Signature-Free Broadcast-Based Intrusion Tolerance: Never Decide a Byzantine Value

Achour Mostefaoui, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1953 – juin 2010

Signature-Free Broadcast-Based Intrusion Tolerance: Never Decide a Byzantine Value

Achour Mostefaoui* , Michel Raynal**
achour@irisa.fr, raynal@irisa.fr

Abstract: Provide application processes with strong agreement guarantees despite failures is a fundamental problem of fault-tolerant distributed computing. Correct processes have not to be "polluted" by the erroneous behavior of faulty processes. This paper considers the consensus agreement problem in a setting where some processes can behave arbitrarily (Byzantine behavior). In such a context it is possible that Byzantine processes collude to direct the correct processes to decide on a "bad" value (a value proposed only by faulty processes).

The paper has several contributions. It presents a family of consensus algorithms in which no bad value is ever decided by correct processes. These processes always decide a value they have proposed (and this is always the case when they all propose the same value) or a default value $\perp$. These algorithms are called *intrusion-free* consensus algorithms. To that end, each consensus algorithm is based on an appropriate underlying broadcast algorithm. One of these abstractions, called *validated broadcast* is new and allows the design of a resilience-optimal consensus algorithm (i.e., it copes with up to $t < n/3$ faulty processes where n is the total number of processes). All proposed consensus algorithms assume the underlying system is enriched with additional computational power provided by a binary Byzantine consensus algorithm. The paper presents also a resilience-optimal randomized binary consensus algorithm based on the validated broadcast abstraction. An important feature of all these algorithms lies in the fact that they are signature-free (and hence particularly efficient).

Key-words: Asynchronous message-passing system, Broadcast abstraction, Byzantine process, Consensus problem, Fault-tolerance, Intrusion-tolerance, Reliable broadcast, Resilience, Signature-free algorithm, Time-free algorithm.

Tolerance aux intrusions sans authentification

Résumé : *Ce rapport propose une famille de protocoles de décision qui assurent que la valeur de décision n'est jamais une valeur proposée par des processus malicieux. Ces protocoles sont fondés sur différentes opérations de diffusion mais sans utiliser la cryptographie.*

Mots clés : *Système à communication par messages, opération de diffusion, processus byzantin, problème de consensus, tolèrance aux fautes, intrusions, diffusion fiable, protocole sans signatures.*

---

* Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes
** Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes

1 Introduction

Asynchronous Byzantine consensus A process has a *Byzantine* behavior when it behaves arbitrarily. This bad behavior can be intentional (malicious behavior, e.g., due to intrusion) or simply the result of a transient fault that altered the local state of a process, thereby modifying its behavior in an unpredictable way. We are interested here is solving the *consensus* problem in asynchronous distributed systems prone to Byzantine process failures whatever their origin.

In a classical crash failure setting, the consensus problem is defined as follows: every process proposes a value and the non-faulty processes have to decide (termination property) on the same value (agreement property), that has to be one of the proposed values (validity). In a Byzantine failure setting, the notion of "value proposed by a faulty process" is not well-defined. Hence, the validity property is weakened and usually replaced by the following: if all non-faulty processes propose the same value, that value is decided.

Aim of the paper Unfortunately, the previous validity property leaves open the possibility for the non-faulty processes to decide an arbitrary value when all of them do not propose the same value, and such a "bad" value can "pollute" their behavior. Hence, the idea to introduce an additional validity property, that we call *non-intrusion*, to prevent this type of behavior, namely, a value proposed only by faulty processes cannot be decided by non-faulty processes. Said in another way, the non-faulty processes are required to decide the value proposed by one of them (and this has to be always the case when they all propose the same value), or a default value (denoted $\perp$) when they are not enough to propose the very same value.

The paper presents a family of asynchronous Byzantine multivalued consensus algorithms that satisfy the previous property. We call them *intrusion-free* consensus algorithms. Of course, as consensus cannot be solved in asynchronous system in which even only one process may crash [11], the underlying system has to be enriched with additional computational power in order consensus can be solved despite the net effect of asynchrony and Byzantine failures. We consider here that this additional power is given by an underlying binary Byzantine consensus algorithm (e.g., [18, 25]).

Content of the paper All the multivalued Byzantine consensus algorithms presented in the paper are signature-free (no underlying cryptography mechanism is assumed). Each algorithm relies on an appropriate underlying broadcast operation (that can be implemented despite asynchrony and up to t Byzantine processes, where t is constrained by a function on the total number n of processes). These broadcast abstractions are the classical unreliable broadcast (that requires $t < n$), the "echo" broadcast introduced in [3] (that we call *no-duplicity* broadcast), the reliable broadcast introduced in [4], plus a novel all-to-all broadcast abstraction that can be interesting by itself, that we call *validated broadcast*. All these broadcast abstractions (but unreliable broadcast) require $n > 3t$ to be implemented in an asynchronous system prone to Byzantine failures. They differ in the number of consecutive communication steps they need.

As we will see, the new validated broadcast abstraction is particularly interesting in the context of Byzantine processes. This is because it allows a correct process to deliver a message only if that message has been validated by at least one correct process. Said differently, validated broadcast eliminates the "noise" introduced by "bad" values (i.e., values proposed only by Byzantine processes).

A resulting multivalued Byzantine consensus algorithm is then characterized by its underlying broadcast algorithm that has a particular cost counted by the number of communication steps, the size of control additional information messages have to carry, and their messages. As we will see, the proposed algorithms are highly modular and exhibit a tradeoff relating their time efficiency (the weaker the underlying broadcast abstraction, the more efficient the algorithm), and the constraint on t they need (the weaker the underlying broadcast abstraction, the stronger the constraint on t).

The paper also presents a binary Byzantine consensus algorithm (which can provide the previous algorithms with the required additional computing power). This algorithm is signature-free, requires $t < n/3$ (and is consequently optimal with respect to resilience), and needs six communication steps per round.

Related work Numerous Asynchronous Byzantine algorithms have been proposed (e.g., [3, 4, 5, 12, 13, 15, 16, 22, 24] to cite a few; see also [19] for a short survey). To our knowledge, the only algorithm that considers the non-intrusion property is the one described in [9] (this algorithm requires messages to carry a vector of proposed values which, as shown here, is not necessary).

The idea to direct the processes to decide $\perp$ in "bad scenarios" (i.e., when they cannot decide a value they have proposed) is different but in the same spirit as the idea developed in the notion of *abortable* objects [1]. In that case, the "bad scenarios" are when there is concurrency among operations. In a concurrency context, operations can return $\perp$, while an operation has always to return a non-trivial result when executed in a concurrency-free context.

Road map The paper is made up of 9 sections. Section 2 presents the computation model, the different broadcast abstractions, and an algorithm implementing the validated broadcast abstraction. Section 3 presents the intrusion-free Byzantine consensus problem. Then, Sections 4 and 5 present a suite of intrusion-free multivalued Byzantine consensus algorithms that differ mainly in the underlying broadcast abstraction they use. Section 6 discusses the previous algorithms. Section 7 presents a randomized binary consensus algorithm based on the validated broadcast abstraction. Finally, Section 9 concludes the paper.

2 Computation system model

2.1 Base model

Asynchronous processes The system is made up of a finite set of $n > 1$ processes denoted $p_1, \ldots, p_n$ that communicate by exchanging messages through a communication network. Each process proceeds to its own speed, which means that processes are asynchronous.

Multiset All algorithms presented in the paper use multisets. A *multiset* (sometimes also called *bag*) differs from a set in that it can contain several copies of the same value. Given a multiset rec_i, $\#_v(rec_i)$ denotes the occurrence number of v in rec_i.

Failure model Up to t processes can exhibit a *Byzantine* behavior. A Byzantine process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transition, etc. Moreover, Byzantine processes can collude to "pollute" the computation. Yet, it is assumed that they do not control the network. This means that they cannot corrupt the messages sent by non-Byzantine processes, and the schedule of message delivery is uncorrelated to Byzantine behavior. A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *correct* or *non-faulty*. Given an execution, $\mathcal{C}$ denotes the set of processes that are correct in that execution.

Notation This process model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$. In the following, this model is enriched with a constraint on t and a specific broadcast abstraction. As an example, $\mathcal{BZ}\_\mathcal{AS}_{n,t}[n > 5t, \mathsf{WB}]$ is $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ in which less than $n/5$ processes are faulty and processes communicate using the operations of the WB broadcast abstraction (see below).

2.2 Asynchronous communication network

Base communication network Each pair of processes is connected by a channel (which means that when a process receives a message, it knows which is the sender of the message). Each channel is asynchronous (no bound on message transfer delay, except it is finite), and reliable (no loss, creation or corruption of messages). Hence, the network is asynchronous.

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()".

In the following, several types of broadcast operations are defined. They all can be implemented from the base send and receive primitives, which means that, while they provide us with distinct communication abstraction levels, they do not provide the processes with additional computing power.

When considering the broadcast abstraction XX (where XX stands for WB, NDB, VB or RB, see below), we say that a process "XX-broadcasts" or "XX-delivers" a message.

Unreliable broadcast The first pair of operations denoted WB_broadcast() and WB_deliver() are used to denote a simple unreliable broadcast. WB_broadcast TAG(m) is used as a shortcut for

for each $j \in \{1, \ldots, n\}$ send TAG(m) to p_j **end for**,

and WB_deliver() is synonym with receive(). This means that a message broadcast by a correct process is delivered to all processes. Differently, while it is assumed to send the same message to all processes, a faulty process can actually send different messages to distinct processes and no message to others.

Trivially, an invocation of WB_broadcast TAG(m) costs one communication step and $O(n)$ messages (more precisely, $n - 1$ messages). This communication abstraction is called WB, and the corresponding system model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\mathsf{WB}]$.

Remark When measuring the cost of a broadcast abstraction we do not take into account the size of the "data message" that is broadcast. This is because this size is independent of the way the broadcast is implemented. We only consider the size of the additional control information required by the corresponding broadcast implementation.

No-duplicity broadcast This communication abstraction, denoted NDB, is defined by the operations NDB_broadcast() and NDB_deliver() that provide the processes with a higher abstraction level than WB. Considering an instance where NDB_broadcast() is invoked by process p_i, this broadcast abstraction is defined by the following properties.

- NDB-No-duplicity. No two correct processes NDB-deliver distinct messages from p_i.
- NDB-Termination. If the sender is correct, all correct processes eventually NDB-deliver its message.

The corresponding system model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\mathsf{NDB}]$. Let us observe that, if the sender p_i is faulty, it is possible that some correct processes deliver a message from p_i while others do not. The no duplicity property prevents correct processes from delivering different messages from a faulty sender. (When considering the less severe crash failure model, no-duplicity broadcast and weak broadcast are equivalent.)

This broadcast primitive has been defined by Toueg [24]. It can be built on top of the base send/receive primitives in systems where $t < n/3$. Such an 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). The size of the control information added to a message is $\log_2 n$ (sender identity).

Reliable broadcast The reliable broadcast abstraction, denoted RB, has been proposed by Bracha [4]. Strictly stronger than the no-duplicity broadcast, it provides processes with the operations RB_broadcast() and RB_deliver() defined by the following properties.

- RB-No-duplicity. No two correct processes RB-deliver distinct messages from p_i.
- RB-Termination. If the sender is correct, all correct processes eventually RB-deliver its message.
- RB-Uniformity. If a correct process RB-delivers a message from p_i (possibly faulty) then all correct processes eventually RB-deliver a message from p_i.

It has been proved in [4] that $n > 3t$ is a necessary requirement to implement this operation. If the sender is correct, only three communication steps and $O(n^2)$ messages whose size is $O(\log_2 n)$ bits are necessary.

Validated broadcast This last communication abstraction, denoted VB, is defined by the operations VB_broadcast() and VB_deliver() described below. It is a new abstraction that provides the processes with a communication level higher than no-duplicity broadcast. More precisely, validated broadcast is an *all-to-all* reliable broadcast with a notion of message *validation*, namely, a message has to be validated by enough processes in order to be VB-delivered, otherwise the default value $\perp$ is VB-delivered instead of it.

As it is an all-to-all broadcast, VB requires that all correct processes invoke VB_broadcast(). The idea is that a value v is valid if there is at least one correct process that broadcasts that value. As no process knows if it is itself correct or faulty (e.g., a process can correctly execute its algorithm and then crash), a value broadcast by a process is required to be validated by $n - 2t \geq t + 1$ processes to be valid. As already indicated, if a message value is not validated, $\perp$ is delivered instead of it.

More precisely, assuming a broadcast instance in which every correct process invokes VB_broadcast(), let us consider the invocation of a particular process p_i that invokes VB_broadcast(m). VB is defined by the following properties.

- VB-No-duplicity. No two correct processes VB-deliver distinct messages from p_i (the message that is VB-delivered can be a non-$\perp$ value or the default value $\perp$).
- VB-Termination. If the sender is correct and VB-broadcast m, all correct processes eventually VB-deliver the same message m' where m' is m or $\perp$.
- VB-Uniformity. If a correct process VB-delivers a message from p_i (possibly faulty), all correct processes eventually VB-deliver a message from p_i.
- VB-Validation. If $\perp$ is VB-delivered, there is at least one correct process that does not validate the message VB-broadcast by p_i. If $m' \neq \perp$ is VB-delivered, m' has been validated by at least one correct process.

broadcast	x-to-y	# comm. steps	message size	# msgs	constraint on t
WB	1-to-n	1	constant	$n-1$	$n > t$
NDB	1-to-n	2	$\log_2 n$	$0(n^2)$	$n > 3t$
RB	1-to-n	3	$\log_2 n$	$0(n^2)$	$n > 3t$
VB	n-to-n	6	$\log_2 n$	$n \times 0(n^2)$	$n > 3t$

Table 1: Cost and constraint of the different broadcast abstractions

Comparing the broadcast abstractions Table 1 compares the costs of the three previous broadcast abstractions. Considering one broadcast instance, the second column indicates the broadcast type (1-to-n or n-to-n). The third column indicates the number of (sequential) communication steps that are needed. The fourth column presents the size of additional control information that an implementation message has to carry (the $\log_2 n$ comes from the fact that the identity of the process that broadcasts a message has to be sent together with it when forwarded by another process). The fifth column indicates the number of implementation messages that are needed. Finally, the last column states the constraint on t required to implement the corresponding abstraction in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$.

2.3 An implementation of the validated broadcast abstraction

Algorithm 1 implements the all-to-all validated broadcast. Let us recall that all-to-all means here that all correct processes are assumed to invoke VB_broadcast(). This means that a process VB-delivers at least $n - t$ messages. This implementation uses two consecutive RB-broadcast invocations. Its cost is consequently, $2 \times 3 = 6$ communication steps and $O(n^3)$ messages of size $O(\log_2 n)$ bits.

The implementation of a VB-broadcast instance is made up of two parts.

- The first part is made up of two consecutive RB-broadcasts. More precisely, a process p_i first invokes RB_broadcast INIT(v_i) and waits until it has RB-delivered messages from at least $n - t$ processes (lines 01-03). The values RB-delivered are deposited in a multiset denoted rec_i.

  Then, if value v_i has been RB-delivered from at least $n - 2t \geq t + 1$ processes (which means that it has been RB-broadcast by at least one correct process), p_i validates it by assigning yes to aux_i. Otherwise p_i sets aux_i to no at line 04 (in that case it does not validate v_i). Then, p_i issues a second RB-broadcast (line 05) to disseminate aux_i (that is equal to yes or no) to all processes.

```
operation VB_broadcast(v_i)
(01)  RB_broadcast INIT(v);
(02)  let rec_i = multiset of values RB_delivered to p_i;
(03)  wait until (|rec_i| ≥ n − t);
(04)  if (#_{v_i}(rec_i) ≥ n − 2t) then aux_i ← yes else aux_i ← no end if;
(05)  RB_broadcast VALID(aux_i).

for 1 ≤ j ≤ n VB-delivery task T_i[j]:
(06)  wait until (VALID(x) and INIT(v) are RB_delivered from p_j);
(07)  if (x = yes) then wait until (#_v(rec_i) ≥ n − 2t); d = v;
(08)                else wait until (#_{v'≠v}(rec_i) ≥ t + 1); d = ⊥;
(09)  end if;
(10)  VB_deliver(d) at p_i as the value VB-broadcast by p_j.
```

Algorithm 1: A reliable-broadcast-based implementation of VB-broadcast

- The second part is made up of n tasks. The task $T_i[j]$ starts by the wait statement for both the value v RB-broadcast by p_j and the boolean x RB-broadcast by p_j to say whether its value v has been validated or not. Note that the value v can be delivered either at line 03 or at line 06. (Let us remind that each time a message INIT(v) is RB-delivered to p_i, the value v is added to rec_i, which means that, after the predicate $|rec_i| \geq n - t$ has become true at line 03, the set rec_i still keeps on being updated when new messages INIT() are RB-delivered to p_i.)

  If $x = yes$, as p_j can be Byzantine, v has not necessarily been validated. Hence, p_i has to check it. To that end, p_i waits until the predicate $\#_v(rec_i) \geq n - 2t$ becomes true (line 07). When this predicate $\#_v(rec_i) \geq n - 2t \geq t + 1$ becomes true (if ever it does, line 07) we have $\#_v(rec_i) \geq t + 1$ and, consequently, v is VB-delivered to p_i as being the value VB-broadcast by p_j.

  Differently, if $x = no$, p_i waits until rec_i contains more than t values different from v (the value RB-delivered from p_j). When this occurs (if ever it does, line 07) p_i VB-delivers $\perp$ as the value VB-broadcast by p_j.

Theorem 1. *Algorithm 1 implements the validated broadcast abstraction in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathsf{RB}]$.*

Proof Proof of the VB-no-duplicity property (no two correct processes VB-deliver distinct messages from a given process p_i). This property is a direct consequence of the RB-no-duplicity property of the underlying RB-broadcast abstraction used to send a message VALID() at line 05 (which states that no two correct processes RB-deliver different values from the same process).

Proof of the VB-termination property (if p_i is correct and VB-broadcast m, all correct processes eventually VB-deliver the same message m' from p_i where m' is m or $\perp$). As there are at least $n - t$ correct processes, and each correct process VB-broadcasts a value, we eventually have $|rec_j| \geq n - t$ at any correct process p_j. Hence, no correct process blocks forever at line 03 and RB-broadcasts a message VALID() at line 05. We now consider two cases.

- Let p_i be a correct process that RB-broadcast VALID(yes). It follows from line 07 that (a) $d = v_i$ (the value VB-broadcast by p_i), and (b) rec_i contains at least $n - 2t$ copies of $v = v_i$, i.e., p_i has RB-delivered $n - 2t$ messages INIT (v). Due to the RB-uniformity of RB-broadcast, any correct process p_j eventually RB-delivers these $n - 2t$ messages INIT (v) and VALID(yes), from which follows that p_j VB-delivers $v = v_i$ at line 10.

- Let p_i be a correct process that RB-broadcasts VALID(no). It follows from the RB-termination property that any correct process p_j RB-delivers no from p_i. Moreover, from the test line 04, if p_i RB-broadcast VALID(no) then among the $n-t$ values in rec_i less then $n - 2t$ values are equal to v_i i.e. more than t values are different from v_i. Hence due to the RB-uniformity of RB-broadcast, any correct process p_j eventually RB-delivers at least $t + 1$ values different from v_i and consequently VB-delivers $\perp$ at line 10.

Proof of the VB-uniformity property (if a correct process p_i VB-delivers a message from p_j -possibly faulty-, all correct processes eventually VB-deliver a message from p_j). Let p_i be a correct process that VB-delivers a value d from p_j. This means that p_i has previously RB-delivered a message INIT(v) and a message VALID(x) from p_j at the latest in its delivery task $T_i[j]$. The proof of this property is very similar to the previous one.

It follows from the predicate at line 06 that p_i has RB-delivered (1) two messages VALID(x) and INIT(v) from p_j and (2) a set rec_i of values that satisfies some property (depending on the value of x). Due to the RB-uniformity property, any correct process p_j eventually RB-delivers the messages VALID(x) and INIT(v) RB-delivered by p_i and also a set $rec_j = rec_i$ of values. It follows that p_j eventually RB-delivers the same value as p_i. from p_j.

Proof of the VB-validation property (if $\perp$ is VB-delivered from p_i there is at least one correct process that does not validate the message VB-broadcast by p_i. If $v \neq \perp$ is VB-delivered, v has been validated by at least one correct process).

If a value $v \neq \perp$ is VB-delivered by a correct process p_i as the value VB-broadcast by p_j, this value appears at least $n - 2t \geq t + 1$ times in rec_i which means that at least one correct process has VB-broadcast v. Similarly, if $\perp$ is VB-delivered, at least $t + 1$ processes have VB-broadcast values that differ from the value VB-broadcast by p_j. $\square_{Theorem\ 1}$

3 Intrusion-tolerant Byzantine consensus and the enriched model

3.1 Byzantine consensus

Byzantine consensus The consensus problem has been informally stated in the Introduction. Assuming that at least each correct process proposes a value, each of them has to decide on a value in such a way that the following properties are satisfied.

- C-Termination. Every correct process eventually decides on a value.
- C-Agreement. No two correct processes decide on different values.
- C-Obligation (validity). If all correct processes propose the same value v, then v is decided.

Intrusion-tolerant Byzantine (ITB) consensus In Byzantine consensus, if not all correct processes propose the same value, any value can be decided. As indicated in the Introduction, we are interested in a stronger version of the consensus problem in which a value proposed only by faulty processes can never be decided. This consensus problem instance is defined by the termination, agreement and obligation properties stated above plus the following validity property.

- C-Non-intrusion (validity). A decided value is a value proposed by a correct process or $\perp$.

The fact that no value proposed only by faulty processes can be decided gives its name (namely *intrusion-tolerant*) to that consensus problem instance.

Binary consensus The consensus is *binary* when only two values (e.g., 0 and 1) can be proposed. When more than two values can be proposed, consensus is *multivalued*.

Interestingly, the fact that only two values can be proposed to a binary Byzantine consensus algorithm provides it with an interesting property, namely, if all correct processes propose the same value $b \in \{0, 1\}$, it follows from the obligation property that they decide b, whatever the value (b or $\overline{b} = 1 - b$) proposed by the faulty processes. Hence, we have the following property (that is no longer true for multivalued consensus).

Property 1. *Any binary Byzantine consensus algorithm that satisfies the obligation property, satisfies also the non-intrusion property. Moreover, $\perp$ is never decided.*

3.2 Enriched model for multivalued ITB consensus

Additional power is required It is well-known that Byzantine consensus cannot be solved when $t \leq n/3$ in synchronous systems [14, 20]. Moreover, consensus cannot be solved in asynchronous systems as soon as even only one process may crash [11], which means that Byzantine consensus cannot be solved either as soon as one process can be faulty. Said another way, additional computational power is needed if one wants to solve Byzantine consensus in an asynchronous system.

Such an additional power can be obtained by randomization (e.g., [3, 9, 13, 21, 24]), failure detectors (e.g., [13, 15]), additional synchrony assumptions (e.g., [10, 16]), or even the assumption that there is a binary consensus algorithm that is given for free by the underlying system (e.g., [6, 9, 18, 22, 25]).

Enriched model for multivalued ITB consensus In the following, BBC denotes any algorithm that solves the binary Byzantine consensus problem. (Such algorithms are described in [4, 9, 13, 24]. See also Section 7). Let $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\mathsf{XX}, BBC]$ denote the system model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ enriched with BBC (computational power) and the broadcast abstraction XX.

As announced in the Introduction, the aim is to design a generic multivalued ITB consensus algorithm on top of $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\mathsf{XX}, BBC]$.

4 A generic consensus algorithm based on the WB or NDB abstraction

This section presents a generic multivalued ITB consensus algorithm that can be instantiated with WB or NDB. It uses two rounds for each process to compute a value it proposes to the underlying binary consensus. The instantiation with WB requires $n > 5t$, while the one with NDB requires $n > 4t$.

Principles and description of the algorithm In Algorithm 2, a process invokes propose(v_i) where v_i is the value it proposes to the consensus. It terminates when it executes the return() statement (line 14) that supplies it with the decided value. (In order to prevent confusion, the operation of the underlying binary consensus that is built is denoted bin_propose().)

In order to reduce the Byzantine consensus problem to its binary counterpart to benefit from BBC, the processes first exchange the values they propose. If a process sees that a value v has been proposed "enough" times, it proposes 1 to BBC, otherwise it proposes 0. Then, if 1 is decided from BBC, the correct processes decide the value v that has been proposed "enough" times, otherwise they decide $\perp$ (lines 09-14). For this to work, two things are necessary:

- (a) A value has to appear as if it has been proposed by enough processes.
- (b) If a process p_i proposes 1 to BBC because it has seen enough copies of a value v, it must be sure that any other correct process p_j will be able to decide v even if it has proposed 0 to BBC (because it has not seen enough copies of v).

```
operation propose(v_i)
(01) XX_broadcast EST1(v_i);
(02) wait until (EST1(−) messages XX_delivered from (n − t) processes);
(03) let rec1_i = multiset of values XX_delivered and carried by EST1 messages;
(04) if (∃v : #_v(rec1_i) ≥ n − 2t) then aux_i ← v else aux_i ← ⊥ end if;
(05) XX_broadcast EST2(aux_i);
(06) wait until (EST2(−) messages XX_delivered from (n − t) processes);
(07) let rec2_i = multiset of values XX_delivered and carried by EST2 messages;
(08) if (∃v ≠ ⊥ : #_v(rec2_i) ≥ n − 2t) then bp_i ← 1 else bp_i ← 0 end if;
(09) if (∃v ≠ ⊥ : v ∈ rec2_i) then let v = most frequent non-⊥ value in rec2_i;
(10)                                 res_i ← v
(11)                            else res_i ← ⊥
(12) end if;
(13) b_dec_i ← bin_propose(bp_i);      % underlying binary consensus %
(14) if (b_dec_i = 1) then return(res_i) else return(⊥) end if.
```

Algorithm 2: A generic intrusion-tolerant Byzantine consensus algorithm

These two issues are solved by two asynchronous rounds executed before invoking the underlying BBC algorithm (lines 01-12). The messages of the first round and the second round are tagged EST1 and EST2, respectively. Interestingly, we will state below two properties $PR1$ and $PR2$ that are the same as the properties used in [17] to solve consensus on top of an asynchronous system enriched with any of Chandra and Toueg's failure detectors [8].

It is important to remark that, at the abstraction level of the consensus algorithm, a message carries only a tag (EST1 or EST2) and a proposed value or $\perp$. Hence, considering that proposed values have constant size, the size of the messages used by the algorithm is $O(1)$ (no message is required to carry array-like data structures whose size would depend on n).

First round The aim of this round (lines 01-04) is to direct each process p_i to define a "new" proposed value aux_i in such a way that the values aux_i of the correct processes satisfy the following property (Lemma 1):

$$PR1 \equiv \big[\forall i, j \in \mathcal{C} : \big((aux_i \neq \perp) \wedge (aux_j \neq \perp)\big) \Rightarrow (aux_i = aux_j = v) \wedge (v \text{ has been proposed by a correct process})\big].$$

Hence this round replaces (for the correct processes) the set of values they propose by a non-empty set including at most two values (namely, a value v proposed by a correct process and $\perp$).

From an operational point of view, this is obtained as follows. The processes first exchange (with the help of the underlying broadcast facility) the values they propose (lines 01-02). The values delivered at p_i are kept in the multiset $rec1_i$. Then, if there is a value v in $rec1_i$ such that $\#_v(rec1_i) \geq n - 2t$, v is assigned to aux_i. Otherwise $aux_i = \perp$.

Second round The aim of the second round (lines 05-12) is to establish the following property denoted $PR2$ (Lemma 2) in order the result of the underlying BBC algorithm can be safely exploited as described previously (lines 13-14). The local variable bp_i contains the value proposed by p_i to the underlying BBC algorithm, and res_j contains the non-$\perp$ value that any correct process p_j will decide if $\perp$ is not decided.

$$PR2 \equiv \big[(\exists i \in \mathcal{C} : bp_i = 1) \Rightarrow (\forall j \in \mathcal{C} : res_j = res_i = v \neq \perp)\big].$$

Operationally, this is obtained as follows. With the help of the underlying broadcast abstraction the correct processes exchange the values of their aux_i variables. The values delivered at p_i are saved in the multiset $rec2_i$. (This multiset contains $n - t$ values, and, due to $PR1$, those can be $\perp$, a non-$\perp$ value v proposed by a correct process, and at most t arbitrary values sent by faulty processes.)

If there is a non-$\perp$ value v such that $\#_v(rec2_i) \geq n - 2t$, p_i proposes $bp_i = 1$ to the binary consensus. Otherwise, p_i has not seen enough copies of a value $v \neq \perp$ and consequently proposes $bp_i = 0$. In all cases, p_i defines res_i as the most frequent non-$\perp$ value it has received. As the proof of Lemma 2 will show, if a correct process p_i invokes $\mathsf{bin\_propose}(1)$, all correct processes will have the same non-$\perp$ value in their res_j variables.

4.1 Proof of the algorithm

Let us recall that $\mathcal{C}$ denotes the set of processes that are correct in the considered execution.

Lemma 1. *$PR1$ holds in both system models $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathsf{WB}]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathsf{NDB}]$.*

Proof Let p_i and p_j be two correct processes such that $aux_i = v \neq \perp$. We consider separately each case stated in the lemma assumption.

- Case 1: $t < n/5$ and the correct processes use the broadcast abstraction WB.

  As $aux_i = v \neq \perp$, it follows that $\#_v(rec1_i) \geq n - 2t$ (line 04). Hence, due to the WB abstraction, among the $n - t$ messages it has WB-delivered (from different processes), at least $n - 2t$ are $\text{EST1}(v)$. As at most t processes are faulty, it follows that at least $n - 3t$ correct processes have broadcast a message $\text{EST1}(v)$. It then follows from $n > 5t$ that, among the $n - t$ messages it has WB-delivered (from different processes), p_j has WB-delivered at least $n - 4t$ messages $\text{EST1}(v)$ from correct processes, i.e., p_j has WB-delivered at least $t + 1$ messages $\text{EST1}(v)$. On another side, as p_j assigns w to aux_j, it has WB-delivered $n - 2t$ messages $\text{EST1}(w)$ (from different processes).

  Hence, among the $n-t$ messages that p_j has WB-delivered, (i): $n-2t$ are from different processes and carry w, and (ii): at least $t + 1$ are from different correct processes and carry v. As $(n - 2t) + (t + 1) > n - t$, it follows that there is a correct process p_x that WB-broadcast $\text{EST1}(w)$ and $\text{EST1}(v)$, but as it is correct it WB-broadcast a single message and we consequently have $v = w$.

  Finally, the proof that v has been proposed by a correct process follows from the observation that v has been sent by at least $n - 3t > t$ correct processes.

- Case 2: $t < n/4$ and the correct processes use the broadcast abstraction NDB.

  In that case, p_i has NDB-delivered at least $n - 2t$ messages $\text{EST1}(v)$, from different processes, and p_j has NDB-delivered at least $n-2t$ messages $\text{EST1}(w)$ from different processes. As $n > 4t$, it follows that there is a process p_x such that p_i has NDB-delivered $\text{EST1}(v)$ from p_x and p_j has NDB-delivered $\text{EST1}(w)$ from p_x. If p_x is correct, this is impossible (NDB-termination property). If p_x is faulty, this is also impossible due to the NDB-no-duplicity property (if a correct process NDB-delivers a value from a faulty process p_x, any other correct process either NDB-delivers the same value from p_x or does not NDB-deliver a message from p_x). It follows that we have $v = w$.

  Finally, similarly to the previous case, the proof that v has been proposed by a correct process follows from the observation that v has been NDB-broadcast by at least $n - 2t > t$ correct processes.

$\square_{Lemma\ 1}$

Lemma 2. *$PR2$ holds in both system models $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathsf{WB}]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathsf{NDB}]$.*

Proof Let p_i be a process such that $bp_i = 1$. It follows from lines 06-08 that the multiset $rec2_i$ contains $n - t$ values (including $\perp$). From line 08 we also have $(bp_i = 1) \Rightarrow \big(\exists v \neq \perp : \#_v(rec2_i) \geq n - 2t\big)$, from which we conclude that p_i has delivered at least $n - 2t$ messages $\text{EST2}(v)$. Moreover, due to Lemma 1, the values sent by correct processes are only v or $\perp$. Let us consider separately each case stated in the lemma assumption.

- Case 1: $t < n/5$ and the correct processes use the broadcast abstraction WB.

  As there are at most t faulty processes, at most t messages $\text{EST2}(v)$ WB-delivered by p_i are from faulty processes. Consequently, at least $n-3t$ correct processes have WB-broadcast $\text{EST2}(v)$ to p_j. As p_j waits for $n-t$ messages, it can miss at most t messages

EST2(v) from correct processes (this is because, in the worst case, the t messages missed by p_j are from correct processes that WB-broadcast EST2(v)). Consequently, p_j WB-delivers at least $n-4t$ messages EST2(v) from correct processes. As $n > 5t$, we have $\#_v(rec2_j) \geq n-4t > t$. Let us finally notice that, as at most t processes are faulty, p_j WB-delivers at most t messages EST2$(-)$ carrying values different from v and $\perp$. Hence, $\forall w \neq \perp$ we have $\#_v(rec2_j) > t \geq \#_w(rec2_j)$, which proves the lemma.

- Case 2: $t < n/4$ and the correct processes use the broadcast abstraction NDB.

  In that case, due to the NDB broadcast, no two correct processes can NDB-deliver different values from the same faulty process. The worst case is then when (a) t processes are faulty and NDB-broadcast the same value $w \notin \{v, \perp\}$, and (b) p_j NDB-delivers these t messages EST2(w). We trivially have $t \geq \#_w(rec2_j)$. On another side, as $\#_v(rec2_i) \geq n-2t > 2t$, and p_j misses at most t messages EST2(v), we have $\#_v(rec2_j) > t$. Hence, we have $\#_v(rec2_j) > t \geq \#_w(rec2_j)$, which concludes the proof of the lemma.

$\square_{Lemma\ 2}$

Theorem 2. *Algorithm 2 solves the multivalued consensus problem (as defined by the C-termination, C-agreement, C-obligation and C-non-intrusion properties) in both $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathsf{WB}, BBC]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathsf{NDB}, BBC]$.*

Proof Proof of the C-termination property (every correct process decides). As at most t processes are faulty, no correct process blocks forever at line 02 or line 06. Finally, due to the termination property of the underlying binary consensus algorithm BBC, every correct process decides.

Proof of the C-agreement property (no two correct processes decide differently). If BBC returns 0, all correct processes decide $\perp$, and C-agreement trivially follows. Hence, let us consider that BBC returns 1. It then follows from Property 1 of the underlying BBC that there is a correct process p_i such that $bp_i = 1$. Hence, due to Lemma 2, any correct process p_j is such that $res_j = v$, and all correct processes decide v.

Proof of the C-obligation property (if all correct processes propose the same value, that value is decided). Let us assume that all correct processes propose value v. Let p_i be any correct process. We then have $\#_v(rec1_i) \geq n-2t$ at each correct process p_i, and consequently each of the (at least) $n-t$ correct process sends EST2(v) (line 05). So, each correct process delivers at least $n-2t$ of these messages and we have $\#_v(rec2_i) \geq n-2t$. Hence, any correct p_i is such that $bp_i = 1$ and sets res_i to v. Due to the obligation property of the underlying binary consensus, value 1 is decided, and consequently all correct processes decide v.

Proof of the C-non-intrusion property (a non-$\perp$ value proposed only by faulty processes cannot be decided). If a non-$\perp$ value is decided, it follows from Property 1 of the underlying BBC that a correct process p_i has proposed 1. Hence, we have $bp_i = 1$, and consequently $\#_v(rec2_i) \geq n-2t$. As there are at most t faulty processes, it follows that correct processes have broadcast EST2(v), which in turn implies that $n-2t$ processes have broadcast EST1(v), i.e., at least $n-3t$ processes have broadcast EST1(v), from which we finally conclude that v has been proposed by correct processes. $\square_{Theorem\ 2}$

5 A consensus algorithm based on the validated broadcast abstraction

This section presents an intrusion-free Byzantine consensus algorithm based on validated broadcast. This algorithm requires $t < n/3$ and has consequently an optimal resilience. It requires a single round (instead of 2 as in Figure 2) but, as it uses a validated broadcast, this round requires four communication steps.

In Algorithm 3, after having VB-broadcast its value, a process p_i waits for EST$()$ messages from $n-t$ processes and deposits the corresponding values in the multiset rec_i. Then, p_i checks if (in addition to $\perp$) it has VB-delivered exactly one non-$\perp$ value v and that value has been VB-broadcast by at least $n-2t$ processes (line 04). If there is such a value, p_i proposes 1 to the underlying binary consensus, otherwise it proposes 0 (line 05).

Finally, p_i decides $\perp$ if the underlying binary consensus returns 0 (lines 08 and 11). Differently, if 1 is returned, p_i waits until it has VB-delivered $n-2t$ EST$()$ carrying the very same value v (line 09) and then decides that value (line 10). Let us notice that, among these $n-2t$ messages, some have been already VB-delivered at line 02. The important point is (as shown in the proof) that the net effect of (a) the validated broadcast, (b) the predicate used at line 04, and (c) the predicate used in the wait statement at line 09, ensures that if a correct process invokes bin_propose(1), then all correct processes eventually VB-deliver $n-2t$ times the very same value v and decide it.

Theorem 3. *Algorithm 3 solves the multivalued consensus problem (defined by the C-termination, C-agreement, C-obligation and C-non-intrusion properties) in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathsf{VB}, BBC]$.*

```
operation propose(v_i)
(01)  VB_broadcast EST1(v_i);
(02)  wait until (EST(−) messages VB_delivered from (n − t) processes);
(03)  let rec_i = multiset of the values v such that EST(v) is VB_delivered to p_i;
(04)  if (∃v ≠ ⊥ : #_v(rec_i) ≥ n − 2t) ∧ (rec2_i contains a single non-⊥ value)
(05)    then bp_i ← 1 else bp_i ← 0
(06)  end if;
(07)  b_dec_i ← bin_propose(bp_i);      % underlying binary consensus %
(08)  if (b_dec_i = 1)
(09)    then wait until (∃v ≠ ⊥ such that EST(v) VB_delivered from (n − 2t) processes);
(10)         return(v)
(11)    else return(⊥)
(12)  end if.
```

Algorithm 3: A validated-broadcast-based intrusion-tolerant Byzantine consensus algorithm

Proof Proof of the C-termination property (every correct process decides). If the underlying binary consensus returns 0, termination is trivial. Hence, let us consider that 1 is returned. Due to Property 1, there is a correct process p_i such that $bp_i = 1$, which in turn implies that, at line 02, p_i has received at least $n - 2t$ messages $\text{EST}(v)$. Due to the VB-no-duplicity and VB-uniformity properties of the VB abstraction, any correct process eventually VB-delivers these $n - 2t$ messages $\text{EST}(v)$. Hence, no correct process p_j blocks forever at line 09, which concludes the proof of the termination property.

Proof of the C-agreement property (no two correct processes decide differently). The proof is similar to the previous one. If the underlying binary consensus returns 0, agreement is trivial. If 1 is returned, it follows from $n - 2t > t$, that the value v the processes are waiting for at line 09 is unique, which completes the proof of the agreement property.

Proof of the C-obligation property (if all correct processes propose the same value, that value is decided). If all correct processes propose the same value v, it follows from the VB-validation property that v is necessarily validated, and from the other properties that they all VB-deliver at least $n - 2t$ messages $\text{EST}(v)$. Moreover, as $n - 2t > t$, v is unique. Due to VB-validation property, a value VB-broadcast only by faulty processes cannot be validated and consequently no correct process can VB-deliver it. This means that only v, $\perp$ or nothing at all can be VB-delivered from a faulty process. It follows that, at each correct process p_i, the predicate of line 04 is satisfied and p_i proposes $bp_i = 1$. Due to the C-Obligation property of the binary consensus, they all decide 1 and consequently decide the same proposed value v.

Proof of the C-non-intrusion property (a non-$\perp$ value proposed only by faulty processes cannot be decided). If a value w is proposed only by faulty processes, due to the VB-validation property, no correct process p_i VB-delivers it. If the binary consensus returns 0, w is not decided. If binary consensus returns 1, we have seen in the proof of the C-agreement property that the processes decide a value v such that at least $n - 2t$ messages $\text{EST}(v)$ have been VB-delivered, from which we conclude that w cannot be decided. $\square_{Theorem\ 3}$

6 Discussion

An interesting property of the previous ITB consensus algorithms Let v be the most proposed value (it is possible that several values are equally most proposed, in that case any of them is chosen), and let $\#_v$ be the number of processes that propose it. The previous algorithms have the following noteworthy property. (This follows from Lemma 1, Lemma 2 and Theorem 2 for the instances obtained from the generic Algorithm 2, and from Theorem 3 for Algorithm 3 based on a validated broadcast.)

- If $\#_v \geq n - t$, then v is always decided by the correct processes (let us observe that, in that case, there is a single most proposed value).
- If $\#_v < n - 2t$, then $\perp$ is always decided by the correct processes.
- If $n - 2t \leq \#_v < n - t$, then which value (v or $\perp$) is decided by the correct processes depends on the behavior of the Byzantine processes.

Let us consider an omniscient observer that would know which are the proposed values In the first and the second cases, this omniscient observer can compute the result in a deterministic way. Differently, in the last case it cannot. The value that is decided depends actually on the behavior of Byzantine processes (that can favor the most proposed value, or entail a $\perp$ decision). These different possibilities are depicted on Figure 1. Of course, a value proposed only by Byzantine processes is necessarily proposed by less then $n - 2t$ processes as $n > 3t$ and hence cannot be the decision value.

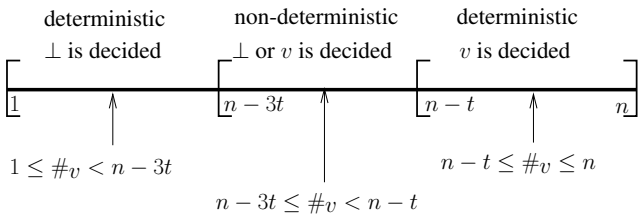

Figure 1: Deterministic vs non-deterministic scenarios

Comparing the previous signature-free multivalued ITB algorithms Table 2 presents a summary of the cost and the constraint on t associated with the previous signature-free multivalued ITB consensus algorithms. As they can all use the same underlying BCC algorithm, the comparison does not take it into account.

It is easy to see that, due the weaker constraint on t, Algorithm 3 instantiated with VB outperforms Algorithm 1 instantiated with NDB. On another side, in a system where the number of Byzantine processes remains small, Algorithm 1 instantiated with WB is the most efficient.

Consensus algorithm instantiated with	# communication steps	message size at send/receive level	# msgs at send/receive level	constraint on t
Algorithm 1 with WB	$2 \times 1 = 2$	constant	$O(n^2)$	$n > 5t$
Algorithm 1 with NDB	$2 \times 2 = 4$	$\log_2 n$	$O(n^3)$	$n > 4t$
Algorithm 3 with VB	$1 \times 6 = 6$	$\log_2 n$	$O(n^3)$	$n > 3t$

Table 2: Cost of the ITB consensus algorithms

7 A randomized VB-based Byzantine binary consensus algorithm

This section presents a randomized Byzantine binary consensus algorithm (that can be used as the underlying BBC algorithm). The additional power needed to solve consensus is given here by random coins. In addition to being optimal from a resilience point of view ($t < n/3$), this algorithm has two noteworthy features:

- It is based on the validated broadcast abstraction, and
- Each round requires 6 communication steps (a single VB-broadcast instance).

When looking at Byzantine consensus algorithms that are optimal from a resilience point of view (i.e., algorithms able to cope with up to $\lfloor (n-1)/3 \rfloor$ faulty processes), the best consensus algorithm we are aware of has rounds made up of three communication steps [7]. Moreover, this algorithm is based on signatures (public key cryptography). As far as signature-free algorithms are concerned, the best resilience-optimal algorithm, that uses control information whose size is only $O(\log_2 n)$ we are aware of, is the one described in [24, 23], which requires five communication steps per round. Algorithm 4 that is presented in this section is signature-free and requires six communication steps per round. The fifth step in [23] is necessary to ensure errorless termination as explained in the original paper [24].

7.1 Randomized model

The asynchronous system is equipped with a *common coin* as defined by Rabin [21] and improved in [7] in order to get rid of the trusted dealer. Such an oracle is denoted CC, hence the system model is $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, CC]$. A common coin can be seen as a global entity that delivers a sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to processes (each bit b_r has the value 0 or 1, with probability $1/2$).

More precisely, this oracle provides the processes with a primitive denoted random() that returns a bit each time it is called by a process. In addition to being random, this bit has the following global property: the rth invocation of random() by any correct process p_i returns it the bit b_r. This means the same random bit b_r is returned to each correct process as the result of its rth invocation of random(). It is important to notice that the network has no access to the common coin, which corresponds to the *oblivious scheduler* model [2]. (The reader interested in the implementation of a common coin can consult [2, 7].)

On randomized consensus When using additional computing power provided by random coins, the consensus termination property can no longer be deterministic. The *Randomized Consensus* problem is defined by C-validity (Obligation), C-agreement plus the following termination property. [3, 21]:

- Proba-C-Termination: Every correct process decides with probability 1.

7.2 The algorithm

Underlying principles and description of the algorithm In Algorithm 4, a process p_i invokes the function bin_propose(v_i) where v_i is the value it proposes. It decides when it executes the statement decide(v) (line 07). The design of Algorithm 4 is close to an algorithm we have proposed in [13]. Its fundamental difference is that it is resilience-optimal ($t < n/3$), while the one described in [13] requires $t < n/5$.

The local variable est_i of process p_i keeps its current estimate of the decision value (initially, $est_i = v_i$). The processes proceed by consecutive asynchronous rounds. Thus, the pair (r_i, est_i) of a correct process p_i describes its current state (r_i is p_i's current round number). The first part of Algorithm 4 consists of lines 01-04 that describes the communication of the current round. The second part, made up of lines 05-10, defines the management of the local estimate est_i and the decision rule. More precisely, we have the following.

- At every round r_i, each correct process p_i VB-broadcasts EST(r_i, est_i), and waits until it has VB-delivered EST$(r_i, -)$ from at least $n - t$ processes (lines 02-04).
- In the second part, p_i first computes the random number s associated with the current round r_i (line 05). Then, p_i checks if it has received a non-$\perp$ value v from at least $n - 2t$ different processes, and v is the only non-$\perp$ value in rec_i (predicate at line 06). If this predicate holds, p_i adopts v as new estimate (line 07) and decides the random value s if $v = s$ (line 08). If the predicate is false, p_i updates its estimate est_i to the random value s. In all cases, p_i starts a new asynchronous round.

The statement decide() allows the invoking process to decide but does not stop its execution. Hence, a process executes rounds forever. This facilitates the description of the algorithm. Using techniques such as the one developed in [13] allows a process to both decide and stop.

Remark It is possible to add the following test after line 04:

if $(\exists v : \#_v(rec_i) \geq n - t)$ **then** decide(v) **end if**

This allows the algorithm to always terminate in a single round whatever the value of the common coin when all correct processes propose the same value and no process exhibits a Byzantine behavior. This scenario is very likely to happen in actual executions.

```
operation bin_propose(v_i)
est_i ← v_i; r_i ← 0;
repeat forever
(01)  r_i ← r_i + 1;
(02)  VB_broadcast EST(r_i, est_i);
(03)  let rec_i = multiset of values est such that EST(r_i, est) has been VB_delivered to p_i;
(04)  wait until (|rec_i| ≥ n − t);
(05)  s_i ← random();
(06)  if (∃v ≠ ⊥ : #_v(rec_i) ≥ n − 2t) ∧ (rec_i contains a single non-⊥ value)
(07)    then est_i ← v;
(08)         if (v = s) ∧ (p_i has not yet decided) then decide(s) end if
(09)    else est_i ← s
(10)  end if
end repeat.
```

Algorithm 4: A binary Byzantine consensus algorithm based on VB-broadcast

8 Proof

Lemma 3. *Let $n > 3t$. Consider the situation where, at the beginning of a round r, all correct processes p_i have the same estimate value v. The correct processes will never change their estimates, thereafter.*

Proof As all correct processes VB-broadcast the same value v at the beginning of round r (line 02), it follows from the VB-validity property of the VB-broadcast, that v is validated by all correct processes. Hence, the only values that can be VB-delivered by a process are v and $\perp$.

Moreover, by the VB-validity and the VB-termination properties, all correct processes eventually VB-deliver v at least from all correct processes. Each correct process p_i will VB-deliver at least $n-2t$ values v as p_i waits for $n-t$ messages (line 04), among which at most t are VB-broadcast by Byzantine processes. Let us remark that, due to the VB-validation property, a value $w \neq v$ VB-broadcast by Byzantine process p_j cannot be validated, and consequently $\perp$ or no value at all is VB-delivered from p_j.

It follows that the predicate of line 06 is satisfied, and consequently p_i sets est_i to v (line 07), which concludes the proof of the lemma. $\square_{Lemma\ 3}$

Let $COND(v, i)$ be the predicate that process p_i tests at line 06.

Lemma 4. *Let $n > 3t$. If two correct processes p_i and p_j are such that both $COND(v, i)$ and $COND(w, j)$ hold at round r, then $v = w$.*

Proof By the VB-uniformity property of VB-broadcast, no two correct processes VB-deliver different values from the same process. Hence, if $COND(v, i)$ holds for some process p_i, no other correct process p_j can VB-deliver a value $w \neq v$ from the same set of processes. Consequently, if p_j VB-delivers a value $w \neq v$, then the number of occurrences of w is necessarily at most $t < n - 2t$, and consequently $COND(w, j)$ cannot be satisfied. $\square_{Lemma\ 4}$

Lemma 5. [C-validity (Obligation)] *Let $n > 3t$. If all correct processes propose the same value v, then no value $v' \neq v$ can be decided.*

Proof This lemma is an immediate consequence of Lemma 3, when we consider $r = 1$. As all estimates of correct processes remain equal to v, it follows from line 08 that no value $v' \neq v$ can be returned by a correct process. $\square_{Lemma\ 5}$

Lemma 6. [C-agreement] *No two correct processes decide different values.*

Proof Let r be the first round during which correct processes decide. If two processes p_i and p_j decide at line 08 of round r, they decide the value s computed by the common coin for that round. Moreover, before deciding during round r a process previously updated its estimate to value s. Hence, all processes that decide during round r have their estimates equal to s. Let us also notice that, if two processes p_i and p_j are such that both $COND(v, i)$ and $COND(w, j)$ hold, by Lemma 4 we have $v = w$, which means that they decide the same value.

Let us now consider a correct process p_k such that $COND(v', k)$ does not hold during round r. It follows from line 08 that p_k updates its estimate to s. Hence, all correct processes start round $r + 1$ with their estimates equal to s. It then follows from Lemma 4 that they keep always the same value from round $r+1$. As a decided value is an estimate value, only value v can be decided. $\square_{Lemma\ 6}$

Lemma 7. [Proba-C-termination] *Each correct process decides with probability* 1.

Proof No correct process remains blocked forever during a round r. This follows from the fact that, at every round, a correct process p_i waits for the VB-delivery of a message $\text{EST}(r, -)$ from $n - t$ distinct processes, and at every round each correct VB-broadcasts such a message that (due to the VB-termination property) entails a corresponding VB-delivery at each correct process.

Claim. With probability 1, there is a round r at the end of which all correct processes have the same estimate value. *End of the claim.*

Assuming the claim holds, it follows from Lemma 3 that all the correct processes p_i keep their estimate value $est_i = v$ and consequently the predicate $COND(v, i)$ (line 06) is true at every round. Due to common coin CC, it follows that, with probability 1, there is eventually a round in which the random() outputs v. Then, the condition of line 08 evaluates to true, and all correct processes decide.

Proof of the claim. We need to prove that, with probability 1, there is a round at the end of which all correct processes have the same estimate value. Let us consider a round r.

- Observe that if all correct processes execute line 09 then, at the end of r, they all adopt the same value (defined by the common coin) by the end of r. The claim directly follows.
- If all the correct processes execute line 07, due to Lemma 4 they adopt the same value v as their estimate, and the claim follows.
- The third case is when some correct processes execute line 07 and (by Lemma 4) adopt the same value v, while others execute line 09 and adopt the same value s.

  Due to the properties of the common coin, the value it computes at a given round is independent from the values it computes at the other rounds (and also from the Byzantine behavior and the network scheduler). Thus, s is equal to v with probability $p = 1/2$. Let $P(r)$ be the following probability (where var^r is the value of var at round r): $P(r) = \text{Probability}[\exists r' : r' \leq r : v^{r'} = s^{r'}]$. We have $P(r) = p + (1 - p)p + \cdots + (1 - p)^{r-1}p$. So, $P(r) = 1 - (1 - p)^r$. As $\lim_{r \to +\infty} P(r) = 1$, the claim follows.

End of the proof of the claim. $\square_{Lemma\ 7}$

Theorem 4. *Algorithm 4 solves the randomized binary consensus problem in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathsf{VB}, CC]$.*

Proof Follows from lemmas 5, 6 and 7. $\square_{Theorem\ 4}$

Theorem 5. *Let $n > 3t$. The expected decision time of Algorithm 4 is constant.*

Proof As indicated in the proof of Lemma 7, termination is obtained in two phases. First, all correct processes must adopt the same value v. Second, the outcome of the common coin has be the same as the commonly adopted value v.

Additionally, from the proof of Lemma 7, there is only one situation in which correct processes do not adopt the same value. This is when the predicate of line 06 is satisfied for a subset of correct processes and not for the other correct processes. Thus, the expected number of rounds for this to happen is bounded by 2. As for the second phase, here again, the probability that the value output by the common coin is the same as the value held by all correct processes is $1/2$. Thus, the expected time for this to occur is also 2. Combining the two phases, the expected termination time is 4 rounds (i.e., a small constant). $\square_{Theorem\ 5}$

9 Conclusion

The paper has presented a family of multivalued intrusion-free Byzantine consensus algorithms. The intrusion-freedom property means that no value proposed only by Byzantine processes can ever be decided. These consensus algorithms are built on top of appropriate broadcast abstractions. One of these abstractions, called validated broadcast, is new (and can be interesting by itself to solve other problems than consensus). Moreover, all proposed algorithms are signature-free (hence efficient).

The intrusion-free consensus algorithm based on the validated broadcast abstraction has several noteworthy features: it is optimal from a resilience point of view ($t < n/3$), each round requires only a single validated broadcast invocation which costs four communication steps and the size of message control information is $O(\log_2 n)$. This improves on the best signature-free (non-intrusion-free) resilience-optimal Byzantine consensus know so far (which requires five communication steps per round [23]).

The paper has also presented a novel randomized binary Byzantine consensus algorithm that is resilient-optimal and, in a very interesting way, is also based on the validated broadcast abstraction.

References

[1] Aguilera M.K., Frolund S., Hadzilacos V., Horn S. and Toueg S., Abortable and Query-abortable Objects and their Efficient Implementation. *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing (PODC'07)*, pp. 23-32, 2007.

[2] Aspnes J., Lower Bounds for Distributed Coin Flipping and Randomized Consensus. *JACM*, 45(3):415-450, 1998.

[3] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing(PODC'83)*, ACM Press, pp. 27-30, 1983.

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

[5] Bracha G. and Toueg S., Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4):824-840, 1985.

[6] Brasileiro F., Greve F., Mostéfaoui A. and Raynal M., Consensus in One Communication Step. *Proc. 6th Int'l Conference on Parallel Computing Technologies (PaCT'01)*, Springer Verlag LNCS #2127, pp. 42-50, 2001.

[7] Cachin Ch., Kursawe K. and Shoup V., Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography. *Proc. 19th Annual ACM Symp. on Principles of Distributed Computing (PODC'00)*, pp. 123-132, 2000.

[8] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 43(2):225-267, 1996.

[9] 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.

[10] Dwork C., Lynch N. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *JACM*, 35(2), 288-323, 1988.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] Mostéfaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach. *Proc. 13th Int'l Symp. on Distributed Computing (DISC'99)*, LNCS #1693, pp. 49-63, 1999.

[18] Mostéfaoui A., Raynal M. and Tronel F., From Binary Consensus to Multivalued Consensus in Asynchronous Message-Passing Systems. *Information Processing Letters*, 73:207-213, 2000.

[19] Okun M., Byzantine Agreement. *Springer Verlag Encyclopedia of Algorithms*, pp. 116-119, 2008.

[20] Pease M., R. Shostak R. and Lamport L., Reaching Agreement in the Presence of Faults. *JACM*, 27:228-234, 1980.

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

[22] Song Y.J. and van Renesse R., Bosco: One-Step Byzantine Asynchronous Consensus. *Proc. 22th Symposium on Distributed Computing (DISC'08)*, Springer Verlag LNCS #5218, 438-450, 2008.

[23] Srikanth T.K. and Toueg S., Simulating Authenticated Broadcasts to Derive Simple Fault-tolerant Algorithms. *Distributed Computing*, 2:80-94, 1987.

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

[25] Turpin R. and Coan B.A., Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement. *Information Processing Letters*, 18:73-76, 1984.