(\alpha) = 1$, it follows that, with probability 1, all processes will start a round with no more than k estimate values, each different from $\perp$. Then, according to Lemma 2, they will decide. □

Lemma 6. *No more than k different values are decided.*

Proof. When a process decides at line 16 due to a message DEC(r, v), we say that it decides at round r. This is because the value v was computed at round r by some process.

Let r be the first round during which processes decide. They decide because some processes issued an R-broadcast at line 11. Due to Lemma 3, the set $PH2\_EST[r]$ contains at most k non-$\perp$ values. Moreover, it follows from line 11, that a process that decides can only decide one of those at most k non-$\perp$ values.

Let us now consider a process p_j that proceeds to round $(r+1)$. We claim (proof below) that its estimate est_j is updated to a value of the set $PH2\_EST[r]$ before it proceeds to the round $(r+1)$. From this claim we conclude that, after round r, a value $\notin PH2\_EST[r]$ cannot be the value of a local variable est_j. Hence, any future value r-broadcast at line 11 (and consequently, potentially decided) can be one of the at most k non-$\perp$ values of $PH2\_EST[r]$.

Proof of the claim. Let p_i be a process that issues R_broadcast DEC(r_i, v) at line 11 of round r $(r_i = r)$. This means that p_i computed v at this line. Hence, $\perp \notin ph2\_rec_i$. Moreover, the set $ph2\_rec_i$ contains only values coming from the local variables $ph2\_est$ of a majority of processes (line 9). Let p_j be a process that progresses to round $(r+1)$. As p_j has also received estimate values from a majority of processes (line 9), we have $ph2\_rec_i \cap ph2\_rec_j \neq \emptyset$. Hence, $ph2\_rec_j \neq \{\perp\}$. It follows that p_j has not executed line 10 before progressing to the round $(r+1)$. It has necessarily executed line 12. Consequently it updated est_j to one of the at most k different non-$\perp$ values of $PH2\_EST[r]$. End of the proof of the claim. $\square$

Theorem 1. *Algorithm 1 solves the k-set agreement problem in* $\mathcal{CAMP}_{n,t}[t < \min(n/2, n - k\lfloor \frac{n}{k+1} \rfloor), \text{LRC}]$.

Proof. The proof follows from Lemma 4 (Validity), Lemma 5 (P-Termination), and Lemma 6 (Agreement). $\square$

4. Asynchronous model with Byzantine failures, and definitions

4.1. Computation model

From $\mathcal{CAMP}_{n,t}$ to Byzantine failures. The computation model is the asynchronous message passing model presented in Section 2 enriched with local random coins (LRC). It differs only in the nature of process failures.

Failure model. Up to t processes may exhibit a *Byzantine* behavior [21,29]. A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *correct* or *non-faulty*. A Byzantine process is a process that behaves arbitrarily: it may crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. As a simple example, a Byzantine process, which is assumed to send a message m to all the processes, can send a message m_1 to some processes, a different message m_2 to another subset of processes, and no message at all to the other processes. More generally, a Byzantine process has an unlimited computational power, and Byzantine processes can collude to "pollute" the computation. Let us notice that, as each pair of processes is connected by a channel, no Byzantine process can impersonate another process, but Byzantine processes are not prevented from influencing the delivery order of messages sent to correct processes.

Discarding messages from Byzantine processes. If, according to its algorithm, a process p_j is assumed to send a single message TAG() to a process p_i, then p_i processes only the first message TAG(v) it receives from p_j. This means that, if p_j is Byzantine and sends several messages TAG(v), TAG(v') where $v' \neq v$, etc., all of them except the first one are discarded by their receivers. (Let us observe that this does not prevent multiple copies of the first message TAG() to be received and processed by their receiver.)

Notation. This computation model is denoted $\mathcal{BAMP}_{n,t}[\emptyset]$ (BAMP stands for "Byzantine Asynchronous Message Passing"). As for $\mathcal{CAMP}_{n,t}[\emptyset]$, this basic model is both restricted with a constraint on t and enriched with local coins. It is consequently denoted $\mathcal{BAMP}_{n,t}[t < n/\alpha, \text{LRC}]$, where $\alpha \geq 1$.

4.2. The no-duplicity broadcast abstraction

The following broadcast abstraction will be a basic component used in the all-to-all SMV-broadcast abstraction presented in Section 5 (which is the communication abstraction on which is built the Byzantine-tolerant k-set algorithm presented in Section 6).

Definition of the ND-broadcast communication abstraction. This abstraction was introduced by S. Toueg in [39]. It is defined by two operations denoted ND_broadcast() and ND_deliver(), which allow the processes to eliminate one bad behavior of Byzantine processes. More precisely, a Byzantine process is prevented from sending different messages to different correct processes, while it is assumed to send the very same message to all of them.

As previously, when a process invokes ND_broadcast TAG() we say that it "ND-broadcasts" a message, and when it invokes ND_deliver() we say that it "ND-delivers" a message. Considering an instance of ND-broadcast where the operation ND_broadcast TAG() is invoked by a process p_i, this communication abstraction is defined by the following properties.

- ND-Validity. If a non-faulty process ND-delivers a message from p_i, then, if it is non-faulty, p_i ND-broadcast this message.
- ND-No-duplicity. No two non-faulty processes ND-deliver distinct messages from p_i.
- ND-Termination. If the sender p_i is non-faulty, all the non-faulty processes eventually ND-deliver its message.

Let us observe that, if the sender p_i is faulty, it is possible that some non-faulty processes ND-deliver a message from p_i while others do not ND-deliver a message from p_i. As already indicated, the no-duplicity property prevents non-faulty processes from ND-delivering different messages from a faulty sender.

```
operation ND_broadcast MSG($v_i$) is
(1)  broadcast ND_INIT($i, v_i$).

when ND_INIT($j, v$) is delivered do
(2)  if (first reception of ND_INIT($j, -$)) then broadcast ND_ECHO($j, v$) end if.

when ND_ECHO($j, v$) is delivered do
(3)  if (ND_ECHO($j, v$) received from $(n-t)$ different processes and MSG($j, v$) not yet ND_delivered)
(4)    then ND_deliver MSG($j, v$)
(5)  end if.
```

Fig. 2. Implementing ND-broadcast in $\mathcal{BAMP}_{n,t}[t < n/3]$ (Algorithm 2) [39].

An algorithm implementing ND-broadcast. It is shown in [39] that $t < n/3$ is a necessary requirement to implement ND-broadcast in a Byzantine asynchronous message-passing system. Algorithm 2 (from [39]) implements ND-broadcast in $\mathcal{BAMP}_{n,t_{n,t}}[t < n/3]$.

When a process p_i wants to ND-broadcast a message whose content is v_i, it broadcasts the message ND_INIT(i, v_i) (line 1). When a process receives a message ND_INIT$(j, -)$ for the first time, it broadcasts a message ND_ECHO(j, v) where v is the data content of the ND_INIT() message (line 2). If the message ND_INIT(j, v) received is not the first message ND_INIT$(j, -)$, p_j is Byzantine and the message is discarded. Finally, when p_i has received the same message ND_ECHO(j, v) from $(n-t)$ different processes, it locally ND-delivers MSG(j, v) (lines 3–4).

The algorithm considers an instance of ND-broadcast, i.e., a correct process invokes at most once ND-broadcast. Adding a sequence number to each message allows any process to ND-broadcast a sequence of messages.

Theorem 2. *Algorithm 2 implements* ND-broadcast *in the system model* $\mathcal{BAMP}_{n,t}[t < n/3]$.

Proof. (The proof is from [39]. It is given for completeness.) To prove the ND-termination property, let us consider a non-faulty process p_i that ND-broadcasts the message MSG(v_i). As p_i is non-faulty, the message ND_INIT(i, v_i) is received by all the non-faulty processes, which are at least $(n-t)$, and every non-faulty process broadcasts ND_ECHO(i, v_i) (line 2). Hence, each non-faulty process receives the message ND_ECHO(i, v_i). from $(n-t)$ different processes. It follows that every non-faulty process eventually ND-delivers the message MSG(i, v_i) (lines 3–4).

To prove the ND-no-duplicity property, let us assume by contradiction that two non-faulty processes p_i and p_j ND-deliver different messages m_1 and m_2 from some process p_k (i.e., $m_1 =$ MSG(k, v) and $m_2 =$ MSG(k, w), with $v \neq w$). It follows from the predicate of line 3, that p_i received ECHO(k, v) from a set of $(n-t)$ distinct processes, and p_j received ECHO(k, w) from a set of $(n-t)$ distinct processes. As $n > 3t$, it follows that the intersection of these two sets contains a non-faulty process. But, as it is non-faulty, this process sent the same message ND_ECHO() to p_i and p_j (line 2). Hence, $m_1 = m_2$, which contradicts the initial assumption.

To prove the ND-validity property, we show that, if Byzantine processes forge and broadcast a message ND_ECHO(i, w) such that p_i is correct and has never invoked ND_broadcast MSG(w), then no correct process can ND-deliver MSG(i, w). Let us observe that at most t processes can broadcast the message ND_ECHO(i, w). As $t < n - t$, it follows that the predicate of line 3 can never be satisfied at a correct process. Hence, if p_i is correct, no correct process can ND-deliver from p_i a message that has not been ND-broadcast by p_i. $\square$

It is easy to see that this implementation uses two consecutive communication steps and $O(n^2)$ underlying messages ($n-1$ in the first communication step, and $n(n-1)$ in the second one). Moreover, there are two types of protocol messages, and the size of the control information added to a message is $\log_2 n$ (sender identity).

4.3. *k*-Set agreement

Definition. The intrusion-tolerant Byzantine (ITB) k-set agreement was informally presented in the introduction. When considering round-based randomized k-set agreement algorithms (namely, the system model $\mathcal{BAMP}_{n,t}$[LRC]) these properties are the following.

- B-KS-Validity. If a correct process decides v, then v was proposed by a correct process.
- B-KS-Agreement. The set of values decided by the correct processes contains at most k values.
- B-KS-P-Termination. $\lim_{r \to +\infty} \big(\text{Probability } [p_i \text{ decides by round } r]\big) = 1$.

Additional constraint. As stated in the Introduction, we assume $k \leq t$. Moreover, we have also seen that, in order for a correct process to decide neither a value proposed only by Byzantine processes, nor a predefined default value, it is assumed that, whatever the domain of the values that can be proposed by the correct processes, in any execution, at most m different values are proposed by correct processes, where m depends on n and t, namely, $n > t(m+1)$. As shown in [18], this condition is necessary.

```
let $witness(v)$ = number of different processes from which MV_VAL($v$) was received.

operation MV_broadcast MSG($v_i$) is
(1)    broadcast MV_VAL($v_i$); return().

when MV_VAL($v$) is received
(2)    if $(witness(v) \geq t+1) \wedge$ (MV_VAL($v$) not yet broadcast)
(3)       then broadcast MV_VAL($v$)       % a process echoes a value only once %
(4)    end if;
(5)    if $(witness(v) \geq n-t) \wedge (v \notin mv\_valid_i)$
(6)       then $mv\_valid_i \leftarrow mv\_valid_i \cup \{v\}$      % local delivery of a value %
(7)    end if.
```

Fig. 3. Implementing MV-broadcast in $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$ (Algorithm 3).

Hence, assuming the non-triviality conditions $k \leq t$, and the fact that, in any execution, at most m different values are proposed by the correct processes, the system model considered here to solve the ITB k-set agreement problem is $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$.

5. Two multivalued validated broadcast abstractions

This section presents the all-to-all communication abstractions MV-broadcast and SMV-broadcast. "All-to-all" mean that it is assumed that all the non-faulty processes invoke the corresponding broadcast operation. As indicated in the introduction, these abstractions extend to the "multivalue" case the BV-broadcast and SBV-broadcast communication abstractions introduced in [23], which consider binary values only.

5.1. Multivalued validated all-to-all broadcast

Definition of MV-broadcast. This communication abstraction provides the processes with a single operation denoted MV_broadcast(). When a process invokes MV_broadcast TAG(m), we say that it "MV-broadcasts the message typed TAG and carrying the value m". The invocation of MV_broadcast TAG(m) does not block the invoking process. The aim of MV-broadcast is to eliminate the values (if any) that have been broadcast only by Byzantine processes.

In each instance of the MV-broadcast abstraction, each correct process p_i MV-broadcasts a value and eventually obtains a set of values. To store these values, MV-broadcast provides each process p_i with a read-only local variable denoted mv_values_i. This set variable, initialized to $\emptyset$, increases asynchronously when new values are received. Each instance of MV-broadcast is defined by the four following properties.

- MV-Termination. The invocation of MV_broadcast() by a correct process terminates.
- MV-Justification. If p_i is a correct process and $v \in mv\_valid_i$, v has been MV-broadcast by a correct process.
- MV-Uniformity. If p_i is a correct process and $v \in mv\_valid_i$, eventually $v \in mv\_valid_j$ at every correct process p_j.
- MV-Obligation. Eventually the set mv_valid_i of each correct process p_i is not empty.

The following properties are immediate consequences of the previous definition.

- MV-Equality. The sets mv_valid_i of the correct processes are eventually non-empty and equal.
- MV-Integrity. The set mv_valid_i of a correct process p_i never contains a value MV-broadcast only by Byzantine processes.

On the feasibility condition $n > (m+1)t$. Let m be the number of different values MV-broadcast by correct processes. It follows from the previous specification that, even when the (at most) t Byzantine processes propose the same value w, which is not proposed by correct processes, w cannot belong to the set mv_valid_i of a correct process p_i. This can be ensured if and only if there is a value MV-broadcast by at least $(t+1)$ correct processes. This feasibility condition is captured by the predicate $n - t > mt$ (see [18] for a proof of this feasibility condition). Hence $n > (m+1)t$ is a feasibility condition for MV-broadcast to cope with up to t Byzantine processes. Let us notice that, as $m \geq 2$, $n > (m+1)t$ implies $n > 3t$.

An MV-broadcast algorithm. Algorithm 3 describes a simple implementation of MV-broadcast, suited to the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$. This algorithm is based on a simple "echo" mechanism. Differently from previous echo-based algorithms (e.g., [7,38]), the echo is used here with respect to each value that has been received (whatever the number of processes that broadcast it), and not with respect to each pair composed of a value plus the identity of the process that broadcast this value. Hence, a value entails at most one echo per process, whatever the number of processes that MV-broadcast this value.

When a process p_i invokes MV_broadcast MSG(v_i), it broadcasts MV_VAL(v_i) to all the processes (line 1). Then, when a process p_i receives (from any process) a message MV_VAL(v), (if not yet done) it forwards this message to all the processes (line 3) if it has received the same message from at least $(t+1)$ different processes (line 2). Moreover, if p_i has received v from at least $(n-t) \geq (2t+1)$ different processes, the value v is added to mv_valid_i (lines 5–6). Let us notice that, except in the case where $|mv\_valid_i| = m$, no correct process p_i can know if its set mv_valid_i has obtained its final value.

Theorem 3. *Algorithm 3 implements MV-broadcast in the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$.*

Proof. The proof of the MV-Termination property is trivial. If a correct process invokes MV_broadcast(), it eventually sends a message to each process, and terminates.

Proof of the MV-Justification property. To show this property, we prove that a value MV-broadcast only by faulty processes cannot be added to the set mv_valid_i of a correct process p_i. Hence, let us assume that only faulty processes MV-broadcast v. It follows that a correct process can receive the message MV_VAL(v) from at most t different processes. Consequently the predicate of line 2 cannot be satisfied at a correct process. Moreover, as $n-t > t$, the predicate of line 5 cannot be satisfied either at a correct process, and the property follows.

Proof of the MV-Uniformity property. If a value v is added to the set mv_valid_i of a correct process p_i (local delivery), this process received MV_VAL(v) from at least $(n-t)$ different processes (line 5), i.e., from at least $(n-2t)$ different correct processes. As each of these correct processes sent this message to all the processes, it follows that the predicate of line 2 is eventually satisfied at each correct process, which consequently broadcasts MV_VAL(v) to all. As there are at least $(n-t)$ correct processes, the predicate of line 5 is then eventually satisfied at each correct process, and the MV-Uniformity property follows.

Proof of the MV-Obligation property. It follows from the feasibility condition $n > (m+1)t$, that there is a value v MV-broadcast by at least $(t+1)$ correct processes. It then follows that these processes issue MV_broadcast MSG(v), and consequently all correct processes first deliver the message MV_VAL(v) and then broadcast at line 3 (if not previously done). Hence, each correct process p_i eventually delivers this message from $(n-t)$ processes and adds v to its set mv_valid_i (line 5–6), which proves the property. $\square$

Cost of the algorithm. As at most m values are MV-broadcast by the correct processes, it follows from the text of the algorithm that each correct process broadcasts each of these values at most once (at line 1 or line 3). Hence, if there are $c \in [n-t..n]$ correct processes, their broadcasts entail the sending of at most $m\ c\ n$ messages MV_VAL(). Finally, whatever the number of values that are MV-broadcast, the algorithm requires at most two communication steps.

5.2. Synchronized multivalued validated all-to-all broadcast

Definition of SMV-broadcast. This all-to-all communication abstraction provides the processes with a single operation denoted SMV_broadcast TAG(). As indicated by its name, its aim is to synchronize processes so that, if a single value v is delivered to a correct process, then v is delivered to all the correct processes.

In each instance of the SMV-broadcast abstraction, each correct process invokes SMV_broadcast TAG(). Such an invocation returns to the invoking process p_i a set denoted $view_i$ and called a local view. We say that a process *contributes* to a set $view_i$ if the value it SMV-broadcasts belongs to $view_i$. SMV-broadcast is defined by the following properties.

- SMV-Termination. The invocation of SMV_broadcast TAG() by a correct process terminates.
- SMV-Obligation. The set $view_i$ returned by a correct process p_i is not empty.
- SMV-Justification. If p_i is correct and $v \in view_i$, then a correct process SMV-broadcast v.
- SMV-Inclusion. If p_i and p_j are correct processes and $view_i = \{v\}$, then $v \in view_j$.
- SMV-Contribution. If p_i is correct, at least $(n-t)$ processes contribute to its set $view_i$.
- SMV-No-duplicity. Let *VIEW* be the union of the sets $view_i$ of the correct processes. A process contributes to at most one value of *VIEW*.

The following property is an immediate consequence of the previous definition.

- SMV-Singleton. If p_i and p_j are correct, $[(view_i = \{v\}) \wedge (view_j = \{w\})] \Rightarrow (v = w)$.

Let $v \in VIEW$, p_i a correct process, and p_j a Byzantine process. It is possible that, while the value v was SMV-broadcast by p_i (hence p_i contributed to *VIEW*), p_j also appears as contributing to *VIEW* with the same value v. The SMV-No-duplicity property states that if a value v proposed by a Byzantine process appears in *VIEW* then v was also proposed by a correct process.

Communication abstractions similar to – but different from – SMV-broadcast are described in [4,12] (GradeCast), and [1] (Binding Gather).

```
operation SMV_broadcast MSG ($v_i$) is
(1)   MV_broadcast MSG($est_i$);
(2)   wait ($mv\_values_i \neq \emptyset$);
      % $mv\_values_i$ has not necessarily its final value when the wait statement terminates %
(3)   ND_broadcast ND_AUX($w$) where $w \in mv\_values_i$;
(4)   wait ($\exists$ a set $view_i$ such that (i) $view_i \subseteq mv\_values_i$, and
              (ii) $view_i$ is built with the values in the messages ND_AUX() rec. from $(n-t)$ distinct processes);
(5)   return ($view_i$).
```

Fig. 4. Implementing SMV-broadcast in $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$ (Algorithm 4).

An SMV-broadcast algorithm. Algorithm 4 implements the SMV-broadcast abstraction in the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$. A process p_i first MV-broadcasts a message MSG (v_i) and waits until the associated set mv_values_i is not empty (lines 1–2). Let us remind that, when p_i stops waiting, the set mv_values_i has not necessarily obtained its final value. Then, p_i extracts a value w from mv_values_i and ND-broadcasts it to all (line 3). Let us notice that, due to the ND-no-duplicity property, no two correct processes can ND-deliver different values from the same Byzantine process.

Finally, p_i waits until the predicate of line 4 is satisfied. This predicate, which defines the set $view_i$ returned by the invocation of SMV_broadcast MSG (v_i) issued by p_i, has two aims.

- The first is to discard from $view_i$ (the set returned by p_i) a value broadcast only by Byzantine processes. As mv_values_i contains only values broadcast by correct processes, this property is obtained from the predicate $view_i \subseteq mv\_values_i$.
- The second aim is to ensure that, if the view $view_i$ of a correct process p_i contains a single value, then this value eventually belongs to the view $view_j$ of any correct process p_j.
  To attain this goal, it is required that each value of $view_i$ is a value received in one (or more) message ND_AUX(), and messages ND_AUX() from at least $(n-t)$ different processes (hence, from at least $(n-2t)$ correct processes, i.e., a majority of processes) contribute to $view_i$.[2]

Multiset version of SMV-broadcast. While a value belongs or does not belong to a set, a multiset (also called a bag) is a set in which the same value can appear several times. As an example, while $\{a, b, c\}$ and $\{a, b, b, c, c, c\}$ are the same set, they are different multisets.

It is easy to see that the "set" version of the SMV-broadcast (where $view_i$ is a set) and Algorithm 4 can be easily converted into a "multiset" version where $view_i$ is a multiset. Both versions will be used in the randomized k-set agreement presented in Section 6.

Theorem 4. *Algorithm 4 implements* SMV-broadcast *in the system model* $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$.

Proof. Proof of the SMV-Termination property. Let us first observe that, due to the MV-Termination property and the MV-Obligation property of the underlying MV-broadcast, no correct process blocks forever at line 2. As there are at least $(n-t)$ correct processes, and none of them blocks forever a line 2, it follows from the ND-Termination property that each correct process returns from the ND-broadcast at line 3, and eventually ND-delivers values from at least the $(n-t)$ correct processes. Moreover, due to the MV-Justification property, these values have been SMV-broadcast by correct processes, and, due to the MV-Uniformity property, the sets mv_valid_i of all correct processes are eventually equal. It then follows that the predicate of line 4 becomes eventually satisfied at any correct process p_i, and consequently the invocations of SMV_broadcast() of the correct processes terminate.

Proof of the SMV-Obligation property. Any correct process p_i eventually ND-delivers $(n-t)$ messages ND_AUX() sent by correct processes. As (a) these messages carry values taken from the set mv_values_x of correct processes, and (b) these sets (b.1) are eventually equal at all correct processes, and (b.2) contain all values ND-broadcast at line 3 by the correct processes, it follows (from the predicate of line 4) that the set $view_i$ returned by a correct process is not empty.

Proof of the SMV-Justification property. This property follows directly from the fact that the predicate of line 4 discards the values ND-broadcast only by Byzantine processes, and from the MV-Justification property, namely, the set mv_values_i of a correct process contains only values MV-broadcast by correct processes.

Proof of the SMV-Inclusion property. Let us consider a correct process p_i and assume $view_i = \{v\}$. It follows from the predicate of line 4 that p_i has ND-delivered the same message ND_AUX(v) from at least $(n-t)$ different processes. As at most t of them are Byzantine, it follows that p_i ND-delivered this message from at least $(n-2t)$ different correct processes, i.e., as $n-2t \geq t+1$, from at least $(t+1)$ correct processes.

Let us consider any correct process p_j. This process ND-delivered messages ND_AUX() from at least $(n-t)$ different processes. As $(n-t)+(t+1) > n$, it follows that there is a correct process p_x that ND-broadcast the same message ND_AUX(v) to p_i and p_j. It follows that $v \in view_j$, which concludes the proof of the lemma.

[2] As an example, if $view_i$ contains two different values $v1$ and $v2$, p_i received the message ND_AUX($v1$) from n_1 processes, the message ND_AUX($v2$) from n_2 processes, where $n_1 > 0$, $n_2 > 0$, and $n_1 + n_2 \geq n - t$.

```
operation propose_k(v_i) is
(1)  $est_i \leftarrow v_i$; $r_i \leftarrow 0$;
(2)  repeat forever
(3)    $r_i \leftarrow r_i + 1$;
// ------------- phase 1 ---------------------
(4)    $view_i[r_i,1] \leftarrow$ SMV_broadcast PHASE$[r_i,1](est_i)$;     % $view_i[r_i,1]$ is a multiset %
(5)    if ($\exists v$ appearing $W$ times in $view_i[r_i,1]$) then $aux \leftarrow v$ else $aux \leftarrow \perp$ end if;
// ------------- phase 2 ---------------------
(6)    $view_i[r_i,2] \leftarrow$ SMV_broadcast PHASE$[r_i,2](aux)$;     % $view_i[r_i,2]$ is a set %
(7)    case ($\perp \notin view_i[r_i,2]$)          then let $v$ be any value $\in view_i[r_i,2]$;
(8)                                                   broadcast DECIDE$(v)$; return$(v)$
(9)         ($view_i[r_i,2] = \{\perp, v, \cdots\}$)  then $est_i \leftarrow$ any value non-$\perp \in view_i[r_i,2]$
(10)        ($view_i[r_i,2] = \{\perp\}$)             then $est_i \leftarrow$ random$(mv\_valid_i[1,1])$
(11)   end case
(12) end repeat.
```

Fig. 5. SMV-broadcast-based Byzantine k-set agreement in $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$ (Algorithm 5).

Proof of the SMV-Contribution property. This property follows trivially from the part (ii) of the waiting predicate of line 4.

Proof of the SMV-No-duplicity property. This property is an immediate consequence of the ND-No-duplicity property of the ND-broadcast issued at line 3. □

6. Byzantine model: a randomized k-set agreement algorithm

This section presents and proves correct an algorithm which solves the k-set agreement problem in $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$. This algorithm is built in a modular way on top of the SMV-broadcast communication abstraction [24].

6.1. Description of the algorithm

Local variables. To solve the ITB k-set agreement problem, Algorithm 5, which is round-based, relies on a very modular construction. Each process p_i manages two local variables whose scope is the whole execution: a local round number r_i, and a local estimate of a decision value, denoted est_i. It also manages three local variables whose scope is the current round r: a multiset $view_i[r,1]$, an auxiliary variable aux, and a set $view_i[r,2]$.

Description of the algorithm. When p_i invokes propose$_k(v_i)$ it assigns v_i to est_i and initializes r_i to 0 (line 1). Then p_i enters a loop that it will exit at line 8 by executing return(v), which returns the decided value v and stops its participation in the algorithm.

Each round r executed by a process p_i is made up of two phases. During the first phase of round r, each correct process p_i invokes SMV_broadcast(est_i) (multiset version) and stores the multiset returned by this invocation in $view_i[r,1]$. Let us remind that this multiset contains only values SMV-broadcast by at least one correct process. The aim of this phase is to build a global set, that we denote $AUX[r]$ which contains at most $(k+1)$ values, at most k being contributed by correct processes, and the other one being the default value $\perp$.[3] To this end, each correct process p_i checks if there is a value v that appears "enough" (say W) times in the multiset $view_i[r,1]$. If there is such a value v, p_i adopts it (assignment $aux \leftarrow v$), otherwise it adopts the default value $\perp$ (line 5).

To this end, the set $AUX[r]$ is made up of the variables aux of all the correct processes. For $AUX[r]$ to contain at most k non-$\perp$ values, W has to be such that $(k+1)W > n$ (there are not enough processes for $(k+1)$ different values such that each of them was contributed by W processes).[4] Hence, $W > n/(k+1)$.[5]

When it starts the second phase of round r, each correct process p_i invokes SMV_broadcast(aux) (set version) and stores the set it obtains in $view_i[r,2]$. Due to the properties of SMV-broadcast, $view_i[r,2]$ is a local approximation of $AUX[r]$, namely, we have $view_i[r,2] \subseteq AUX[r]$. Then, the behavior of p_i depends on the content of the set $view_i[r,2]$.

- If $\perp \notin view_i[r,2]$, p_i decides any value v in $view_i[r,2]$ (lines 7–8). The choice of the value v is non-deterministic.
- If $view_i[r,2]$ contains $\perp$ and non-$\perp$ values, p_i updates its current estimate est_i to any non-$\perp$ value of $view_i[r,2]$ and starts new round (line 9).

[3] While the value of the set $AUX[r]$ could be known by an external global observer, its value can never be explicitly known by a correct process. However, a process can locally build an approximation of it during the second phase.

[4] Let us remind that, due to the ND-broadcast used in the algorithm implementing SMV-broadcast, two correct processes cannot ND-deliver different values from the same Byzantine process.

[5] See the computation of W in Section 3.1.

- If $view_i[r, 2]$ contains only $\bot$, p_i starts a new round, but updates previously its current estimate est_i to a random value (line 10). This random value is obtained from the set (denoted $mv\_valid_i[1, 1]$ in the algorithm) locally output by the first MV-broadcast instance invoked by p_i. The use of these sets allows the algorithm to benefit from the fact that these sets are eventually equal at all correct processes (MV-Equality property). The B-KS-P-Termination relies on this property.

As shown in the proof, an important behavioral property of the algorithm lies in the fact that, at any round r, it is impossible for two correct processes p_i and p_j to be such that $(\bot \notin view_i[r, 2]) \wedge (view_i[r, 2] = \{\bot\})$. These two predicates are mutually exclusive.

On the value of W. (This discussion is similar to the one on the definition of W and R appearing in Section 3.1.) The value W is used at line 5 for a safety reason, namely, no more than k non-$\bot$ values can belong to the set $AUX[r]$. As we have seen, this is captured by the constraint $W(k + 1) > n$. It appears that W has also to be constrained for a liveness reason, namely, when the correct processes start a new round r with at most k different estimates values, none of them must adopt the value $\bot$ at line 5 (otherwise, instead of deciding at line 7, they could loop forever).

This liveness constraint is as follows. Let us consider the size of the multiset $view_i[r, 1]$ obtained at line 4. In the worst case, when the correct processes start a new round r with at most k different estimates, $view_i[r, 1]$ may contain $(k - 1)$ different values, each appearing $(W - 1)$ times, and only one value that appears W times. Hence, $view_i[r, 1]$ must contain at least $R = (W - 1)(k - 1) + W = (W - 1)k + 1$ elements. As it follows from Algorithm 4 that $|view_i[r, 1] \geq n - t$, we obtain the liveness constraint $n - t \geq (W - 1)k + 1$.

On message identities. The messages PHASE() SVM-broadcast at line 4 and line 6 are identified by a pair $[r, x]$ where r is a round number and $x \in \{1, 2\}$ a phase number. Each of these messages gives rise to underlying messages ND_AUX() (Algorithm 3), MV_VAL() (Algorithm 2), and underlying sets $witness()$ (Algorithm 2). Each of them inherits the pair identifying the message PHASE() it originates from.

On the messages DECIDE(). Before a correct process decides a value v, it sends a message DECIDE(v) to each other process (line 8). Then, it stops its execution. This halting has not to prevent correct processes from terminating, which could occur if they wait forever underlying messages ND_AUX() or MV_VAL() from p_i.

To this end, a message DECIDE(v) has to be considered as representing an infinite set of messages. More precisely if, while executing a round r, a process p_i receives a message DECIDE(v) from a process p_j, it considers that it has received from p_j the following set of messages: $\{$ND_AUX$[r', 1](v)$, ND_AUX$[r', 2](v)$, MV_VAL$[r', 1](v)$, MV_VAL$[r', 2](v)\}_{r' \geq r}$. It is easy to see that the messages DECIDE() simulate a correct message exchange that could be produced, after it has decided, by a deciding but non-terminating process.

Another solution would consist in using a Reliable Broadcast abstraction that copes with Byzantine processes. In this case, a process could decide a value v as soon as it has RB-delivered $(t + 1)$ messages DECIDE(v). An algorithm implementing such a reliable broadcast is presented in [7]. This algorithm requires $O(n^2)$ messages and assumes $n < t/3$, which is a necessary requirement to implement one reliable broadcast in the presence of Byzantine processes.

6.2. Proof of the algorithm

The proof considers the system model $\mathcal{BAMP}_{n,t}[t < n/(m + 1), \mathrm{LRC}]$, the algorithmic safety and liveness constraints on W, namely, $W(k + 1) > n$ and $n - t \geq (W - 1)k + 1$, and the non-triviality condition $(k < m) \wedge (k \leq t)$.

Preliminary remark 1. The proof considers the semantic of the messages DECIDE() described previously. This is equivalent to consider that, after it has decided, a correct process continues executing while skipping line 8.

Notation. Given a round r, let $EST[r]$ be the set of estimate values of the correct processes when they start round r, and $AUX[r]$ be the set including the values of the aux_i variables of the correct processes at the end of the first phase of round r (i.e., just after line 5). Let us notice that $AUX[r]$ can contain $\bot$.

Preliminary remark 2. The proof of the MV-Obligation property requires that at most m different values are MV-broadcast. Hence, this requirement extends to the invocations SMV_broadcastPHASE$[r, x]()$, where $x \in \{1, 2\}$. By assumption, this requirement is initially satisfied, namely, $|EST[1]| \leq m$. We will see in the proof that (i) $AUX[r]$ contains at most k values proposed by correct processes plus possibly $\bot$, (ii) $view_i[r, 2]$ is a subset of $AUX[r]$, and (iii) $mv\_valid_i[1, 1]$ contains only values proposed by correct processes. From the previous observations we conclude that at most m different values are SMV-broadcast at line 4 and line 6 of Algorithm 5.

Lemma 7. *If a correct process decides a value, this value was proposed by a correct process.*

Proof. Let us consider the first round $r = 1$. It follows from the MV-Justification property of the SMV-broadcast invocation at line 4 that the multiset $view_i[1,1]$ of any correct process p_i contains only values SMV-broadcast by correct processes. The same is true for the set $view_i[1,2]$ which, in addition, can also contain the default value $\perp$. It follows that, if a correct process decides at lines 7–8, it decides a value proposed by a correct process. If a correct process progresses to the next round, it executes line 9 or line 10 (for line 10, this follows from the MV-Justification property of the MV-broadcast generated by the invocation SMV_broadcast PHASE$[1,1](est_i)$). In both cases, its new estimate value is a value proposed by a correct process. Hence the estimate values of the processes that start the second round are values proposed by correct processes. Applying this reasoning to the sequence of rounds, it follows that no correct process can decide a value not proposed by a correct process. □

Lemma 8. *$AUX[r]$ contains at most k non-$\perp$ values, plus possibly the default value $\perp$.*

Proof. Let us assume that $AUX[r]$ contains $(k+1)$ non-$\perp$ values. If a value belongs to this set, it is the value of the local variable aux_i of a correct process p_i, which appears at least W times in the multiset $view_i[r,1]$ (line 5). Moreover, due to SMV-No-duplicity property, a process (correct or Byzantine) contributes to at most one of these values. It follows from these observations that, if $AUX[r]$ contains $(k+1)$ non-$\perp$ values, $(k+1)W$ distinct processes have contributed to $AUX[r]$, i.e., have SMV-broadcast PHASE$[r,1]()$ messages. As $(k+1)W > n$, this is impossible. □

Lemma 9. *If $|EST[r]| \leq k$, any correct process that starts round r decides during r a value of $EST[r]$.*

Proof. As by assumption the correct processes have at most k different estimate values at the beginning of round r, it follows from the SMV-Contribution property of the SMV-broadcast of line 4 that at least $(n-t)$ different processes contributed to the multiset $view_i[r,1]$. As $n - t \geq (W-1)k + 1$ (algorithmic liveness), it follows that the multiset $view_i[r,1]$ of any correct process p_i contains at least W copies of a value of $EST[r]$. Hence, $aux_i \in EST[r]$ at each correct process. Consequently $AUX[r] \subseteq EST[r]$. It then follows that the predicate of line 7 is satisfied at any correct process p_i, which decides accordingly a value of $view_i[r,2] \subseteq AUX[r] \subseteq EST[r]$, which concludes the proof of the lemma. □

Lemma 10. *Let p_i and p_j be two correct processes. At any round r, the predicates $\perp \notin view_i[r,2]$ and $view_j[r,2] = \{\perp\}$ are mutually exclusive.*

Proof. Let us assume by contradiction that p_i is a correct process such that the predicate $\perp \notin view_i[r,2]$ is satisfied (line 7), and p_j a correct process such that the predicate $view_j[r,2] = \{\perp\}$ is satisfied (line 10).

Due to the SMV-Contribution property of the SMV-broadcast issued by p_i and p_j at line 6, it follows that $view_i[r,2]$ contains values contributed by at least $(n-t)$ processes, and similarly for the set $view_j[r,2]$ of p_j. As $n > 3t$, the intersection of any two sets of $(n-t)$ processes contains at least $(t+1)$ processes, i.e., one correct process. It then follows that there is a correct process that contributed to both $view_i[r,2]$ and $view_j[r,2]$, from which we conclude that either $view_i[r,2]$ contains $\perp$, or $view_j[r,2]$ contains a non-$\perp$ estimate value. □

Lemma 11. *No more than k different values are decided by the correct processes.*

Proof. Let r be the first round during which correct processes decide. They decide at line 8. Due to Lemma 8, the set $AUX[r]$ contains at most k non-$\perp$ values. Moreover, due to the SMV-broadcast issued by the correct processes at line 6 that we have $view_i[r,2] \subseteq AUX[r]$ at each correct process p_i. Hence, due to line 7, a process that decides during round r can only decide a value of $AUX[r]$.

Let us now consider a correct process p_j that proceeds to round $(r+1)$. Let p_i be a process that decides at round r. It follows from Lemma 10 that the predicates $\perp \notin view_i[r,2]$ and $view_j[r,2] = \{\perp\}$ are mutually exclusive. Consequently, p_j executes line 9 before progressing to the next round. Hence, p_j updated est_j to a non-$\perp$ value of $view_j[r,2] \subseteq AUX[r]$ before progressing to the next round. It follows that the estimates of the correct processes progressing to the next round are non-$\perp$ values of $AUX[r]$. Hence, $EST[r+1] \subseteq AUX[r] \setminus \{\perp\}$. It then follows from Lemma 9 that at most k values are decided. □

Lemma 12. *No correct process blocks forever in a round.*

Proof. The proof is by contradiction. Let r be the first round at which a correct process p_i blocks forever. It can block at line 4 or line 6. Let us first consider line 4. As no correct process blocked forever at a round $r' < r$, all correct processes start round r and invoke SMV_broadcast PHASE$[r,1](-)$. It then follows from the SMV-termination property that p_i returns from its invocation. The same reasoning applies to line 6, which concludes the proof of the lemma. □

Lemma 13. *If a correct process decides during a round r, any other correct process that does not decide by round r, decides during the round $(r+1)$.*

Proof. The proof is by contradiction. Let us suppose that a correct process p_i decides v at round r (line 8) and a correct process p_j, which does not decide by round r. Due to Lemma 12, p_j proceeds to round $(r+1)$. Due to Lemma 10 and the fact that p_i decides at round r, it follows that $view_j[r,2] \neq \{\bot\}$. Hence, p_j executes line 9, and assigns a non-$\bot$ of $AUX[r]$ to est_j. As $AUX[r]$ contains at most k non-$\bot$ values (Lemma 8), we have $EST[r+1] \subseteq AUX[r]$, i.e., the round $(r+1)$ starts with at most k non-$\bot$ values. Due to the Lemma 9, p_j decides in the round $r+1$. A contradiction. □

Lemma 14. *Let $VALID[1,1]$ be the final (common) value of the sets $mv\_valid_i[1,1]$ of the correct processes. $\forall r$ we have $AUX[r] \subseteq VALID[1,1]$.*

Proof. The proof follows from the observation that the values, proposed by a correct process, which are not in $VALID[1,1]$ can appear neither in $view_i[r,1]$ nor in $view_i[r,2]$. Hence, they cannot appear either in a set $AUX[r]$, and $AUX[r] \subseteq VALID[1,1]$ follows. □

Lemma 15. *All correct processes decide with probability* 1.

Proof. Due to Lemma 13, if a correct process decides, all correct processes decide. Hence, let us assume by contradiction that no correct process decides.

Due to the MV-Equality property of the MV-broadcast generated by the invocations of SMV_broadcast PHASE$[1,1]()$ issued by the correct processes, there is a finite time τ after which the sets $mv\_valid_i[1,1]$ of the correct processes remain forever non-empty and equal.

As no correct process blocks forever in a round (Lemma 12), all correct processes progress from round to round forever. Moreover, as the decision predicate of line 7 is never satisfied at a correct process, it follows that, after τ, any correct process executes line 9 or line 10. Let us consider a round r entered by all correct processes after time τ. There are three cases.

- Case 1: At round r, all the correct processes execute line 9. So, each correct process sets its estimate to a non-$\bot$ value of $AUX[r]$. Due to Lemma 8, there are then at most k different estimate (non-$\bot$) values in $AUX[r]$. Hence, all the correct processes start the round $(r+1)$, and $EST[r+1]$ contains at most k different estimate values (none being $\bot$). It then follows from Lemma 9 that all correct processes decide.
- Case 2: During r at least one process (but not all) executes line 9. In this case, due to Lemma 8, each correct process p_i that executes line 9 sets its current estimate est_i to a non-$\bot$ value taken from the set $AUX[r]$, which contains at most k non-$\bot$ values. The other processes execute line 10. This means that each of these processes p_i sets its estimate value est_i to a value $\in mv\_valid_i[r,1] = VALID[1,1]$. As $AUX[r] \subseteq VALID[1,1]$ (Lemma 14), there is a probability $prob_1 > 0$ that they obtain values from $AUX[r]$.
- Case 3: During r no process executes line 9. In this case, all the processes execute line 10. There is a probability $prob_2 > 0$ that they obtain at most k different estimate values.

In Case 1, all correct processes decide. Let us consider Case 2 and Case 3. During any round after τ, there is a probability $p = \min(prob_1, prob_2)$ that the correct processes have at most k different estimate values. Hence, there is a probability $P(\alpha) = p + p(1-p) + p(1-p)^2 + \ldots + p(1-p)^{\alpha-1} = 1-(1-p)^{\alpha}$ that, after at most α rounds, the processes have no more than k estimate values. As $\lim_{\alpha\to\infty} P(\alpha) = 1$, it follows that, with probability 1, all correct processes will start a round with no more than k estimate values. Then, according to Lemma 9, they will decide. □

Theorem 5. *Algorithm 5 solves the randomized Byzantine k-set agreement problem in the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$.*

Proof. B-KS-Validity follows from Lemma 7. B-KS-Agreement follows from Lemma 11. B-KS-P-Termination follows from Lemma 15. □

7. Conclusion

This paper was on k-set agreement in two types of asynchronous message-passing, the ones where processes may commit crash failures, and the ones where they may commit Byzantine failures. As k-set agreement cannot be solved in these basic system models without additional computational power, the paper considered the computational power provided by local multi-sided random coins.

The first randomized algorithm that has been presented solves k-set agreement in the presence of up to $t < n/2$ crash failures. It also assumes $t < n - k\lfloor \frac{n}{k+1} \rfloor$. The second one solves k-set agreement in the presence of up to $t < n/(m+1)$ Byzantine processes, where m is an upper bound on the number of values that can be proposed by the correct processes. The design of both algorithms is modular. The modular construction of the Byzantine-tolerant algorithm rests on (i) a broadcast abstraction which guarantees that two non-faulty processes cannot receive distinct messages from the same (possibly Byzantine) sender, and (ii) the stacking of two all-to-all communication abstractions which generalize the "binary" communication abstractions introduced in [23] to the multivalue domain. Two interesting features of this algorithm lie in (a) the validity condition it ensures, namely, no value proposed only by Byzantine processes can be decided by non-faulty processes, and (b) its signature-freedom, which does not limit the computational power of the Byzantine adversary.

Acknowledgments

We want to thank the reviewers for their constructive comments, which help us improve the presentation. Both authors were partially supported by the Franco–German DFG-ANR Project 40300781 DISCMAT ANR 14-CEA35-0010-001 (2017–2020) devoted to connections between mathematics and distributed computing, and the French ANR project DESCARTES ANR 16-CEA40-0023-03 (2017–2020) devoted to distributed software engineering.

References

[1] I. Abraham, M. Aguilera, D. Malkhi, Fast asynchronous consensus with optimal resilience, in: Proc. 24th Int'l Symposium on Distributed Computing (DISC'10), in: Lecture Notes in Comput. Sci., vol. 6343, Springer, 2010, pp. 4–19.
[2] H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edition, Wiley–Interscience, 2004, 414 pp.
[3] M. Ben-Or, Another advantage of free choice: completely asynchronous agreement protocols, in: Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC'83), ACM Press, 1983, pp. 27–30.
[4] M. Ben-Or, D. Dolev, E.N. Hoch, Brief announcement: simple gradecast based algorithms, in: Proc. 24th Int'l Symposium on Distributed Computing (DISC'10), in: Lecture Notes in Comput. Sci., vol. 6343, Springer, 2010, pp. 194–197.
[5] E. Borowsky, E. Gafni, Generalized FLP impossibility results for t-resilient asynchronous computations, in: Proc. 25th ACM Symposium on Theory of Computing (STOC'93), ACM Press, 1993, pp. 91–100.
[6] Z. Bouzid, A. Mostéfaoui, M. Raynal, Minimal synchrony for Byzantine consensus, in: Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC'15), ACM Press, 2015, pp. 461–470.
[7] G. Bracha, Asynchronous Byzantine agreement protocols, Inform. and Comput. 75 (2) (1987) 130–143.
[8] Hillel K. Censor, Multi-sided shared coins and randomized set agreement, in: Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'10), ACM Press, 2010, pp. 60–68.
[9] T. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, J. ACM 43 (2) (1996) 225–267.
[10] S. Chaudhuri, More choices allow more faults: set consensus problems in totally asynchronous systems, Inform. and Comput. 105 (1) (1993) 132–158.
[11] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323.
[12] P. Feldman, S. Micali, Optimal algorithms for Byzantine agreement, in: Proc. 20th ACM Symposium on Theory of Computing (STOC'88), ACM Press, 1988, pp. 148–161.
[13] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (2) (1985) 374–382.
[14] R. Friedman, A. Mostéfaoui, S. Rajsbaum, M. Raynal, Distributed agreement problems and their connection with error-correcting codes, IEEE Trans. Comput. 56 (7) (2007) 865–875.
[15] R. Friedman, A. Mostéfaoui, M. Raynal, Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems, IEEE Trans. Dependable Secure Comput. 2 (1) (2005) 46–56.
[16] E. Gafni, R. Guerraoui, Generalizing universality, in: Proc. 22nd Int'l Conference on Concurrency Theory (CONCUR'11), in: Lecture Notes in Comput. Sci., vol. 6901, Springer, 2011, pp. 17–27.
[17] V. Hadzilacos, S. Toueg, Reliable broadcast and related problems, in: Distributed Systems, ACM Press, 1993, pp. 97–145.
[18] M.P. Herlihy, D. Kozlov, S. Rajsbaum, Distributed Computing Through Combinatorial Topology, Morgan Kaufmann/Elsevier, ISBN 9780124045781, 2014, 336 pp.
[19] M. Herlihy, N. Shavit, The topological structure of asynchronous computability, J. ACM 46 (6) (1999) 858–923.
[20] V. King, J. Saia, Byzantine agreement in expected polynomial time, J. ACM 63 (2) (2016) 14.
[21] L. Lamport, R. Shostack, M. Pease, The Byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401.
[22] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Pub., San Francisco, CA, 1996, ISBN 1-55860-384-4, 872 pp.
[23] A. Mostéfaoui, H. Moumen, M. Raynal, Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time, J. ACM 62 (4) (2015) 31.
[24] A. Mostéfaoui, H. Moumen, M. Raynal, Modular randomized Byzantine k-set agreement in asynchronous message-passing systems, in: Proc. 17th Int'l Conference on Distributed Computing and Networking (ICDCN'16), ACM Press, 2016, 10 pp.
[25] A. Mostéfaoui, S. Rajsbaum, M. Raynal, Conditions on input vectors for consensus solvability in asynchronous distributed systems, J. ACM 50 (6) (2003) 922–954.
[26] A. Mostéfaoui, M. Raynal, Solving consensus using Chandra–Toueg's unreliable failure detectors: a generic quorum-based approach, in: Proc. 13th Int'l. Symposium on Distributed Computing (DISC'99), in: Lecture Notes in Comput. Sci., vol. 1693, Springer, 1999, pp. 49–63.
[27] A. Mostéfaoui, M. Raynal, Randomized k-set agreement, in: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), ACM Press, 2001, pp. 291–297.
[28] A. Mostéfaoui, M. Raynal, Intrusion-tolerant broadcast and agreement abstractions in the presence of Byzantine processes, IEEE Trans. Parallel Distrib. Syst. 27 (4) (2016) 1085–1098.
[29] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (1980) 228–234.
[30] R. de Prisco, D. Malkhi, M.K. Reiter, On k-set consensus problems in asynchronous systems, IEEE Trans. Parallel Distrib. Syst. 12 (1) (2001) 7–21.
[31] M. Rabin, Randomized Byzantine generals, in: Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83), IEEE Computer Society Press, 1983, pp. 116–124.
[32] M. Raynal, Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems, Morgan & Claypool, ISBN 978-1-60845-293-4, 2010, 251 pp.
[33] M. Raynal, Fault-Tolerant Agreement in Synchronous Message-Passing Systems, Morgan & Claypool, ISBN 978-1-60845-525-6, 2010, 165 pp.
[34] M. Raynal, Concurrent Programming: Algorithms, Principles and Foundations, Springer, ISBN 978-3-642-32026-2, 2013, 515 pp.
[35] M. Raynal, J. Stainer, G. Taubenfeld, Distributed universality, Algorithmica 76 (2) (2016) 502–535.
[36] L. Rodrigues, P. Verìssimo, Topology-aware algorithms for large scale communication, in: Advances in Distributed Systems, in: Lecture Notes in Comput. Sci., vol. 1752, Springer, 2000, pp. 1217–1256.

[37] M. Saks, F. Zaharoglou, Wait-free *k*-set agreement is impossible: the topology of public knowledge, SIAM J. Comput. 29 (5) (2000) 1449–1483.
[38] T.K. Srikanth, S. Toueg, Simulating authenticated broadcasts to derive simple fault-tolerant algorithms, Distrib. Comput. 2 (1987) 80–94.
[39] S. Toueg, Randomized Byzantine agreement, in: Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84), ACM Press, 1984, pp. 163–178.