Modular randomized byzantine k-set agreement in asynchronous message-passing systems

Achour Mostefaoui, Hamouma Moumen, Michel Raynal

Modular Randomized Byzantine k-Set Agreement in Asynchronous Message-passing Systems

Achour Mostéfaoui[†] Hamouma Moumen[‡] Michel Raynal[⋆,◦]

[†] LINA, Université de Nantes, 44322 Nantes Cedex, France
[‡] University of Bejaia, Algeria
[⋆] Institut Universitaire de France
[◦] IRISA, Université de Rennes 35042 Rennes Cedex, France
Achour.Mostefaoui@univ-nantes.fr hamouma.moumen@gmail.com raynal@irisa.fr

June 29, 2015

Abstract

k-Set agreement is a central problem of fault-tolerant distibuted 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 dfferent values are decided, and the decided values satisfy some context-depending validity condition. Synchronous message-passing algorithms solving k-set agreement 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 a distributed algorithm that solves k-set agreement in an asynchronous message-passing system wher up to t processes may commit Byzantine failures. To that end, each process is enriched with randomization power. While randomized k-set agreement algorithms exist for the asynchronous process crash failure model where $t \geq k$, to our knowledge the proposed algorithm is the first that solves k-set agreement in the presence of up to $t \geq k$ Byzantine processes. Interestingly, this algorithm is signature-free, and ensures that no value proposed only by Byzantine processes can be decided by a non-faulty process. Its design is based on a modular construction which rests on a "no-duplicity" one-to-all broadcast abstraction, and two all-to-all communication abstractions.

Keywords: Asynchronous message-passing system, Broadcast abstraction, Byzantine process, Coin, Distributed algorithm, k-Set agreement, Randomized algorithm, Signature-free algorithm.

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 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 [7, 17].

The k-set agreement problem is a natural weakening of consensus [6]. It allows the non-faulty processes to decide different values, as long as no more than k values are decided. 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 [9, 22].

Byzantine failures This failure type has first been introduced in the context of synchronous distributed systems [12, 17, 20], and then investigated in the context of asynchronous distributed systems [1, 13, 21]. 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 process crashes (unexpected halting) define a strict subset of Byzantine failures.

Let us remind that (as already said) the world is distributed, asynchronous message-passing systems are more and more pervasive, processes have to agree in one way or another, and 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 consider a system made up of n processes, where up to t may be faulty. Whatever the value of k (i.e., even if $k \leq t$), k-set agreement can always be solved if the system is synchronous [20]. The situation is different in asynchronous systems where k-set agreement is impossible to solve in the process crash failure model when $k \leq t$ [2, 11, 23]. As Byzantine failures are more severe than process crash failures, this impossibility remains true in asynchronous Byzantine systems.

It follows from this impossibility that, when $k \leq t$, the underlying asynchronous distributed system has to be enriched with additional computational power for k-set agreement to be solved. Such an additional computational power can be provided with minimal synchrony assumptions (e.g., [3] which considers $k = 1$ and Byzantine failures), appropriate failure detectors (e.g., [8] which considers $k = 1$ and Byzantine failures), or randomization (e.g., [19] which considers $k = 1$ and Byzantine failures, and [5, 15] which consider $k \leq t$ and crash failures, in read/write shared memory systems and message-passing systems, respectively).

Intrusion-tolerant agreement with respect to Byzantine processes The validity property associated with a distributed agreement problem relates its outputs to its inputs. In a system where processes may commit Byzantine failures, there is no way to direct a Byzantine process to decide some specific value, and consequently the k-set agreement validity property can only be on the values decided by the correct processes.

A classical validity property for Byzantine agreement states that, when 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.) It follows that, as noticed and deeply investigated in [18], the solvability of Byzantine k-set agreement is sensitive to the particular validity property that is considered.

In this paper we consider the following validity property (introduced in [16] 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 very 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 be this number. It is shown in [10] that, in a system of n processes 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 Byzantine k-set agreement. It 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 [14], where the set of values that the processes exchange is limited to two values.
- The second contribution is a modular k-set agreement algorithm for asynchronous message-passing systems where processes may commit Byzantine failures. This algorithm, which round-based, relies on the previous SMV-broadcast abstraction, and (as already announced) on the additional computational power supplied by local random coins (with several sides). As far as we know, this is the first randomized k-set agreement algorithm for asynchronous Byzantine message-passing systems.

The previous Byzantine-tolerant algorithms have two noteworthy features. The first is that they all are signature-free. This means that the "adversary" is not required to be computationally bounded. The second is their conceptual simplicity, which is a first-class property.

Roadmap The paper is composed of 5 sections. Section 2 presents the computation model and a basic broadcast abstraction (called ND-broadcast, where ND stand for "No Duplicity") introduced in [25]. As indicated by its name, this broadcast operation, which requires $t < n/3$, allow to hide a duplicity behavior which can be produced by Byzantine processes. Then Section 3 presents the MV-broadcast and SMV-broadcast abstractions. SMV-broadcast is based on both MV-broadcast and ND-broadcast. Section 4 presents the modular randomized k-set agreement algorithm, whose construction relies on two instances of SMV-broadcast per round. Finally, Section 5 concludes the paper.

2 Computation Model and ND-broadcast

2.1 Computation model

Asynchronous processes The system is made up of a finite set Π of $n > 1$ asynchronous sequential processes, namely $\Pi = \{p_1, \ldots, p_n\}$. "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 "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** $j \in \{1, \ldots, n\}$ send TAG(m) to p_j **end for**".This operation is usually called *unreliable* broadcast (if the sender crashes in the middle of the **for** loop, it is possible that only an arbitrary subset correct processes receives a message).

Failure model Up to t processes may exhibit a *Byzantine* behavior. 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"). In the following, this model is both restricted with a constraint on t and enriched with an object providing processes with additional computational power. More precisely, $\mathcal{BAMP}_{n,t}[t < n/\alpha]$ (where α is a positive integer) denotes the model $\mathcal{BAMP}_{n,t}[\emptyset]$ where the maximal number of faulty processes is smaller than n/α, and $\mathcal{BAMP}_{n,t}[t < n/\alpha, \text{LRC}]$ denotes the model $\mathcal{BAMP}_{n,t}[t < n/\alpha]$ where each process is enriched with a local random coin (LRC). Let us notice that, as LRC belongs to the model, it is given for free in $\mathcal{BAMP}_{n,t}[t < n/\alpha, \text{LRC}]$.

Time complexity When computing the time complexity we ignore local computation time, and consider the longest sequence of causally relate messages m_1, m_2, ..., 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 No-duplicity broadcast

Definition of ND-broadcast The ND-broadcast communication abstraction was introduced by S. Toueg in [25]. It is defined by two operations denoted ND_broadcast() and ND_deliver(), which allow the processes to eliminate bad behaviors 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.

When a process invokes ND_broadcast() we say that it "ND-broadcasts" a message, and when it invokes ND_deliver() we say that it "ND-delivers" a message[1]. Considering an instance of ND-broadcast where the operation ND_broadcast() 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.

An algorithm implementing ND-broadcast It is shown in [25] that $t < n/3$ is a necessary requirement to implement ND-broadcast in a Byzantine asynchronous message-passing system. Algorithm 1 (from [25]) implements ND-broadcast in $\mathcal{BAMP}_{n,t_{n,t}}[t < n/3]$ as follows.

```
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 UB_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.
```

Algorithm 1: Implementing ND-broadcast in $\mathcal{BAMP}_{n,t}[t < n/3]$

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 each process to ND-broadcast a sequence of messages.

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

[1] A similar vocabulary will be used for the abstractions MV-broadcast and SMV-broadcast introduced in Section 3.

Proof 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 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 was not been ND-broadcast by p_i. $\square_{Theorem\ 1}$

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).

3 Multivalued Validated Broadcast: MV-broadcast and SMV-broadcast

This section presents the all-to-all MV-broadcast and SMV-broadcast communication abstractions. "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 [14], which consider binary values only.

3.1 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 an MV-broadcast instance, 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.

Definition 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.

Feasibility condition in the presence of up to t Byzantine processes 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 $n - t > mt$ (a proof of this feasibility condition can be found in [10]). 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 2 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., [4, 24]), 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.

```
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) ≥ t + 1) ∧ (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) ≥ n − t) ∧ (v ∉ mv_valid_i)
(6)     then mv_valid_i ← mv_valid_i ∪ {v}    % local delivery of a value %
(7)   end if.
```

Algorithm 2: Implementing MV-broadcast in $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$

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 $(2t+1)$ 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 2. *Algorithm* 2 *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 he 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 and adds v to its set mv_valid_i (line 5-6), which proves the property. $\square_{Theorem\ 2}$

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.

3.2 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(). 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 an SMV-broadcast instance, each correct process invokes SMV_broadcast TAG(m), where TAG is the type of the message and m value it wants to broadcast. 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. property.

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

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 the following: no value $w \in VIEW \setminus \{v\}$ appears as a contribution of p_j.

An SMV-broadcast algorithm Algorithm 3 implements SMV-broadcast in $\mathcal{BAMP}_{n,t}[t < n/(m+1)]$. 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, 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.

```
operation SMV_broadcast MSG (v_i) is
(1)  MV_broadcast MSG(est_i);
(2)  wait (mv_values_i ≠ ∅);
     % mv_values_i has not necessarily its final value when the wait statement terminates %
(3)  ND_broadcast ND_AUX(w) where w ∈ mv_values_i;
(4)  wait (∃ a set view_i such that its values (i) belong to mv_values_i, and
           (ii) come from messages ND_AUX() received from (n − t) distinct processes);
(5)  return (view_i).
```

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

Finally, p_i waits until the predicate of line 4 is satisfied. This predicate has two aims. The first is to discard from $view_i$ (the set returned by p_i) a value broadcast only by Byzantine processes. Hence 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 this end, $(n-t)$ different processes (hence, at least $(n-2t)$ correct processes) must contribute to $view_i$.

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 3 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 4.

Theorem 3. *Algorithm* 3 *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 return 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.

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. $\square_{Theorem\ 3}$

4 Randomized Byzantine k-Set Agreement

This section presents and proves correct a Byzantine k-set agreement algorithm, which is modularly built on top of the SMV-broadcast communication abstraction and the additional computational power supplied to each process by a local random coin (LRC).

4.1 Intrusion-tolerant Byzantine k-set agreement

The intrusion-tolerant Byzantine (ITB) k-set agreement was informally presented in the introduction. It is assumed that each non-faulty process invokes an operation called $\mathsf{propose}_k()$. This operation has an input parameter v, which is the value proposed by the invoking process. It returns a value, which is called the "value decided" by the invoking process. ITB k-set agreement is formally defined in terms of properties that any solution must to satisfy. When considering deterministic k-set agreement algorithms, these properties are the following ones.

- KS-D-Termination. If a correct process invokes $\mathsf{propose}_k()$, it decides a value.
- KS-Validity. If a correct process decides v, then v was proposed by a correct process.
- KS-Agreement. The set of values decided by the correct processes contains at most k values.

As we are interested in a randomized algorithm to solve k-set agreement, the termination property has to be weakened as follows: any correct process decides with probability 1. In the context of round-based randomized algorithms, this property can restated as follows.

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

4.2 Enriching the basic Byzantine asynchronous model with a random coin

As announced, the additional computational power used to solve ITB k-set agreement despite Byzantine processes is supplied by a multi-sided random coin denoted LRC. The random abstraction LRC provides each process with a local coin that provides it with a single operation denoted $\mathsf{random}()$. Each invocation takes a finite set X as input parameter, and returns a value of X such that each value of X as the probability $1/|X|$ to be returned.

As seen in the introduction, we assume $k \leq t$. Moreover, we have also seen that, in order a correct process decides 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, $n > m(t+1)$. Finally, to rule out the trivial algorithm in which a correct process decides the value it proposes, we assume $k < m$.

Hence, assuming the non-triviality conditions $k < m$ and $k \leq t$, 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 $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$.

4.3 A randomized Byzantine k-set agreement algorithm

Local variables To solve the ITB k-set agreement problem, Algorithm 4, 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 $\mathsf{propose}_k(v_i)$ 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 $\mathsf{return}(v)$, which returns the decided value v and stops its participation in the algorithm.

```
operation propose_k(v_i) is
(1)  est_i ← v_i; r_i ← 0;
(2)  repeat forever
(3)    r_i ← r_i + 1;
// ——————————————— phase 1 ———————————————
(4)    view_i[r_i,1] ← SMV_broadcast PHASE[r_i,1](est_i);   % view_i[r_i,1] is a multiset %
(5)    if (∃v appearing W times in view_i[r_i,1]) then aux ← v else aux ← ⊥ end if;
// ——————————————— phase 2 ———————————————
(6)    view_i[r_i,2] ← SMV_broadcast PHASE[r_i,2](aux);   % view_i[r_i,2] is a set %
(7)    case (⊥ ∉ view_i[r_i,2])          then let v be any value ∈ view_i[r_i,2];
(8)                                           broadcast DECIDE(v); return(v)
(9)         (view_i[r_i,2] = {⊥,v,···}) then est_i ← any value non-⊥ ∈ view_i[r_i,2]
(10)        (view_i[r_i,2] = {⊥})       then est_i ← random(mv_valid_i[1,1])
(11)   end case
(12) end repeat.
```

Algorithm 4: Byzantine k-set agreement based on SMV-broadcast, and local random coins

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 $\mathsf{SMV\_broadcast}(est_i)$ (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 [2], denoted $AUX[r]$, which contains at most $(k+1)$ values, such that at most k of them are contributed by correct processes, and the other one is the default value $\perp$. 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).

The set $AUX[r]$ is made up of the aux variables 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)[3]. Hence, $W > n/(k+1)$.

When it starts the second phase of round r, each correct process p_i invokes $\mathsf{SMV\_broadcast}(aux)$ (set version) and stores the set it obtains in $view_i[r,2]$. Due to the properties of SMV-broadcast,

[2] While the value of this set 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, see below.

[3] 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.

$view_i[r,2]$ is a local approximation of $AUX[r]$, namely, $view_i[r,2] \subseteq AUX[r]$. 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 in $view_i[r,2]$ (lines 7-8).
- 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).
- If $view_i[r,2]$ contains only $\perp$, 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 KS-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 $(\perp \notin view_i[r,2]) \wedge (view_i[r,2] = \{\perp\})$. These two predicates are mutually exclusive.

On the value of W The value W is used at line 5 for a safety reason, namely, no more than k non-$\perp$ 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 $\perp$ 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 $(W-1)(k-1)+W = (W-1)k+1$ elements. As it follows from Algorithm 3 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 2), MV_VAL() (Algorithm 1), and underlying sets $witness()$ (Algorithm 1). 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 [4]. This algorithm requires $O(n^3)$ messages and assumes $n < t/3$, which is a necessary requirement to implement reliable broadcast in the presence of Byzantine processes.

4.4 Proof of the algorithm

The proof considers the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$, 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 $\perp$.

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 $\perp$, (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 4.

Lemma 1. *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 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. $\square_{Lemma\ 1}$

Lemma 2. *$AUX[r]$ 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) contribute 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. $\square_{Lemma\ 2}$

Lemma 3. *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 $n - t \geq (W-1)k + 1$ (algoritmic 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. $\square_{Lemma\ 3}$

Lemma 4. *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).

It follows from the SMV-Contribution property of the SMV-broadcast issued by$_i$ and $_j$ at line 6 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. $\square_{Lemma\ 4}$

Lemma 5. *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 2, 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 4 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 3 that at most k values are decided. $\square_{Lemma\ 5}$

Lemma 6. *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 MV-termination property that p_i returns from its invocation. The same reasoning applies to line 6, which concludes the proof of the lemma. $\square_{Lemma\ 6}$

Lemma 7. *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 6, p_j proceeds to round $(r+1)$. Due to to Lemma 4 and the fact that p_i decides at round r, it follows that $view_j[r,2] \neq \{\perp\}$. Hence, p_j executes line 9, an assigns a non-$\perp$ of $AUX[r]$ to est_j. As $AUX[r]$ contains at most k non-$\perp$ values (Lemma 2), we have $EST[r+1] \subseteq AUX[r]$, i.e., the round $(r+1)$ starts with at most k non-$\perp$ values. Due to to the Lemma 3, p_j decides in the round $r+1$. A contradiction. $\square_{Lemma\ 7}$

Lemma 8. *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 we have $AUX[r] \subseteq VALID[1,1]$. $\square_{Lemma\ 8}$

Lemma 9. *All correct processes decide with probability* 1.

Proof Due Lemma 7 if a correct process decides, all correct processes decides. 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 6), 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-$\perp$ value of $AUX[r]$. Due to Lemma 2, there are then at most k different estimate (non-$\perp$) 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 $\perp$). It then follows from Lemma 3 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 2, each correct process p_i that executes line 9 sets its current estimate est_i to a non-$\perp$ value taken from the set $AUX[r]$, which contains at most k non-$\perp$ values. The other processes execute line 10. This means that each of these processes $_i$ 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 8), there is a probability $p_1 > 0$ 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 $p_2 > 0$ 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 $p = \min(p_1, p_2)$ that the correct processes have at most k different estimate values. Hence, there is a probability $P(\alpha) = p + p(1-p) + p(1-p)^2 + ... + p(1-p)^{\alpha-1} = 1 - (1-p)^\alpha$ 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 processes will start a round with no more than k estimate values. Then, according to Lemma 3, they will decide. $\square_{Lemma\ 9}$

Theorem 4. *Algorithm* 4 solves the randomized Byzantine k-set agreement problem in the system model $\mathcal{BAMP}_{n,t}[t < n/(m+1), \text{LRC}]$.

Proof The KS-Validity property follows from Lemma 1. The KS-Agreement property follows from Lemma 5. The KS-RR-Termination follows from Lemma 9. $\square_{Theorem\ 4}$

5 Conclusion

This paper presented a signature-free randomized distributed algorithm that solves k-set agreement in asynchronous message-passing systems where up to $t \geq k$ processes may commit Byzantine failures.

Its design is based on a modular construction which 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 [14] to the multivalue domain. An interesting feature of the propsed algorithm lies in the validity condition it ensures, namely, no value proposed only by Byzantine processes can be decided by non-faulty processes.

Acknowledgments

This work has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing, and the Franco-German project DISCMAT devoted to the mathematics of distributed computing.

References

[1] Attiya H. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, (2d Edition), Wiley-Interscience, 414 pages, 2004.

[2] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computing (STOC'93)*, ACM Press, pp. 91-100, 1993.

[3] Bouzid Z., Mostéfaoui A., and Raynal M., Minimal synchrony for Byzantine consensus. *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC'15)*, ACM Press, 2015.

[4] Bracha G., Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2):130-143, 1987.

[5] Censor Hillel K., Multi-sided shared coins and randomized set agreement. *Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'10)*, ACM Press, pp. 60-68, 2010.

[6] Chaudhuri S., More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132158, 1993.

[7] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985.

[8] Friedman R., Mostéfaoui A., and Raynal M., Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46-56, 2005.

[9] Gafni E. and Guerraoui R., Generalizing universality. *Proc. 22nd Int'l Conference on Concurrency Theory (CONCUR'11)*, Springer LNCS 6901, pp. 17-27, 2011.

[10] Herlihy M.P., Kozlov D., and Rajsbaum S., *Distributed computing through combinatorial topology*, Morgan Kaufmann/Elsevier, 336 pages, 2014.

[11] Herlihy M., Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[12] Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401, 1982.

[13] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996 (ISBN 1-55860-384-4).

[14] Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM*, To appear, 2015.

[15] Mostéfaoui A. and Raynal M., Randomized k-set agreement. *Proc. 12nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'00)*, ACM Press, pp. 291-297, 2000.

[16] Mostéfaoui A. and Raynal M., Signature-free broadcast-based intrusion tolerance: never decide a Byzantine value. *Proc. 14th Int'l Conference On Principles Of Distributed Systems (OPODIS'10)*, Springer LNCS 6490, pp. 144-159, 2010.

[17] Pease M., R. Shostak R., and Lamport L., Reaching agreement in the presence of faults. *J. of the ACM*, 27:228-234, 1980.

[18] de Prisco R., Malkhi D., and Reiter M.K., On k-Set consensus problems in asynchronous systems. *IEEE Transactions on Parallel Distributed Systems*, 12(1):7-21, 2001.

[19] Rabin M., Randomized Byzantine generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 116-124, 1983.

[20] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[21] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, 2013 (ISBN 978-3-642-32026-2).

[22] Raynal M., Stainer J., and Taubenfeld G., Distributed universality. *Proc. 18th Int'l Conference on Principles of Distributed Systems (OPODIS'14)*, Springer LNCS 8878, pp. 469-484, 2014.

[23] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

[24] Srikanth T.K. and Toueg S., Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80-94, 1987.

[25] Toueg S., Randomized Byzantine agreement. *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, ACM Press, pp. 163-178, 1984.