Signature-Free Asynchronous Binary Byzantine Consensus with $t < n/3$, $O(n^2)$ Messages, and $O(1)$ Expected Time

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

Abstract

This paper is on broadcast and agreement in asynchronous message-passing systems made up of n processes, and where up to t processes may have a Byzantine Behavior. Its first contribution is a powerful, yet simple, all-to-all broadcast communication abstraction suited to binary values. This abstraction, which copes with up to $t < n/3$ Byzantine processes, allows each process to broadcast a binary value, and obtain a set of values such that (1) no value broadcast only by Byzantine processes can belong to the set of a correct process, and (2) if the set obtained by a correct process contains a single value v, then the set obtained by any correct process contains v.

The second contribution of the paper is a new round-based asynchronous consensus algorithm that copes with up to $t < n/3$ Byzantine processes. This algorithm is based on the previous binary broadcast abstraction and a weak common coin. In addition of being signature-free and optimal with respect to the value of t, this consensus algorithm has several noteworthy properties: the expected number of rounds to decide is constant; each round is composed of a constant number of communication steps and involves $O(n^2)$ messages; each message is composed of a round number plus a constant number of bits. Moreover, the algorithm tolerates message re-ordering by the adversary (i.e., the Byzantine processes).

Keywords: Asynchronous message-passing system, Broadcast abstraction, Byzantine process, Common coin, Consensus, Distributed algorithm, Optimal resilience, Randomized algorithm, Signature-free algorithm, Simplicity.

1 Introduction

Distributed computing Distributed computing occurs when one has to solve a problem in terms of physically distinct entities (usually called nodes, processors, processes, agents, sensors, etc.) such that each entity has only a partial knowledge of the many parameters involved in the problem. In the following, we use the term *process* to denote any computing entity. From an operational point of view this means that the processes of a distributed system need to exchange information, and agree in some way or another, in order to cooperate to a common goal. Hence, a distributed system has to provide the processes with communication and agreement abstractions.

Understanding and designing distributed applications is not an easy task [2, 10, 27, 38, 39, 40]. This is because, due to the very nature of distributed computing, no process can capture instantaneously the global state of the application it is part of. This comes from the fact that, as processes are geographically localized at distinct places, distributed applications have to cope with the uncertainty created by asynchrony and failures. As a simple example, it is impossible to distinguish a crashed process from a very slow process in an asynchronous system prone to process crashes.

As in sequential computing, a simple approach to facilitate the design of distributed applications consists in designing appropriate abstractions. More generally, computing science is a science of abstraction and distributed computing is the science of distributed abstractions [17]. With such abstractions, the application designer can think about solutions to solve problems at a higher conceptual level than the basic send/receive communication level.

Communication and agreement abstractions Broadcast abstractions are among the most important abstractions required to address fault-tolerant distributed computing [2, 10, 12, 27, 38]. Roughly speaking, these abstractions allow processes to disseminate information in such a way that specific provable properties concerning this dissemination are satisfied. One of the most popular of these abstractions is reliable broadcast [7, 12].

As far as agreement abstractions are concerned, *non-blocking atomic commit* [22, **?**] and *consensus* [16, 36] are certainly the most important abstractions of fault-tolerant distributed computing. Assuming that each process proposes a value, the consensus abstraction allows the non-faulty processes to agree on the same value, which has to satisfy some validity condition depending on both the proposed values and the failure model [2, 27, 38, 39].

Asynchronous Byzantine systems This paper is on all-to-all broadcast abstractions and their use to solve binary consensus in an asynchronous message-passing system where processes can commit Byzantine failures [25]. This failure type has first been introduced in the context of synchronous distributed systems [25, 36, 39], and then investigated in the context of asynchronous distributed systems [2, 27, 38]. A process has a *Byzantine* behavior when it arbitrarily deviates from its intended behavior: it then commits a Byzantine failure (otherwise we say it is *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.

Consensus related work Let t denote the model upper bound on the number of processes that can have a Byzantine behavior. It is shown in several papers (e.g., [14, 25, 36, 43]) that Byzantine consensus cannot be solved when $t \geq n/3$, be the system synchronous or asynchronous, be the algorithm allowed to use cryptography or not, or be the algorithm allowed to use random numbers or not. As far as synchronous systems are concerned, it has been shown in [15] that $(t+1)$ rounds is a lower bound on the number of communication steps needed to solve Byzantine consensus. It has also been shown in [21] that, using randomization, this bound can be improved to an expected $O(\log n)$ value in systems that are full-information, synchronous, and where where $t \leq \frac{n}{3+\epsilon}$ and $\epsilon > 0$.

In asynchronous systems, it is well-known that there is no deterministic consensus algorithm as soon as one process may crash [16], which means that Byzantine consensus cannot be solved either if one process can be faulty. Said another way, the basic asynchronous Byzantine system model has to be enriched with additional computational power. Such an additional power can be obtained by randomization (e.g., [3, 13, 20, 37]), assumption on message delivery schedules [7, 43], failure detectors suited to Byzantine systems (e.g., [19, 23]), additional –deterministic or probabilistic– synchrony assumptions (e.g., [7, 14, 29]), or restrictions on the vectors of input values proposed by the processes [18, 32]. As this paper focuses on binary consensus (i.e., the consensus problem where only two values can be proposed by processes), it considers that the system is enriched with the additional computational power supplied by a *common coin*. Such an object, introduced by Rabin [37], is a distributed object that is expected to deliver, with some probability, the same sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to each process.

Some of the first randomized consensus algorithms (such as the ones of Ben-Or [3] or Bracha [5]) use local coins (the values returned to a process by a local coin is not related to the values returned by the local coins of the other processes). As a consequence they have an expected number of rounds which is exponential (unless $t = O(\sqrt{n})$), which makes them of theoretical interest, but of limited practical use. As randomized algorithms based on a common coin can have an expected number of rounds which is constant, this paper focuses only on such algorithms.

Protocol	$n >$	signatures	msgs/round	bits/msg
[37]	$10t$	yes	$O(n^2)$	$O(1)$
[4]	$5t$	no	$O(n^2)$	$O(1)$
[20]	$5t$	no	$O(n^2)$	$O(1)$
[6]	$3t$	no	$O(n^3)$	$O(\log(n))$
[42]	$3t$	no	$O(n^3)$	$O(\log(n))$
[43]	$3t$	yes	$O(n^3)$	$O(n)$
[11]	$3t$	yes	$O(n^2)$	poly(n)
[8]	$3t$	yes	$O(n^2)$	$O(\ell)$
This paper	$3t$	no	$O(n^2)$	$O(1)$

Table 1: Cost and constraint of different Byzantine binary consensus algorithms

Round-based asynchronous Byzantine algorithms using a common coin are listed in Table 1 (at the line before the last line, ℓ denotes the length of an RSA signature)[1].

All these algorithms, which address binary consensus, are round-based, and each message carries the

[1]For synchronous systems, an efficient coin-based algorithm that solves consensus despite Byzantine processes is described in [24].

number of the round in which it is sent. Hence, when comparing their message size, we do not consider round numbers. We have the following.

- The first algorithm is such that $n < 10t$, has an $O(n^2)$ message complexity, and requires signatures.
- The algorithms of the two next lines are such that $t < n/5$, and their message complexity is $O(n^2)$. These algorithms are simple, signature-free, and use one or two communication steps per round, but none of them is optimal with respect to t-resilience.
- The algorithms of the next three lines are optimal with respect to t, but have an $O(n^3)$ message complexity. Moreover, [43] uses signed messages[2], while [6] does not use a common coin, and may consequently execute an exponential number of rounds. Due to their message complexity, these algorithms are costly.
- The algorithm proposed in [11], although polynomial, has a huge bit complexity. The algorithm presented in [8] is optimal with respect to t, uses only $O(n^2)$ messages per round, has two communication steps per round, and uses signed messages. However, as noticed by its authors, "because of public key operations, the computational complexity of this protocol is typically higher than those using authentication codes".

Content of the paper The paper first introduces a new, yet simple, *all-to-all* broadcast abstraction suited to binary values, that we call DSBV (*Double Synchronized Binary Value*) broadcast. The corresponding broadcast operation, denoted DSBV_broadcast(), allows each process to broadcast an input value and obtain a non-empty set of at most two values called a *view*. A novel feature of this operation lies in the fact that it focuses on values and not on processes, which means that it does not consider the fact that "this" value has been broadcast by "this" process. More explicitly, if a value v has been broadcast by the processes p and q, what is important is only the fact that v was broadcast by two distinct processes. The second feature of DSBV-broadcast lies in the fact that it reduces the power of Byzantine processes in such a way that no view delivered at a correct process contains values broadcast only by Byzantine processes, and, if the set delivered to a correct process contains a single value v, then the set delivered to any correct process contains this value v. The last noteworthy feature of DSBV-broadcast lies in its implementation, which is particularly simple and easy to prove. Then, the paper uses this all-to-all value-oriented broadcast abstraction to solve Byzantine binary consensus in an asynchronous message-passing system enriched with a common coin. The resulting consensus algorithm is a round-based asynchronous algorithm that has the following noteworthy properties.

- The algorithm requires $t < n/3$ and is consequently optimal with respect to t.
- The algorithm uses a constant number of communication steps per round.
- The expected number of rounds to decide is constant.
- The message complexity is $O(n^2)$ messages per round.
- Each message carries its type, a round number plus a constant number of bits.

[2] Signatures are used to prevent message falsification by Byzantine processes.

- Byzantine processes may re-order messages sent to correct processes.
- The algorithm uses a weak coin. *Weak* means here that there is a constant probability that, at every round, the coin returns different values to distinct processes.
- Finally, the algorithm does not assume a computationally-limited adversary (and consequently it does rely on signed messages).

Hence, contrarily to what one could believe from existing asynchronous Byzantine binary consensus algorithms, the formula

$$\text{[quadratic message complexity]} \Rightarrow \text{[(use of signatures)} \vee (t < n/5)]$$

is false. The proposed algorithm shows that $t < n/3$ (as in [8]), quadratic message complexity (as in [4, 8]), and absence of signatures (as in [4, 20]) are not incompatible.

Remark The algorithm presented in this paper is an improved version of the algorithm presented in [31], which requires a perfect common coin and considers a fair message scheduling (i.e., it assumes that the adversary -Byzantine processes- cannot re-order messages sent by correct processes). Both the algorithm presented in the paper and the previous one have the same asymptotic upper bound on time and message cost.

Roadmap The paper is composed of 6 sections. Section 2 presents the computation model. Section 3 presents and proves correct the DSBV broadcast abstraction. Section 4 presents and proves correct a consensus algorithm, based on a common coin, in which the correct processes *decide* in a constant expected number of rounds. Then, Section 5 extends the previous algorithm to obtain a binary consensus algorithm in which the correct processes *decide and terminate* in a constant expected number of rounds. Finally, Section 6 concludes the paper.

2 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 that has been sent is eventually received by its destination process, i.e., there is no bound on message transfer delays. "Reliable" means that the network does not lose, 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 commits a

failure in the middle of the **for** loop, it is possible that only an arbitrary subset correct processes receives the 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*. 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. Hence, 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.

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, $\mathcal{BAMP}_{n,t}[n > 3t]$ denotes the model $\mathcal{BAMP}_{n,t}[\emptyset]$ where the number of faulty processes is smaller than $n/3$, and $\mathcal{BAMP}_{n,t}[n > 3t, \text{CC}]$ denotes the model $\mathcal{BAMP}_{n,t}[n > 3t]$ enriched with a common coin (CC) abstraction [37] (definition and details of CC will be given Section 4.1). Let us notice that, as CC belongs to the model, it is given for free in $\mathcal{BAMP}_{n,t}[n > 3t, \text{CC}]$.

Time complexity When computing the time complexity we ignore local computation time, and consider the longest sequence of messages m_1, m_2, ..., m_z which are causally related (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.

3 The Double Synchronized Binary-Value Broadcast Abstraction

This section introduces the DSBV broadcast abstraction, which (as announced) will be at the core of the binary consensus algorithm presented in Section 4. This abstraction is built modularly on top of an underlying abstraction called SBV (*Strong Binary Value*) broadcast, which is itself built on top of an extremely simple broadcast abstraction called BV (*Binary Value*) broadcast.

These three broadcast abstractions are one-shot, all-to-all, and binary. "One-shot" means that, in each broadcast instance, a correct process invokes at most once the corresponding communication operation. "All-to-all" means that, by assumption, all the correct processes invoke the corresponding broadcast operation. "Binary value" means that only two different predefined values a and b can be broadcast.

3.1 Binary-value Broadcast

Definition This all-to-all communication abstraction (in short, BV-broadcast) provides the processes with a single operation denoted BV_broadcast(). When a process invokes BV_broadcast TAG(m), we say that it "BV-broadcasts the message typed TAG and carrying the binary value m". The invocation of BV_broadcast

TAG(m) does not block the invoking process. The aim of BV-broadcast is to eliminate a value (if any) that has been broadcast only by Byzantine processes.

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

- BV-Termination. The invocation of BV_broadcast() by a correct process terminates.
- BV-Justification. If p_i is correct and $v \in bin\_values_i$, v has been BV-broadcast by a correct process.
- BV-Uniformity. If a value v is added to the set bin_values_i of a correct process p_i, eventually $v \in bin\_values_j$ at every correct process p_j.
- BV-Obligation. Eventually the set bin_values_i of each correct process p_i is not empty.

Let us observe that, as only two values can be BV-broadcast, the following properties are an immediate consequence of the previous definition.

- BV-Single-value. If all the correct processes BV-broadcast the same value v, v is eventually added to the set bin_values_i of each correct process p_i.
- The sets bin_values_i of the correct processes p_i eventually (a) become non-empty and equal, (b) do not contain values broadcast only by Byzantine processes.

A BV-broadcast Algorithm A simple algorithm implementing the BV-broadcast abstraction is described in Figure 1. This algorithm is based on a particularly simple "echo" mechanism. Differently from previous echo-based algorithms (e.g., [6, 42]), 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, in the algorithm of Figure 1, a value entails a single echo, whatever the number of processes that broadcast this value.

let $witness(v)$ = number of different processes from which B_VAL(v) was received;
bin_value_i is initially empty.

operation BV_broadcast MSG(v_i) **is**
(1) broadcast B_VAL(v_i); return().

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

Figure 1: An algorithm implementing BV-broadcast in $\mathcal{BAMP}_{n,t}[n > 3t]$

When a process p_i invokes BV_broadcast MSG(v_i), $v_i \in \{a, b\}$, it broadcasts B_VAL(v_i) to all the processes (line 1). Then, when a process p_i receives (from any process) a message B_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 bin_values_i (lines 5-6).

Remark Let us notice that, when bin_values_i contains a single value, process p_i cannot know if bin_values_i has obtained its final value. (Otherwise, consensus will be directly obtained by directing each process p_i to deterministically extract the same value from bin_values_i). This impossibility is due to the net effect of asynchrony and process failures [16].

Theorem 1. *The algorithm described in Figure* 1 *implements the* BV-broadcast *abstraction in the system model* $\mathcal{BAMP}_{n,t}[t < n/3]$.

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

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

Proof of the BV-Uniformity property. If a value v is added to the set bin_values_i of a correct process p_i (local delivery), this process has received B_VAL(v) from at least $(2t + 1)$ different processes (line 5), i.e., from at least $(t + 1)$ different correct processes. As each of these correct processes has 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 B_VAL(v) to all. As $n - t \geq 2t + 1$, the predicate of line 5 is then eventually satisfied at each correct process, and the BV-Uniformity property follows.

Proof of the BV-Obligation property. As (a) there are at least $(n - t)$ correct processes, (b) each of them invokes BV_broadcast MSG(), (c) $n - t \geq 2t + 1 = (t + 1) + t$, and (d) only two values a and b can be BV-broadcast, it follows that there is a value $v \in \{a, b\}$ that is BV-broadcast by at least $(t + 1)$ correct processes. Hence, each correct process eventually receives a message B_VAL(v) from $(t + 1)$ processes, and then forwards this message (if not already done). Consequently, the predicate of line 5 becomes satisfied and v is added to bin_value_i at every correct process, from which follows the BV-Obligation property. $\square_{Theorem\ 1}$

Cost of the algorithm We have the following for each BV-broadcast instance.

- If all correct processes BV-broadcast the same value, the algorithm requires a single communication step. Otherwise, it can require two communication steps.
- Let $c \geq n - t$ be the number of correct processes. The correct processes send $c\ n$ messages if they BV-broadcast the same value, and at most $2\ c\ n$ messages otherwise.
- In addition to the control tag B_VAL, a message carries a single bit. Hence, message size is constant.

3.2 Synchronized Binary-value Broadcast

Definition This all-to-all communication abstraction (in short SBV-broadcast) provides the processes with a single operation denoted SBV_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 SBV-broadcast instance, each correct process invokes SBV_broadcast TAG(m), where TAG is the type of the message and $m \in \{a, b\}$ the binary value that it wants to broadcast. Such an invocation returns to the invoking process p_i a set denoted $view_i$ and called a (local) view. SBV-broadcast is defined by the following properties.

- SBV-Termination. The invocation of SBV_broadcast TAG() by a correct process terminates.
- SBV-Obligation. The set $view_i$ returned by a correct process p_i is not empty.
- SBV-Justification. If p_i is correct and $v \in view_i$, then a correct process SBV-broadcast v.
- SBV-Inclusion. If p_i and p_j are correct processes and $view_i = \{v\}$, then $v \in view_j$.

Let us observe that the first property that follows is an immediate consequence of the previous SBV-Justification and SBV-Obligation properties, while the second one is an immediate consequence of the previous SBV-inclusion property.

- SBV-Uniformity. If all correct processes SBV-broadcast the same value v, then $view_i = \{v\}$ at every correct process p_i.
- SBV-Singleton. If p_i and p_j are correct, $[(view_i = \{v\}) \wedge (view_i = \{w\})] \Rightarrow (v = w)$.

An SBV-broadcast algorithm An algorithm implementing the SBV-broadcast abstraction is described in Figure 2. A process p_i first BV-broadcasts a message MSG (v_i) and waits until the associated set bin_values_i is not empty (lines 1-2). Let us remind that, when p_i stops waiting, bin_values_i has not necessarily obtained its final value. Then, p_i extracts a value w from bin_values_i and broadcasts it to all (line 3). If there are two values in bin_values_i, w is any of them. Moreover, let us remind that a Byzantine process can send messages AUX() carrying different values to distinct processes.

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

Figure 2: An algorithm implementing SBV-broadcast in $\mathcal{BAMP}_{n,t}[n > 3t]$

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. The second 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 every correct process p_j. From an operational point of view, this translates as follows at a process p_i. There is a set ($view_i$) of values received in messages AUX() from $(n-t)$ different processes, and these values originated from correct processes (let us remind that bin_values_i can contain only values BV-broadcast by correct processes). Hence, at line 5, we have $view_i \subseteq \{a, b\}$, and for any $v \in view_i$, v is a value SBV-broadcast by a correct process.

Theorem 2. *The algorithm described in Figure* 2 *implements the* SBV-broadcast *abstraction in the system model* $\mathcal{BAMP}_{n,t}[t < n/3]$.

Proof Proof of the SBV-Termination property. Let us first observe that, due to the BV-Termination property and the BV-Obligation property of the underlying BV-broadcast, a correct process cannot block forever at line 2. As there are at least $(n-t)$ correct processes, it follows that any correct process p_i receives values at least from these $(n-t)$ processes and, due to the BV-Justification property, these values have been SBV-broadcast by correct processes. It follows that the predicate of line 4 becomes eventually satisfied at any correct process p_i, and consequently the invocations of SBV_broadcast() by the correct processes terminate.

Proof of the SBV-Obligation property. Any correct process p_i eventually receives $(n-t)$ messages AUX() sent by correct processes. As these messages carry values taken from the set bin_values of 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 SBV-Justification property. This proof follows directly from the BV-Justification property, namely, the set bin_values_i of a correct process contains only values BV-broadcast by correct processes.

Proof of the SBV-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 received the same message AUX(v) from at least $(n-t)$ different processes. As at most t processes can be Byzantine, it follows that p_i received 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 received messages AUX() from at least $(n-t)$ different processes. As $(n-t) + (t+1) > n$, it follows that a correct process p_x sent the same message AUX(v) to p_i and p_j, and consequently $v \in view_j$, which concludes the proof of the lemma. $\square_{Theorem\ 2}$

Cost of the algorithm We have the following for each SBV-broadcast instance.

- If all the correct processes SBV-broadcast the same value, the algorithm requires two communication steps (one in the underlying BV-broadcast plus the exchange of the AUX() messages). Otherwise, it can require three communication steps.
- Let $c \geq n - t$ be the number of correct processes. The correct processes send $2c\,n$ messages if they SBV-broadcast the same value, and at most $3\,c\,n$ messages otherwise.
- In addition to the control tag MSG, a message carries a single bit. Hence, message size is constant.

3.3 Double Synchronized Binary-value Broadcast

Definition This all-to-all communication abstraction (in short DSBV-broadcast) provides the processes with a single operation denoted DSBV_broadcast(). When a correct process p_i invokes DSBV_broadcast TAG(m), it broadcasts the binary value m and, as in the SBV broadcast, it obtains a view $view_i$, which may

now contain a default value denoted $\perp$ and at most one of the broadcast values (either a or b). This means that if p_i is correct and $view_i$ contains a, no correct process has a view containing b. DSBV-broadcast is defined by the following properties.

- DSBV-Termination. The invocation of DSBV_broadcast TAG() by a correct process terminates.
- DSBV-Obligation. The set $view_i$ returned by a correct process p_i is such that $1 \leq |view_i| \leq 2$.
- DSBV-value-Justification. If p_i is correct, $v \in view_i$ and $v \neq \perp$, then a correct process DSBV-broadcast v.
- DSBV-bottom-Justification. If p_i is correct and $\perp \in view_i$, then both a and b have been DSBV-broadcast by correct processes.
- DSBV-Inclusion. If p_i and p_j are correct processes and $view_i = \{v\}$ (where v is a value DSBV-broadcast by a correct process, or $\perp$), then $v \in view_j$.

Let us observe that the following properties are immediate consequences of the previous justification, obligation, and inclusion, properties.

- DSBV-Uniformity. If all correct processes DSBV-broadcast the same value v, then $view_i = \{v\}$ at every correct process p_i.
- DSBV-Mutex. If both p_i and p_j are correct, and $v \neq \perp$, $view_i = \{v\}$ and $view_j = \{\perp\}$ are mutually exclusive.
- DSBV-Singleton. If p_i and p_j are correct, $[(view_i = \{v\}) \wedge (view_j = \{w\})] \Rightarrow (v = w)$.

```
operation DSBV_broadcast MSG(v_i) is
(1)  view_i[0] ← SBV_broadcast STAGE[0](v_i);
(2)  if (view_i[0] = {v}) then aux_i ← v else aux_i ← ⊥ end if;
(3)  view_i[1] ← SBV_broadcast STAGE[1](aux_i);
(4)  return (view_i[1]).
```

Figure 3: An algorithm implementing DSBV-broadcast in $\mathcal{BAMP}_{n,t}[n > 3t]$

A DSBV-broadcast algorithm A very simple DSBV-broadcast algorithm is described in Figure 3. This algorithm consists in two sequential instances of SBV-broadcast (hence its name), separated by an "if" statement. Using an array-like notation, the local variables and messages of each instance are tagged with an additional bit. The aim is to replace the (possibly two) values v and w, which are DSBV-broadcast by the correct processes, by at most one of them (either v or w) plus possibly the default value $\perp$. Hence, if both v and w have been DSBV-broadcast, one of them is eliminated. To this end, the first invocation (line 1) returns at each correct process p_i a view $view_i[0]$, that is then locally filtered by an "if" statement (line 2) so that at most one of the values v or w DSBV-broadcast by the correct processes will be SBV-broadcast at line 3. Let y be this value (if any). It follows that the binary set $\{a, b\}$ of values that can be SBV-broadcast at line 3 is the set $\{y, \perp\}$. Hence, the view returned by a correct process p_i at line 4 is a subset of $\{y, \perp\}$ satisfying (among others) the DSBV-Mutex and DSBV-Singleton properties stated above.

Theorem 3. *The algorithm described in Figure* 2 *implements the* DSBV-broadcast *abstraction in the system model* $\mathcal{BAMP}_{n,t}[t < n/3]$.

Proof Proof of the DSBV-Termination property. The proof follows directly from the termination property of the SBV-broadcast instances.

Proof of the DSBV-Obligation property. Due to singleton property of the first SBV-broadcast instance, no two correct processes can be such that $view_i[0] = \{v\}$ and $view_j[0] = \{w\}$, where v and w have been SBV-broadcast at line 1 and are such that $v \neq w$. Let C be the set of correct processes. It follows that the set $AUX = \cup_{x \in C}\{aux_x\}$ is equal to either $\{v\}$, or $\{\bot\}$, or $\{v, \bot\}$, where v is a value SBV-broadcast at line 1 by a correct process. It then follows from the justification property of the second SBV-broadcast instance that the set $view_i[1]$ of any correct process p_i is such that $view_i[1] \subseteq AUX$, and consequently contains at most two values.

Proof of the DSBV-value-Justification property. If p_i is correct and $view_i[1] = \{v\}$ where $v \neq \bot$, it follows from the justification property of the second SBV-broadcast that a correct process p_x invoked SBV_broadcast STAGE$[1](v)$ at line 3. As the "if" statement of line 2 does not add new non-$\bot$ values, it follows that $view_x$ was such that $view_x[0] = \{v\}$. We then conclude from the justification property of the first SBV-broadcast instance that a correct process invoked SBV_broadcast STAGE$[0](v)$ at line 1, which concludes the proof of the DSBV-value-Justification property.

Proof of the DSBV-bottom-Justification property. The proof is similar to the previous one. If there is a correct process p_i such that $view_i[1] = \{\bot\}$, there is a correct process p_x that invoked SBV_broadcast STAGE$[1](\bot)$ at line 3. It follows from line 2 and the SBV-Justification property of the first SBV-broadcast instance that this process is such that $view_x[0] = \{v, w\}$, where v and w are different and were SBV-broadcast at line 1 by two correct processes, which concludes the proof of the property.

Proof of the DSBV-Inclusion property. The proof follows directly from the inclusion property of the second SBV-broadcast instance (line 3). $\square_{Theorem\ 3}$

Cost of the algorithm The cost of a DSBV-broadcast instance is twice the cost of an SBV-broadcast. Hence, we have the following.

- If all the correct processes DSBV-broadcast the same value, the algorithm requires four communication steps (two in each underlying SBV-broadcast). Otherwise, it can require up to six communication steps.
- Let $c \geq n - t$ be the number of correct processes. The correct processes send $4c\ n$ messages if they DSBV-broadcast the same value, and at most $6\ c\ n$ messages otherwise.
- In addition to the message tags needed by the two underlying SBV-broadcast instances, an additional bit is needed to distinguish these instances. The message size is consequently constant.

4 A Byzantine Consensus Algorithm

This section presents and proves correct a Byzantine binary consensus algorithm, which is built on top of the DSBV-broadcast communication abstraction and a weak common coin.

4.1 Byzantine Consensus and Enriched Model

Binary Byzantine consensus The Byzantine consensus abstraction has been informally stated in the Introduction. It provides the processes with a single operation denoted propose(), which returns a value (we then say that the corresponding process *decides* the returned value).

Assuming that each correct process p_i invokes propose(v_i), where $v_i \in \{0, 1\}$ is the value it proposes to the consensus, each correct process has to decide a value such that the following properties are satisfied.

- BC-Validity. A decided value was proposed by a correct process.
- BC-Agreement. No two correct processes decide different values.
- BC-Decision. Each correct process decides.

The BC-Validity property states that no value proposed only by faulty processes can be decided. As we consider binary consensus, it is equivalent to the following property: if all correct processes propose the same value v, the value $\overline{v}$ cannot be decided (where $\overline{v}$ is the other binary value).

From binary to multivalued Byzantine consensus Interestingly, asynchronous multivalued Byzantine consensus (i.e., when more than two values can be proposed) can be solved on top of binary Byzantine consensus. Such constructions are described in [9, 13, 26, 30, 33, 34, 35, 41] (see [44] for synchronous systems).

Enriching the basic Byzantine asynchronous model with a common coin As indicated in the Introduction, the model $\mathcal{BAMP}_{n,t}[t < n/3]$ is not computationally strong enough to solve consensus. It has to be enriched so that Byzantine consensus can be solved. The additional computational power we consider here is a *common coin* (CC) as introduced by Rabin [37][3]. As already said, the corresponding enriched system model is denoted $\mathcal{BAMP}_{n,t}[t < n/3, \mathrm{CC}]$ [4].

A common coin is "a random source observable by all participants but unpredictable by an adversary" [8]. More explicitly, a common coin is a global entity delivering to each correct process p_i a sequence of random bits $b_i[1], b_i[2], \ldots, b_i[r], \ldots$ such that, for each r, the bits $b_i[r]$, $b_j[r]$,, of all the correct processes p_i, $p_j, \ldots$, are such that (1) they all have the value 0 with probability $1/d$, (2) they all have the value 1 with probability $1/d$, and (3) some correct processes obtain the value 0 and the other correct processes obtain the value 1 with probability $(d-2)/d$, where $d \geq 2$ is a known constant. A perfect common coin corresponds to the case $d = 2$ (namely, for any r, the correct processes obtain the same bit –0 or 1– with probability $1/2$). If $d > 2$ the common coin is *weak*. From an operational point of view, this random oracle provides the processes with a primitive denoted random() such that the r-th invocation by p_i returns it the bit $b_i[r]$.

A common coin demands the processes to cooperate to compute the value of the bit during the round r, namely, no Byzantine process p_j can obtain the value of this random bit before at least one correct process started its r-th invocation of random() [10]. This is required to prevent Byzantine processes to compute

[3]The common coin algorithm described in [37] is based on a trusted dealer. Differently, the algorithm presented in [8] does require a trusted dealer.

[4]Hence, as noticed in Section 2, in the model $\mathcal{BAMP}_{n,t}[t < n/3, \mathrm{CC}]$, CC is given for free. A practical common coin implementation is described in [8]. It is important to notice that, if cryptography is ever used to build an underlying common coin, the paper shows that cryptography is no longer needed at the upper layer where Byzantine consensus is solved. This contrasts with the signature-based algorithms cited in Figure 1.

random bit values in advance and exploit these values to produce message delivery schedules that would prevent termination.

On randomized consensus When relying on the additional computing power provided by common coins, the BC-decision property can no longer be deterministic. *Randomized consensus* is defined by BC-Validity (Obligation), BC-Agreement, plus the following BC-Decision property [3, 37]: Every non-faulty process decides with probability 1. For round-based algorithms, this decision property is re-stated as follows. For any correct process p_i, we have:

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

4.2 Randomized Byzantine Consensus Algorithm

Principles and description of the algorithm The consensus algorithm is described in Figure 4. It requires $t < n/3$ and is consequently optimal with respect to the maximal number of Byzantine processes that can be tolerated. A process p_i invokes propose(v_i) where $v_i \in \{0, 1\}$ is the value it proposes. It decides when it executes the statement decide(v) at line 6. The processes proceed in consecutive asynchronous rounds, where a round consists of two phases, each made up of a single DSBV-broadcast instance.

The local variable est_i of a process p_i keeps its current estimate of the decision (initially $est_i = v_i$). The local variable r_i denotes the current round of process p_i, while the local variables $view_i[r_i, 1..2]$ contain the results of the DSBV-broadcast instances used in round r_i. As these two variables are used one after the other, a single variable $view_i$ could be used. As previously, we use an array-like notation to make the presentation clearer. The behavior of a correct process p_i during a round r is as follows.

```
operation propose(v_i) is
 est_i ← v_i; r_i ← 0;
repeat forever
(1)    r_i ← r_i + 1;
// ——————————————————————— phase 1 ———————————————————————
(2)    view_i[r_i, 1] ← DSBV_broadcast PHASE[r_i, 1](est_i)
(3)    b_i[r_i] ← random();
(4)    if (view_i[r_i, 1] = {v}) ∧ (v ≠ ⊥)) then est_i ← v else est_i ← b_i[r_i] end if;
// ——————————————————————— phase 2 ———————————————————————
(5)    view_i[r_i, 2] ← DSBV_broadcast PHASE[r_i, 2](est_i);
(6)    case (view_i[r_i, 2] = {v})      then decide(v) if not yet done end if
(7)         (view_i[r_i, 2] = {v, ⊥})   then est_i ← v
(8)         (view_i[r_i, 2] = {⊥})      then skip
(9)    end case
end repeat.
```

Figure 4: A binary consensus algorithm based on DSBV-broadcast and a common coin

- Phase 1: lines 2-4. The aim of this phase is to help the processes agree on a single value. To that end, the processes execute first a DSBV-broadcast instance (line 2), at the end of which the view of any correct process is such that $view_i[r, 1] = \{v\}$, or $view_i[r, 1] = \{v, \bot\}$, or $view_i[r, 1] = \{\bot\}$, where $v \in \{0, 1\}$ is a value that was DSBV-broadcast, and $view_i[r, 1] = \{0\}$ and $view_j[r, 1] = \{1\}$ cannot co-exist if p_i and p_j are correct processes.

Then the processes compute the random number s associated with the current round (line 3). Finally, if a correct process p_i has seen a single DSBV-broadcast value (i.e., $view_i[r,1] = \{v\}$ and $v \neq \perp$), it adopts it as new estimate. Otherwise, it has seen both 0 and 1, and adopts as new estimate the random value $b_i[r]$ locally output by the common coin; this is to try breaking the non-determinism created by the fact that both the values 0 and 1 have been DSBV-broadcast (line 4).

Due to the use of the common coin, it follows (as we will see in the proof) that, if the algorithm was a simple "repeat" loop from which the second phase was suppressed (lines 5-9), there is a finite round number r such that, from round r, all the correct processes have forever the same estimate. As no process can know when this round occurs, the issue that remains to be solved is to help decide the processes when the previous pattern occurs. This is the role of the second phase.

- Phase 2: lines 5-9. As just indicated, the second phase aims to help the correct processes decide. To that end, the processes first execute a DSBV-broadcast instance (line 5), at the end of which the view of any correct process p_i is such that $view_i[r,2] = \{v\}$, or $view_i[r,2] = \{v, \perp\}$, or $view_i[r,2] = \{\perp\}$, where $v \in \{0,1\}$ is a value that was DSBV-broadcast at line 2, and $view_i[r,2] = \{v\}$ and $view_j[r,2] = \{\perp\}$ cannot co-exist if both p_i and p_j are correct processes.

  This mutual exclusion property is used to direct the processes to decide without violating the consensus BC-Agreement property. The corresponding behavior for a correct process is captured by the case statement of lines 6-8. If p_i sees only v (different from $\perp$), it decides it (line 6). If it sees both a binary value v and $\perp$, it adopts v as new estimate (line 7). Finally, if it sees only $\perp$, it does not change its estimate value.

Let us notice that the statement decide() (line 6) allows the invoking process to decide, but does not stop its execution. This facilitates the description and the understanding of the algorithm. A terminating version of the algorithm will be presented in Section 5.

4.3 Proof and Cost of the Algorithm

Lemma 1. *Let $n > 3t$. A decided value is a value proposed by a correct process.*

Proof Let us consider the round $r = 1$. It follows from the DSBV-value-Justification property (line 2), that the set $view_i[1,1]$ of any correct process p_i contains only values proposed by correct processes. Moreover, it follows from DSBV-bottom-Justification property that if $view_i[1,1]$ contains more than one value, both 0 and 1 have been DSBV-broadcast at line 2. Consequently, if p_i assigns the random value s to est_i, the value s has been proposed by a correct process. It follows that the set $view_i[1,2]$ obtained by a correct process can contain only values proposed by correct processes. Hence, if a correct process assigns a value v to est_i at line 7, this value was proposed by a correct process.

By induction on the round number, the same reasoning applies to all other rounds, from which it follows that an estimate value decided at line 6 is a value that was proposed by a correct process. $\square_{Lemma\ 1}$

Lemma 2. *Let $n > 3t$. If, at the beginning of a round r, all the correct processes have the same estimate value v, they never change their estimates thereafter.*

Proof If all the correct processes (which are at least $n - t > t + 1$) have the same estimate value v at beginning of a round r, they all DSBV-broadcast the message PHASE$[r,1](v)$ at line 2. It follows from

the DSBV-value-Justification and DSBV-Obligation properties that each correct process p_i is such that $view_i[r,1] = \{v\}$. It then follows from line 4, that all the correct processes assign v to their estimate est_i.

Consequently each correct process p_i DSBV-broadcasts the message PHASE$[r,2](v)$ at line 5, and, due the DSBV-value-Justification property, each correct process p_i obtains $view_i[r,2] = \{v\}$ and does not change the value of its estimate est_i, which concludes the proof of the lemma. $\square_{Lemma\ 2}$

Lemma 3. *No two correct processes decide different values.*

Proof Let r be the first round at which processes decide. If two correct processes p_i and p_j decide at line 6 of round r, we have $view_i[r,2] = \{v\}$ and $view_j[r,2] = \{w\}$. It then follows from the DSBV-Singleton property of the second DSBV-broadcast of round r that $v = w$.

Moreover, due to the DSBV-Inclusion property of the second DSBV-broadcast of round r, if a correct process p_i decides v during round r (i.e., $view_i[r,2] = \{v\}$), while another correct process p_j does not, we necessarily have $view_j[r,2] = \{v, \perp\}$, and consequently, p_j is directed to assign v to est_j (line 7).

Hence, at the beginning of round $(r+1)$, the estimates of all the correct processes are equal to v (the value decided by correct processes at round r). It then follows from Lemma 2 that they keep this value forever. It follows that, as a decided value is an estimate value, only the value v can be decided by a correct process. $\square_{Lemma\ 3}$

Lemma 4. *With probability* 1 *there is a round at the end of which all the correct processes have the same estimate value.*

Proof Let us remind that, during a round r, no Byzantine process obtains the value of the random bit before at least one correct process invoked random() at line 3 of this round. Process p_i being such a correct process, let us consider the set $view_i[r,1]$ obtained by p_i at line 2 (i.e., before it invoked random()). Let us assume that, at round r, both the value 0 and 1 are DSBV-broadcast by correct processes (otherwise, due the DSBV-Uniformity property of both DSBV-broadcast instances of round r, the round trivially satisfies the lemma). According to the value of $view_i[r,1]$, there are five possible cases, each one independent from the others. Let us also notice that this case decomposition is independent of the Byzantine processes (those cannot benefit from the knowledge of the random bit to re-order message delivery at p_i).

- Case 1: $view_i[r,1] = \{v\}$ where $v \neq \perp$. It follows from the DSBV-Inclusion property of the DSBV-broadcast of line 2 that the set $view_j[r,1]$ of any other correct process is such that $view_j[r,1] = \{v\}$ or $view_j[r,1] = \{v, \perp\}$.
- Case 2: $view_i[r,1] = \{\perp\}$. This case is similar to the previous one. It follows from the DSBV-Inclusion property of the DSBV-broadcast of line 2 that the set $view_j[r,1]$ of any other correct process is such that $view_j[r,1] = \{\perp\}$ or $view_j[r,1] = \{v, \perp\}$.
- Cases 3, 4 and 5: $view_i[r,1] = \{v, \perp\}$. In this case, it follows from the DSBV-Mutex property that it is not possible that a correct process p_j be such that $view_j[r,1] = \{\perp\}$ while another correct process p_k is such that $view_k[r,1] = \{v\}$. Hence the three following cases.
  - Case 3: $view_i[r,1] = \{v, \perp\}$, and (1) each other correct process p_j is such that $view_j[r,1] = \{\perp\}$ or $view_j[r,1] = \{v, \perp\}$, and (2) at least one correct process p_j is such that $view_j[r,1] = \{\perp\}$.

– Case 4: $view_i[r,1] = \{v, \perp\}$, and (1) each other correct process p_j is such that $view_j[r,1] = \{v\}$ or $view_j[r,1] = \{v, \perp\}$, and (2) at least one correct process p_j is such that $view_j[r,1] = \{v\}$.
– Case 5: $view_i[r,1] = \{v, \perp\}$ and all the other correct processes p_j are such that $view_j[r,1] = \{v, \perp\}$.

Let $pb(r,x)$, where $x \in \{1,2,3,4,5\}$ be the probability that the case x occurs. We have $\Sigma_x pb(r,x) = 1$. Let us now compute, for each case x, the probability $pb(x)$ that all the correct processes have the same estimate after line 4.

- Case 1. The probability that the same value s is output for $b_j[r]$ at each correct process p_j such that $view_j[r,1] = \{v, \perp\}$ is $\geq 2/d$, and the probability that this value s is equal to v is $1/2$. Hence, $pb(1) = 1/d$.

- Case 2. In this case, it follows from the DSBV-Mutex property of the DSBV-broadcast of line 2, that all the correct processes p_j execute $est_j \leftarrow b_j[r]$ at line 4. Hence, the probability that they all assign the same value to their estimates is $pb(2) = 2/d$.

- Case 3. In this case, it follows from the algorithm that all the correct processes p_j execute $est_j \leftarrow b_j[r]$ at line 4. Hence, as in Case 2, the probability that they all assign the same value to their estimates is $pb(3) = 2/d$.

- Case 4. This case is similar to Case 1, and we have $pb(4) = 1/d$.

- Case 5. This case is similar to Case 3, and we have $pb(5) = 2/d$.

As $\Sigma_x pb(r,x) = 1$, it follows from the previous case analysis that the probability p that all the correct processes terminate the first phase of round r with the same estimate value is equal to $\Sigma_x\big((pb(x)pb(r,x)\big)$, which means $p \geq 1/d$.

Let us now observe that, if all the correct processes have the same estimate value at the end of the first phase of a round r, they invoke the second DSBV-broadcast of the round with the same value. It then follows, from the DSBV-Uniformity property and line 6, that they do not change their estimate by the end of the round.

Consequently the probability $P(r)$ that all the correct processes have the same estimate at the end of a round r is $P(r) = p + (1-p)p + \cdots + (1-p)^r p = 1 - (1-p)^r$. As $\lim_{r \to +\infty} P(r) = 1$, we conclude that with probability 1, there is around after which all correct processes have the same estimate value. $\square_{Lemma\ 4}$

Lemma 5. *Let $n > 3t$. All correct processes decide with probability* 1.

Proof The lemma follows from Lemma 4 (there is a round r after which all the correct processes have the same estimate value), the DSBV-Uniformity property applied to both DSBV-broadcast instances of round r, and line 6. $\square_{Lemma\ 5}$

Theorem 4. *The algorithm described in Figure* 4 *solves the randomized binary consensus problem in $\mathcal{BAMP}_{n,t}[t < n/3, \mathrm{CC}]$.*

Proof BC-Validity follows from Lemma 1. BC-Agreement follows from Lemma 3. BC-Decision follows from Lemma 5. $\square_{Theorem\ 4}$

Theorem 5. *Let $n > 3t$. The expected decision time is constant (d rounds).*

Proof As indicated in the proof of Lemma 4, the probability that all the correct processes have the same estimate value after the first phase of a round is $\geq 1/d$. It follows that the average number of rounds so that all correct processes have the same estimate is $\leq d$. $\square_{Theorem\ 5}$

Cost of the algorithm We have the following, where $c \geq n - t$ denotes the number of correct processes. Each round requires two DSBV-broadcast instances; each of them uses two SBV-broadcast instances, and each SBV-broadcast (in which each process broadcasts a message AUX()) uses an underlying BV-broadcast instance (in which one or two messages B_VAL() are broadcast). Consequently:

- If the correct processes propose the same value, each round requires eight communication steps. Those are due to the four underlying SBV-broadcast instances, each requiring one communication step in its underlying BV-broadcast and one in its own algorithm. The total number of messages sent by correct processes is then $8\ c\ n$.
- If the correct processes propose different values, each round requires twelve communication steps: four SBV-broadcast instances, each requiring up to two communication steps in each underlying BV-broadcast and one communication step for the messages AUX()). Moreover, the total number of messages sent by the correct processes is then $12\ c\ n$ per round.
- The expected number of rounds to decide is $\leq d$ (i.e., 2 rounds when the coin is perfect).
- In addition to a round number, each message carries a single data bit plus a constant number of bits that identifies its type.
- The total number of bits exchanged by the correct processes is $O(n^2 r \log r)$ where r is the number of rounds executed by the correct processes. Hence, as the expected number of rounds is a known constant (d), the expected bit complexity is $O(n^2)$.

Remark The cost of this consensus algorithm can be reduced if one considers that the common coin is perfect and/or that the adversary cannot re-order messages. As an example, [31] assumes a perfect common coin and assumes that the adversary cannot re-order messages.

5 A Terminating Consensus Algorithm

The consensus algorithm that has been previously presented is non-terminating. While every correct process eventually decides a value, it has to execute rounds forever. So the BC-Decision property concerns decision and not halting. This section presents a modified version of the previous algorithms in which a process stops executing when it decides.

Modifying the consensus algorithm As correct processes can decide at different rounds, a process that decides and terminates while executing a round r is now required to broadcast a message TERM$[r](v)$ to inform the other processes that it decides the value v during round r and it stops its execution.

From an operational point of view, the statement "decide(v) if not yet done" at line 6 of Figure 4 is replaced by the statements "broadcast TERM$[r_i](v)$; return(v)".

Identifying DSBV, SBV, and BV broadcast instances Let us notice that each DSBV-broadcast instance is identified by a pair $[r, x]$ where r is a round number and $x \in \{1, 2\}$ is a phase number. Hence each message STAGE$[0]()$ (resp., STAGE$[1]()$) inherits this identity, and becomes STAGE$[r, x, 0]()$ (resp., STAGE$[r, x, 1]()$). Each underlying SBV-broadcast or BV-broadcast inherits also the identity $[r, x, y]$, where $y \in \{0, 1\}$. This means that the message AUX(v) used in the implementation of the $[r, x, y]$ instance of SBV-broadcast (Figure 2) becomes AUX$[r, x, y](v)$, and the message B_VAL(v) used in the $[r, x, y]$ instance of BV-broadcast (Figure 1) becomes B_VAL$[r, x, y](v)$.

Terminating SBV-broadcast Let us consider the case where a process p_j broadcasts the message TERM$[r](v)$. In order not to be blocked forever in the SBV-broadcast instance identified $[r', x, y]$, while waiting from p_j for messages AUX$[r', x, y]()$ during rounds $r' > r$ (those messages will never be sent), any process p_i considers that the message TERM$[r](v)$ sent by p_j is a compressed representation of the four infinite sequences of (the never sent) messages AUX$[r + 1, x, y](v)$, AUX$[r + 2, x, y](v)$, etc., where $(x, y) \in \{1, 2\} \times \{0, 1\}$.

So, to obtain a terminating SBV-broadcast instance identified $[r, x]$, the only modification of the algorithm presented in Figure 2 is in the waiting predicate of line 4, which is extended as follows (where the local variables $view_i$ and bin_values_i are identified by the corresponding instance identity):

$\exists$ a set $view_i[r, x, y]$ of values received in messages AUX$[r, x, y]()$ or in messages TERM$[r']()$ with $r' < r$ such that $((n - t)$ processes contributed to $view_i[r, x, y]) \wedge (view_i[r, x, y] \subseteq bin\_values_i[r, x, y])$.

Terminating BV-broadcast The same occurs at the lower level where is implemented the BV-broadcast instance used by each SBV-instance. Namely, the semantics of the message TERM$[r](v)$ is extended by considering it, for $(x, y) \in \{1, 2\} \times \{0, 1\}$, as a compressed representation of the four infinite sequences of messages B_VAL$[r + 1, x, y](v)$, B_VAL$[r + 2, x, y](v)$, etc.

let $witness_i[r, x, y](v)$ = number of different processes from which the message B_VAL$[r, x, y](v)$, or a message TERM$[r'](v)$ with $r' < r$, have been received.

operation BV_broadcast MSG$[r, x, y](v_i)$ **is**
(1) broadcast B_VAL$[r, x, y](v_i)$.

when $(witness_i[r, x, y](v) \geq t + 1) \wedge$ (B_VAL$[r, x, y](v)$ not yet broadcast) **do**
(2) broadcast B_VAL$[r, x, y](v)$.

when $(witness_i[r, x, y](v) \geq 2t + 1) \wedge (v \notin bin\_values_i[r, x, y])$ **do**
(3) $bin\_values_i[r, x, y] \leftarrow bin\_values_i[r, x, y] \cup \{v\}$.

Figure 5: Modified BV-broadcast algorithm in $\mathcal{BAMP}_{n,t}[n > 3t]$

The modified BV-broadcast algorithm is presented in Figure 5. The main modification appears in the definition of the set $witness_i[r, x, y](v)$. In addition to the processes from which p_i received messages B_VAL$[r, x, y](v)$, this set includes also the processes from which p_i received a message TERM$[r'](v)$ such that $r' < r$. Let us notice that it is possible that both p_j and p_k are counted in $witness_i[r, x, y](v)$ while they sent TERM$[r1](v)$ and TERM$[r2](v)$, respectively, with $r1 \neq r2$. To keep simple its statement, the rest of the algorithm is expressed with two guarded commands. The first guarded command states that, if not yet done, p_i broadcasts B_VAL$[r, x, y](v)$ as soon as it has received B_VAL$[r, x, y](v)$ or TERM$[r'](v)$ with $r' < r$, from at least $(t + 1)$ different processes. The second guarded command adds the value b to $bin\_values_i[r, x, y]$.

Let us observe that, according to the semantics of the messages TERM$[-]()$, a process p_i may send B_VAL$[r, x, 0](v)$ B_VAL$[r, x, 1](v)$ even if it has received only messages TERM$[-](v)$ from $(t + 1)$ different processes. Let TERM$[r^1](v)$, TERM$[r^2](v)$, ..., TERM$[r^{t+1}](v)$ be these messages. At least one of these messages is from a correct process, and consequently this process decided v. Hence, if $r_i > \max(r^1, r^2, \ldots, r^{t+1})$, p_i can broadcast the messages B_VAL$[r_i, x, 0](v)$ and B_VAL$[r_i, x, 1](v)$.

Theorem 6. *The previous modified algorithms, where a process that decides stops its execution, solve the randomized binary consensus problem in the system model $\mathcal{BAMP}_{n,t}[t < n/3, \text{CC}]$.*

Proof The proof is a direct consequence of the following observation. Considering an execution of the terminating algorithm, let us replace each message TERM$[r](v)$ by the infinite sequences of messages B_VAL $[r + 1, x, y](v)$, B_VAL $[r + 2, x, y](v)$, etc., and AUX $[r + 1, x, y](v)$, AUX $[r + 2, x, y](v)$, etc., for each $(x, y) \in \times\{0, 1\}$. This produces an execution whose message pattern could have been produced by the basic non-terminating algorithm of Figure 4. As this basic algorithm satisfies the BC-Validity and BC-Agreement consensus properties, so does its terminating version. $\square_{Theorem\ 6}$

Expedite termination It is possible to expedite termination in favorable circumstances. Those occur when a process p_i has received messages TERM$[-](v)$ (carrying possibly different round numbers but the same value v) from $(t + 1)$ different processes, which means at least one of these messages is from a correct process. Process p_i can then decide and terminate. In this case, to help other processes terminate it must also broadcast the message TERM$[r_i](v)$. The following statement can consequently be inserted in Figure 4 just after line 9 (i.e., just before p_i enters a new round).

if (TERM$[-](v)$ received from $(t + 1)$ different processes)
  then broadcast TERM$[r_i](v)$; return(v)
end if.

Content of messages TERM$()$ The reader can check that the round numbers carried by the messages TERM$()$ can be eliminated. It is sufficient for these messages to carry only a value.

6 Conclusion

Considering an asynchronous message-passing system made up of n processes, where up to t may commit Byzantine failures, this paper has first introduced a suite of simple all-to-all broadcast abstractions suited to binary values. These abstractions require $t < n/3$. The most high-level of these abstractions, denoted

DSBV-broadcast, allows each process to broadcast a binary value, and obtain a set of values such that (1) no value broadcast only by Byzantine processes can belong to the set returned by a correct process, and (2) if the set obtained by a correct process contains a single value v, then the set obtained by any correct process contains v.

Then the paper has presented a new round-based Byzantine binary consensus algorithm built on top of the DSBV-broadcast abstraction in an asynchronous system enriched with a weak common coin. In addition to being t-resilient optimal ($t < n/3$), this new Byzantine consensus algorithm, is signature-free, uses a constant number of communication steps $O(n^2)$ messages per round. Moreover, each message carries a round number plus a small constant number of bits, and the expected number of rounds to decide is constant. Finally, the algorithm tolerates message re-ordering by Byzantine processes, and does not require the coin to be perfect. Last but not least, in addition to its efficiency-related properties, a noteworthy feature of the proposed algorithm lies in its design simplicity, which is a first-class property [1].

Remark The algorithm described in [34] reduces the multivalued consensus problem to the binary consensus problem in signature-free asynchronous Byzantine systems where up to $t < n/3$ processes may be faulty. The cost of this reduction is $O(n^2)$ messages and constant time. It follows that, when used on top of the binary consensus algorithm described in the paper, we obtain a signature-free multivalued Byzantine consensus algorithm where $t < n/3$ processes may be faulty, and whose expected cost is $O(n^2)$ messages and $O(1)$ time.

7 Acknowledgments

The authors want to thank the PODC 2014 referees for their constructive comments, and the referees of the JACM submission. They also thank Shlomi Dolev (PODC 2014 PC chair), and G. Giakkoupis and J. Stainer for discussions on randomized distributed algorithms.

This work has been partially supported by the ANR French Research Council [grant number ANR-11-BS02-014] devoted to computability and complexity in distributed computing (Displexity), and the Franco-German ANR-DFG project DISCMAT devoted to connections between mathematics and distributed computing.

References

[1] [1] Aigner M. and Ziegler G., *Proofs from THE BOOK* (4th edition). Springer, 274 pages, 2010 (ISBN 978-3-642-00856-6).

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

[3] [3] Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocols. *Proc. 2nd Annual ACM Symposium on Principles of Distributed Computing(PODC'83)*, ACM Press, pp. 27-30, 1983.

[4] [4] Berman P. and Garay J.A., Randomized distributed agreement revisited. *Proc. 33rd Annual Int'l Symposium on Fault-Tolerant Computing (FTCS'93)*, IEEE Computer Press, pp. 412-419, 1993.

[5] [5] Bracha G., An asynchronous $(n-1)/3$-resilient consensus protocol. *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, ACM Press, pp. 154-162, 1984.

[6] [6] Bracha G., Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2):130-143, 1987.

[7] [7] Bracha G. and Toueg S., Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824-840, 1985.

[8] [8] Cachin Ch., Kursawe K., and Shoup V., Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*,18(3):219-246, 2005 (first version: PODC 2000).

[9] [9] Cachin Ch., Kursawe K., Petzold F., and Shoup V., Secure and efficient asynchronous broadcast protocols. *Proc. 21st Annual International Cryptology Conference CRYPTO'01*, Springer LNCS 2139, pp. 524541, 2001.

[10] [10] Cachin Ch., Guerraoui R., and Rodrigues L., *Reliable and secure distributed programming*, Springer, 367 pages, 2011 (ISBN 978-3-642-15259-7).

[11] [11] Canetti R., and Rabin T., Fast asynchronous Byzantine agreement with optimal resilience, *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC'93)*, ACM Press, pp. 42-51, 1993.

[12] [12] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996.

[13] [13] Correia M., Ferreira Neves N., and Verissimo P., From consensus to atomic broadcast: time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82-96, 2006.

[14] [14] Dwork C., Lynch N., and Stockmeyer L., Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2), 288-323, 1988.

[15] [15] Fischer M.J. and Lynch N., A lower bound for the time to ensure interactive consistency. *Information Processing Letters*, 14:183-186, 1982.

[16] [16] 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.

[17] [17] Fischer M.J. and Merritt M., Appraising two decades of distributed computing theory research. *Distributed Computing*, 16(2-3): 239-247, 2003.

[18] [18] Friedman R., Mostéfaoui A., Rajsbaum S., and Raynal M., Distributed agreement problems and their connection with error-correcting codes. *IEEE Transactions on Computers*, 56(7):865-875, 2007.

[19] [19] Friedman R., Mostéfaoui A. and Raynal M., $\diamond\mathcal{P}_{mute}$-based consensus for asynchronous Byzantine systems. *Parallel Processing Letters*, 15(1-2):162-182, 2005.

[20] [20] 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.

[21] [21] Goldwasser S., Pavlov E., and Vaikuntanathan V., Fault-tolerant distributed computing in full-information networks. *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*,IEEE Computer Society, pp. 15-26, 2006.

[22] [22] Gray J. and Reuter A., *Transaction processing: concepts and techniques*, Morgan Kaufmann Pub., San Francisco (CA), 1045 pages, 1993, (ISBN 1-55860-190-2).

[23] [23] Kihlstrom K.P., Moser L.E. and Melliar-Smith P.M., Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16-35, 2003.

[24] [24] King V. and Saia J., Breaking the $O(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. *Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC'11)*, ACM Press, pp. 420-429, 2011.

[25] [25] Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401, 1982.

[26] [26] Liang G. and Vaidya N., Error-free multi-valued consensus with Byzantine failures. *Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC'11)*, ACM Press, pp. 11-20, 2011.

[27] [27] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996 (ISBN 1-55860-384-4).

[28] [28] Lynch N.A., Merritt M., Weihl W.E., and Fekete A., *Atomic Transactions*. Morgan Kaufmann Pub., San Francisco (CA), 500 pages, 1994 (ISBN 1-55860-104-X).

[29] [29] Martin J.-Ph. and Alvisi L., Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202-215, 2006.

[30] [30] Milosevic Z., Hutle M., and Schiper A., On the reduction of atomic broadcast to consensus with Byzantine faults. *Proc. 30th IEEE Int'l Symposium on Reliable Distributed Systems (SRDS'11)*, IEEE Computer Press, pp. 235-244, 2011.

[31] [31] Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous Byzantine consensus with $t < n/3$ and $O(n^2)$ messages. *Proc. 33th ACM Symposium on Principles of Distributed Computing (PODC'14)*, ACM Press, pp. 2-9, 2014.

[32] [32] Mostéfaoui A., Rajsbaum S., and Raynal M., Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922-954, 2003.

[33] [33] 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.

[34] [34] Mostéfaoui A. and Raynal M., Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Proc. 22nd Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'15)*, Springer LNCS, 2015.

[35] [35] Patra A., Error-free multi-valued broadcast and Byzantine agreement with optimal communication complexity. *Proc. 15th Int'l Conference On Principles Of Distributed Systems (OPODIS'10)*, Springer LNCS 7109 pp. 34-49, 2011.

[36] [36] Pease M., R. Shostak R., and Lamport L., Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228-234, 1980.

[37] [37] Rabin M., Randomized Byzantine generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 116-124, 1983.

[38] [38] Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*. Morgan & Claypool, 251 pages, 2010 (ISBN 978-1-60845-293-4).

[39] [39] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[40] [40] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, 2013 (ISBN 978-3-642-32026-2).

[41] [41] Song Y.J. and van Renesse R., Bosco: one-step Byzantine asynchronous consensus. *Proc. 22th Symposium on Distributed Computing (DISC'08)*, Springer LNCS 5218, 438-450, 2008.

[42] [42] Srikanth T.K. and Toueg S., Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80-94, 1987.

[43] [43] Toueg S., Randomized Byzantine agreement. *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, ACM Press, pp. 163-178, 1984.

[44] [44] Turpin R. and Coan B.A., Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18:73-76, 1984.