Signature-Free Asynchronous Byzantine Systems: From Multivalued to Binary Consensus with $t < n/3$, $O(n^2)$ Messages, and Constant Time

Achour Mostéfaoui[1] and Michel Raynal[2,3]

[1] LINA, Université de Nantes, 44322 Nantes Cedex, France
[2] Institut Universitaire de France
[3] IRISA, Université de Rennes 35042 Rennes Cedex, France

Abstract. This paper presents a new algorithm that reduces multivalued consensus to binary consensus in an asynchronous message-passing system made up of n processes where up to t may commit Byzantine failures. This algorithm has the following noteworthy properties: it assumes $t < n/3$ (and is consequently optimal from a resilience point of view), uses $O(n^2)$ messages, has a constant time complexity, and does not use signatures. The design of this reduction algorithm relies on two new all-to-all communication abstractions. The first one allows the non-faulty processes to reduce the number of proposed values to c, where c is a small constant. The second communication abstraction allows each non-faulty process to compute a set of (proposed) values such that, if the set of a non-faulty process contains a single value, then this value belongs to the set of any non-faulty process. Both communication abstractions have an $O(n^2)$ message complexity and a constant time complexity. The reduction of multivalued Byzantine consensus to binary Byzantine consensus is then a simple sequential use of these communication abstractions. To the best of our knowledge, this is the first asynchronous message-passing algorithm that reduces multivalued consensus to binary consensus with $O(n^2)$ messages and constant time complexity (measured with the longest causal chain of messages) in the presence of up to $t < n/3$ Byzantine processes, and without using cryptography techniques. Moreover, this reduction algorithm uses a single instance of the underlying binary consensus, and tolerates message re-ordering by Byzantine processes.

1 Introduction

Consensus in Asynchronous Byzantine Systems. The consensus problem lies at the center of fault-tolerant distributed computing. Assuming that each non-faulty process proposes a value, its formulation is particularly simple, namely, each non-faulty process decides a value (termination), the non-faulty processes decide the same value (agreement), and the decided value is related to the proposed values (validity); the way the decided value is related to the proposed values depends on the failure model. Consensus is binary when only two values can be proposed by the processes, otherwise it is multivalued.

Byzantine failures were introduced in the context of synchronous distributed systems [17,28,31], and then investigated in the context of asynchronous distributed systems [2,19,30]. A process has a *Byzantine* behavior (or commits a Byzantine failure) when it arbitrarily deviates from its intended behavior: it then commits a Byzantine

C. Scheideler (Ed.): SIROCCO 2015, LNCS 9439, pp. 194–208, 2015.
DOI: 10.1007/978-3-319-25258-2_14

failure (otherwise we say it is *non-faulty*). 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.

Several validity properties have been considered for Byzantine consensus. This paper considers the following one: a decided value is a value that was proposed by a non-faulty process or a default value denoted $\perp$. Moreover, to prevent trivial or useless solutions, if all the non-faulty processes propose the same value, $\perp$ cannot be decided. As these properties prevent a value proposed only by faulty processes to be decided, such a consensus is called *intrusion-tolerant Byzantine* (ITB) consensus [7,24].

Solving Byzantine Consensus. 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., see [9,17,28,33]) that Byzantine consensus cannot be solved when $t \geq n/3$, be the system synchronous or asynchronous, or be the algorithm allowed to use random numbers or not.

As far as asynchronous systems are concerned, it is well-known that there is no deterministic consensus algorithm as soon as one process may crash [10], which means that Byzantine consensus cannot be solved either as soon as 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., see [3,7,13,22,29]), assumption on message delivery schedules (e.g., [5,33]), failure detectors suited to Byzantine systems (e.g., [12,15]), additional –deterministic or probabilistic– synchrony assumptions (e.g., [5,9,20]), or restrictions on the vectors of input values proposed by the processes (e.g., [11,23]). A reduction of atomic broadcast to consensus in the presence of Byzantine processes is presented in [21].

Finally, for multivalued Byzantine consensus, another approach consists in considering a system model enriched with an algorithm solving (for free) binary Byzantine consensus. This reduction approach has been first proposed in the context of synchronous systems [34]. (See [16,18,27] for recent works for such synchronous systems.) Reductions for asynchronous systems where the communication is by message-passing can be found in [6,7,26]. The case where communication is by read/write registers is investigated in [32]. This reduction approach is the approach adopted in this paper to address multivalued Byzantine consensus.

Contributions of the Paper. Considering asynchronous message-passing systems, this paper presents a new reduction from multivalued Byzantine consensus to binary Byzantine consensus, that has the following properties:

- It tolerates up to $t < n/3$ Byzantine processes,
- Its message cost is $O(n^2)$,
- Its time complexity is constant,
- It tolerates message re-ordering by Byzantine processes,
- It does not use cryptography techniques,
- It uses a single instance of the underlying binary Byzantine consensus.

A simple and efficient Byzantine Binary consensus algorithm has recently been proposed in [22]. This algorithm, which is based on Rabin's common coin, is signature-free and round-based, requires $t < n/3$, has an $O(n^2)$ message complexity per round, and its expected number of rounds is constant. It follows that, when the reduction algorithm proposed in this paper is combined with this binary consensus algorithm, we obtain a Byzantine multivalued consensus algorithm that has the five properties listed previously. To our knowledge, this is the first Byzantine multivalued consensus algorithm that is signature-free, optimal with respect to resilience ($t < n/3$), has an $O(n^2)$ expected message complexity, a constant expected time complexity, and tolerates the re-ordering of message deliveries by Byzantine processes.

The design of the proposed reduction algorithm is based on two new communication abstractions, which are *all-to-all* communication abstractions. The first allows the non-faulty processes to reduce the number of values they propose to $k \leq c$ values where c is a known constant. More precisely, $c = 6$ when $t < n/3$ (worst case), $c = 4$ when $n = 4t$, and $c = 3$ when $t < n/4$. The second communication abstraction allows each non-faulty process to compute a set of (proposed) values such that, if the set of a non-faulty process contains a single value, then this value belongs to the set of any non-faulty process. Both communication abstractions have an $O(n^2)$ message complexity and a constant time complexity.

The structure of the resulting Byzantine multivalued consensus algorithm is as follows. It uses the first communication abstraction to reduce the number of proposed values to a constant. Then, it uses sequentially twice the second communication abstraction to provide each non-faulty process with a binary value that constitutes the value it proposes to the underlying binary Byzantine consensus algorithm. Finally, the value decided by a non-faulty process is determined by the output (0 or 1) returned by the underlying binary Byzantine consensus algorithm. Thanks to the communication abstractions, this reduction algorithm is particularly simple (which is a first class design property).

Roadmap. The paper is composed of 6 sections. Section 2 presents the computing model, and defines the multivalued ITB consensus problem. Section 3 defines the first communication abstraction (called RD-broadcast) that reduces the number of proposed values to a constant, presents an algorithm that implements it, and proves it correct. Section 4 defines the second communication abstraction (called MV-broadcast), presents an algorithm that implements it, and proves it correct. Section 5 presents the algorithm reducing multivalued consensus to binary consensus in the presence of Byzantine processes. Due to page limitation, the reader will find all proofs in [25].

2 Computing Model and Intrusion-Tolerant Byzantine Consensus

2.1 Distributed Computing 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 $\text{TAG}(m)$ 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 $\text{TAG}(m)$ is a macro-operation which stands for “**for each** $j \in \{1, \ldots, n\}$ send $\text{TAG}(m)$ 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 of processes receives the message).

Failure Model. Up to t processes may exhibit a *Byzantine* behavior. 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. Moreover, Byzantine processes can collude to “pollute” the computation. A process that exhibits a Byzantine behavior is also called *faulty*. Otherwise, it is *non-faulty*.

Let us notice that, as each pair of processes is connected by a channel, no Byzantine process can impersonate another process. Byzantine processes can influence the message delivery schedule, but cannot affect network reliability. More generally, the model does not assume a computationally-limited adversary.

Discarding Messages from Byzantine Processes. If, according to its algorithm, a process p_j is assumed to send a single message $\text{TAG}()$ to a process p_i, then p_i processes only the first message $\text{TAG}(v)$ it receives from p_j. This means that, if p_j is Byzantine and sends several messages $\text{TAG}(v)$, $\text{TAG}(v')$ where $v' \neq v$, etc., all of them except the first one are discarded by their receivers.

Notation. In the following, this computation model is denoted $\mathcal{BAMP}_{n,t}[\emptyset]$. In the following, this model is restricted with the constraint on $t < n/3$ and is consequently denoted $\mathcal{BAMP}_{n,t}[n > 3t]$.

2.2 Measuring Time Complexity

When computing the time complexity, we consider the longest sequence of messages $m_1, \ldots, m_z$ whose sending are causally related, i.e., for each $x \in [2..z]$, the reception of m_{x-1} is a requirement for the sending of m_x. The time complexity is the length of this longest sequence. Moreover, we implicitly consider that, in each invocation of an all-to-all communication abstraction, the non-faulty processes invoke the abstraction.

2.3 Multivalued Intrusion-Tolerant Byzantine Consensus

Byzantine Consensus. This problem has been informally stated in the Introduction. Assuming that each non-faulty 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 non-faulty process eventually decides on a value, and terminates.
- C-One-shot. A non-faulty process decides at most once.
- C-Agreement. No two non-faulty processes decide on different values.
- C-Obligation (validity). If all the non-faulty processes propose the same value v, then v is decided.

Intrusion-Tolerant Byzantine (ITB) *Consensus.* Byzantine algorithms differ in the validity properties they satisfy. In classical Byzantine consensus, if the non-faulty processes do not propose the same value, they can decide any value (this is captured by the previous C-Obligation property.

As indicated in the Introduction, we are interested here in a more constrained version of the consensus problem in which a value proposed only by faulty processes cannot be decided. This was first investigated in a systematic way in [7,24]. This consensus problem instance is defined by the C-Termination, C-One-shot, C-Agreement, and C-Obligation properties stated above plus the following C-Non-intrusion (validity) property, where $\perp$ is a predefined default value, which cannot be proposed by a process.

- C-Non-intrusion (validity). A value decided by a non-faulty process is a value proposed by a non-faulty 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[1].

Remark on the Binary Consensus. Interestingly, binary Byzantine consensus (only two values can be proposed by processes) has the following property.

Property 1. The ITB binary consensus problem is such that, if a value v is decided by a non-faulty process, it was proposed by a non-faulty process.
This means that, when considering the ITB binary consensus, $\perp$ can be safely replaced by any of the two possible binary values.

3 The Reducing All-to-All Broadcast Abstraction

3.1 Definition

The *reducing broadcast* abstraction (RD-broadcast) is a one-shot all-to-all communication abstraction, whose aim is to reduce the number of values that are broadcast

[1] Directing the non-faulty processes to decide a predefined default value –instead of an arbitrary value, possibly proposed only by faulty processes– in specific circumstances, is close to the notion of an *abortable* object as defined in [14,32] where an operation is allowed to abort in the presence of concurrency. This notion of an abortable object is different from the notion of a query-abortable object introduced in [1].

to a constant. RD-broadcast provides the processes with a single operation denoted RD_broadcast(). This operation has an input parameter, and returns a value. It is assumed that all the non-faulty processes invoke this operation.

When a process p_i invokes RD_broadcast(v_i) we say that it "RD-broadcasts" the value v_i. When a process returns a value v from an invocation of RD_broadcast(), we say that it "RD-delivers" a value (or a value is RB-delivered). The default value denoted $\bot_{rd}$ cannot be RD-broadcast but can be RD-delivered. RD-broadcast is defined by the following properties.

– RD-Termination. Every non-faulty process eventually RD-delivers a value.
– RD-Integrity. No non-faulty process RD-delivers more than one value.
– RD-Justification. The value RD-delivered by a non-faulty process is either a value RD-broadcast by a non-faulty process, or the default value $\bot_{rd}$.
– RD-Obligation. If the non-faulty processes RD-broadcast the same value v, none of them RD-delivers the default value $\bot_{rd}$.
– RD-Reduction. The number of values that are RD-delivered by the non-faulty processes is upper bounded by a constant c.

3.2 An RD-Broadcast Algorithm

An algorithm implementing the RD-broadcast abstraction is described in Figure 1. This algorithm assumes $t < n/3$. The aim of the local variable rd_del_i is to contain the value RD-delivered by p_i; this variable is initialized to "?", a default value that cannot be RD-delivered by non-faulty processes.

When a process p_i invokes RD_broadcast MSG(v_i), it first broadcasts the message INIT(v_i), and then waits until it is allowed to RD-deliver a value (line 1). During this waiting period, p_i receives and processes the messages INIT() or ECHO() sent by the algorithm.

```
let $rd\_pset_i(x)$ denote the set of processes from which $p_i$ has received INIT$(x)$ or ECHO$(x)$.

operation RD_broadcast$(v_i)$ is
(1)  broadcast INIT$(v_i)$; wait$(rd\_del_i \neq$ "?"); return$(rd\_del_i)$.

when INIT$(v)$ or ECHO$(v)$ is received do
(2)  if $(v \neq v_i) \wedge$ (INIT$(v)$ rec. from $(n-2t)$ different proc.) $\wedge$ (ECHO$(v)$ never broadcast)
(3)     then broadcast ECHO$(v)$
(4)  end if;
(5)  if $\big(\exists\, x : (x \neq v_i) \wedge (|rd\_pset_i(x)| \geq t+1)\big)$ then $rd\_del_i \leftarrow \bot_{rd}$ end if;
(6)  if $\big(\exists\, x : |rd\_pset_i(x)| \geq n-t\big)$ then $rd\_del_i \leftarrow x$ end if;
(7)  let $w$ be the value such that, $\forall\, x$ received by $p_i$: $|rd\_pset_i(w)| \geq |rd\_pset_i(x)|$;
(8)  if $\big(|\cup_x\, rd\_pset_i(x)| - |rd\_pset_i(w)| \geq t+1\big)$ then $rd\_del_i \leftarrow \bot_{rd}$ end if.
```

Fig. 1. An algorithm implementing RD-broadcast in $\mathcal{BAMP}_{n,t}[n > 3t]$

The behavior of a process p_i on its server side, i.e. when –while waiting– it receives a message INIT(v) or ECHO(v), is made up of two phases.

– Conditional communication phase (lines 2-4). If the received value v is different from the value v_i it has RD-broadcast, and $\text{INIT}(v)$ has been received from "enough" processes (namely $(n-2t)$), p_i broadcasts the message $\text{ECHO}(v)$ if not yet done. Let us notice that, as $n-2t \geq t+1$, this means that $\text{INIT}(v)$ was broadcast by at least one non-faulty process.
– Try-to-deliver phase (lines 5-8). Then, for any value x it has seen, a process p_i computes first the set $rd\_pset_i(x)$ composed of the processes from which p_i has received a message $\text{INIT}(x)$ or $\text{ECHO}(x)$. If there is a value x, different from v_i, that has been received from $(t+1)$ different processes, if p_i is non-faulty, it knows that at least two different values have been RD-broadcast by non-faulty processes (its own value v_i, plus another one). In this case, p_i RD-delivers the default value $\bot_{rd}$ (line 5 and line 1). The RD-delivery of a value by p_i terminates its invocation of the RD-broadcast.

  If the predicate of line 5 is not satisfied, p_i checks if there is a value x received from at least $(n-t)$ distinct processes (line 6). Let us notice that, in this case, it is possible that x was RD-broadcast by all correct processes. Hence, p_i RD-delivers this value.

  Finally, if p_i has not yet assigned a value to rd_del_i, it computes the value w that, up to now, it received the most often in an $\text{INIT}()$ or $\text{ECHO}()$ message (line 7). If there are at least $(t+1)$ different processes that sent $\text{INIT}()$ or $\text{ECHO}()$ messages with values different from w (this is captured by the predicate of line 8), it is impossible for p_i to have in the future the same value received from $(n-t)$ distinct processes. This claim is trivially true for w, because at least $(t+1)$ processes sent values different from w. As no value $w' \neq w$ was received more than w, the claim is also true for any such value w'. So, the predicate of line 6 will never be satisfied at p_i, and consequently p_i RD-delivers the default value $\bot_{rd}$.

3.3 Proof of the RD-Broadcast Algorithm

All the proofs assume $t < n/3$.

Lemma 1. *Let nb_echo be the maximal number of different values that a non-faulty process may echo at line* 3. *We have:* $(n/3 > t > n/4) \Rightarrow (nb\_echo \leq 2)$ *and* $(n/4 \geq t) \Rightarrow (nb\_echo \leq 1)$.

Lemma 2. *At most c different values can be* RD-delivered *by the non-faulty processes, where $c = 6$ when $n/3 > t$, $c = 4$ when $n = 4t$, and $c = 3$ when $n/4 > t$.*

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

Theorem 2. *The number of messages sent by the non-faulty processes is upper bounded by $O(n^2)$. Moreover, in addition to a value sent by a process, a message carries a single bit of control information. The time complexity is $O(1)$.*

3.4 RD-Broadcast vs Byzantine k-Set Agreement

In the k-set agreement problem, each process proposes a value, and at most k different values can be decided by the non-faulty processes. It is shown in [8] that the solvability of k-set agreement in the presence of Byzantine processes depends crucially on the validity properties that are considered.

As the reader can easily check, the specification of the RD-broadcast abstraction defines an instance of the intrusion-based Byzantine c-set agreement problem, where c is the constant defined in Lemma 2. It follows that the algorithm presented in Figure 1 solves this Byzantine k-set agreement instance for any $k \geq c$ in the system model $\mathcal{BAMP}_{n,t}[t < n/3]$. (Let us remind that $t < n/3$ is the lower bound on t to solve Byzantine consensus in a *synchronous* system.)

4 The Multivalued Validated All-to-All Broadcast Abstraction

4.1 Definition

The RD-broadcast abstraction reduces the number of values sent by processes to at most six values (five values RD-broadcast by non-faulty processes, plus the default value denoted $\bot_{rd}$), while keeping the number of messages exchanged by non-faulty processes in $O(n^2)$.

Differently, assuming that each non-faulty process broadcasts a value, and at most k different values are broadcast (where k does not need to be known by the processes), the aim of the one-shot *multivalued validated all-to-all broadcast* abstraction (in short MV-broadcast) is to provide each non-faulty process with an appropriate subset of values (called *validated* values), which can be used to solve multivalued ITB consensus. To that end, the fundamental property of MV-broadcast that is used is the following: if a non-faulty process returns a set with a single value, the set returned by any other non-faulty process contains this value. Moreover, from an efficiency point of view, an important point that has to be satisfied is that the message cost of an MV-broadcast instance has to be $O(kn^2)$.

To MV-broadcast a value v_i, a process p_i invokes the operation $\mathsf{MV\_broadcast}(v_i)$. This invocation returns to p_i a non-empty set a values, which consists of validated values, plus possibly a default value denoted $\bot_{mv}$. This default value cannot be MV-broadcast by a process. Similarly to RD-broadcast, when a process invokes the operation $\mathsf{MV\_broadcast}(v)$, we say that it "MV-broadcast v". MV-broadcast is defined by the following properties.

- MV-Obligation. If all the non-faulty processes MV-broadcast the same value v, then no non-faulty process returns a set containing $\bot_{mv}$.
- MV-Justification. If a non-faulty process p_i returns a set including a value $v \neq \bot_{mv}$, there is a non-faulty process p_j that MV-broadcast v.
- MV-Inclusion. Let set_i and set_j be the sets returned by two non-faulty processes p_i and p_j, respectively. $(set_i = \{w\}) \Rightarrow (w \in set_j)$ (let us notice that w can be $\bot_{mv}$).
- MV-Termination. An invocation of $\mathsf{MV\_broadcast}()$ by a non-faulty process terminates (i.e., returns a non-empty set).

The following property follows directly from the MV-Inclusion property.

– MV-Singleton. Let set_i and set_j be the sets returned by two non-faulty processes p_i and p_j, respectively. $[(set_i = \{v\}) \wedge (set_j = \{w\})] \Rightarrow (v = w)$.

```
let $mv\_pset1_i(x)$ denote the set of processes from which $p_i$ has received $\text{MV\_VAL1}(x)$;
$mv\_val2_i$: set, initially $\emptyset$, of pairs $\langle$process index, value$\rangle$ received in messages $\text{MV\_VAL2}()$.

operation MV_broadcast $\text{MSG}(v_i)$ is
(1)  broadcast $\text{MV\_VAL1}(v_i)$; wait $(\exists\, v$ such that $|mv\_pset1_i(v)| \geq 2t+1)$;
                                   % in the previous wait stateemnt $v$ can be $\perp_{mv}$ %
(2)  broadcast $\text{MV\_VAL2}(v)$;  wait $(|mv\_val2_i| \geq n-t)$;
(3)  return $(\{x \mid \langle -, x\rangle \in mv\_val2_i\})$.

when $\text{MV\_VAL1}(y)$ is received do   %  $y$ can be $\perp_{mv}$  %
(4)  if $((|mv\_pset1_i(y)| \geq t+1) \wedge$ ($\text{MV\_VAL1}()$ not broadcast))
                                   then broadcast $\text{MV\_VAL1}(y)$ end if;
(5)  let $w$ be the value such that, $\forall\, x$ received by $p_i$: $|mv\_pset1_i(w)| \geq |mv\_pset1_i(x)|$;
(6)  if $((|\cup_x\, mv\_pset1_i(x)| - |mv\_pset1_i(w)| \geq t+1)$
                                   $\wedge$ ($\text{MV\_VAL1}(\perp_{mv})$ not broadcast))
(7)     then broadcast $\text{MV\_VAL1}(\perp_{mv})$ end if.

when $\text{MV\_VAL2}(x)$ is received from $p_j$ do   %  $x$ can be $\perp_{mv}$  %
(8)  wait$(|mv\_pset1_i(x)| \geq 2t+1)$;
(9)  $mv\_val2_i \leftarrow mv\_val2_i \cup \langle j, x\rangle$.
```

Fig. 2. An algorithm implementing MV-broadcast in $\mathcal{BAMP}_{n,t}[n > 3t]$

4.2 An MV-Broadcast Algorithm

A two-phase algorithm implementing the MV-broadcast abstraction is described in Figure 2. It assumes $t < n/3$, and –as we will see– its message complexity is $O(kn^2)$.

To be *validated*, a value must have been MV-broadcast by at least one non-faulty process. Hence, for a process to locally know whether a value is validated, it needs to receive it from $(t+1)$ processes.

Each process p_i manages a local variable mv_val2_i, which is a set (initially empty). Its aim is to contain pairs $\langle j, x\rangle$, where j is a process index and x a validated value. The behavior of a non-faulty process p_i is as follows.

– In the first phase (line 1) a process p_i broadcasts its initial value by sending the message $\text{MV\_VAL1}(v_i)$. It then waits until it knows (a) a validated value v (hence it has received $\text{MV\_VAL1}(v)$ from at least $(t+1)$ different processes), (b) and this value v is eventually known by all non-faulty processes. This is captured by the following waiting predicate “the message $\text{MV\_VAL1}(v)$ has been received from at least $(2t+1)$ different processes” used at line 1. From then on, p_i will champion this value v for it to belong to the sets returned by the non-faulty processes.

On it server side concerning the reception of a message $\text{MV\_VAL1}(y)$, a process p_i does the following (line 4). If p_i knows that y is a validated value (i.e., the message $\text{MV\_VAL1}(y)$ was received from least $(t+1)$ processes), (if not yet done) p_i broadcasts the very same message to help the validated value y to be known by all non-faulty processes.

Then, according to its current knowledge of the global state, p_i checks if there is a possibility that no value at all be present enough to be validated. It there is such a possibility, p_i broadcasts $\text{MV\_VAL1}(\bot_{mv})$. To that end (as at line 7 of the RD-broadcast algorithm, Figure 1), p_i computes the value w most received from different processes (lines 5). If at least $(t+1)$ processes have broadcast values different from w, p_i broadcasts $\text{MV\_VAL1}(\bot_{mv})$, if not yet done (lines 6-7); p_i sends the default value because it sees too may different values, and it does not know which ones are from non-faulty processes.

– When it enters the second phase (line 2), a process champions the validated value v it has previously computed with the waiting predicate of line 1. This is done by broadcasting the message $\text{MV\_VAL2}(v)$. It then waits until the set mv_val2_i contains at least $(n-t)$ pairs $\langle j, x\rangle$, and finally returns the set of values contained in these pairs (line 3). Let us remind that those are validated values.

  On its server side, when a process p_i receives a message $\text{MV\_VAL2}(x)$ from a process p_j, it waits until it has received a message $\text{MV\_VAL1}(x)$ from at least $(2t+1)$ different processes. This is needed because Byzantine processes can send spurious messages $\text{MV\_VAL2}(x)$ while they have not validated the value x. More precisely, let us notice that the waiting predicate $(|mv\_pset1_i(x)| \geq 2t+1)$ used by p_i at line 8 is the same as the one used at line 2 by p_j –if it is non-faulty– to champion the value x. Hence, in case p_j is not non-faulty, p_i waits until the same validation predicate $(|mv\_pset1_i(x)| \geq 2t+1)$ becomes true before accepting to process the message $\text{MV\_VAL2}(x)$ sent by p_j.

Remark. Let us notice that this algorithm is tolerant to message duplication. Moreover, while a non-faulty process is not allowed to MV-broadcast the default value $\bot_{mv}$, a Byzantine process can do it. Let us also remark that $\bot_{mv}$ is the only default value associated with the MV-broadcast abstraction. Hence, for MV-broadcast, $\bot_{rd}$ is a "normal" value, which can be MV-broadcast, as any value different from $\bot_{mv}$.

4.3 Proof of the MV-Broadcast Algorithm

As previously, all the proofs assume $t < n/3$.

Lemma 3. *The waiting predicate* $(\exists\ v$ such that $|mv\_pset1_i(v)| \geq 2t+1)$ *(used at line* 1*) is eventually satisfied at any non-faulty process* p_i.

Lemma 4. *The waiting predicate* $(|mv\_val2_i| \geq n-t)$ *(used at line* 2*) is eventually satisfied at any non-faulty process* p_i.

Lemma 5. *If all non-faulty processes* MV-broadcast *the same value* v*, no non-faulty process returns a set containing* $\bot_{mv}$.

Lemma 6. *If the set returned by a non-faulty process p_i contains a value $v \neq \bot_{mv}$, then v has been* MV-broadcast *by a non-faulty process.*

Lemma 7. *Let set_i and set_j be the sets returned by two non-faulty processes p_i and p_j, respectively. $(set_i = \{w\}) \Rightarrow (w \in set_j)$.*

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

Theorem 4. *Let us assume that at most k different values are* MV-broadcast *by the processes. The number of messages sent by the non-faulty processes is upper bounded by $O(kn^2)$. A message needs to carry a single bit of control information. The time complexity is $O(1)$.*

5 Multivalued Intrusion-Tolerant Byzantine Consensus

The multivalued intrusion-tolerant Byzantine (ITB) consensus problem was defined in Section 2.3. A signature-free algorithm that solves it despite up to $t < n/3$ Byzantine processes is described in this section. This algorithm is such that the expected number of messages exchanged by the non-faulty processes is $O(n^2)$, and its expected time complexity is constant.

5.1 Enriched Computation Model for Multivalued ITB Consensus

In the following, as announced in the introduction, we consider that the additional computational power that allows multivalued ITB consensus to be solved in $\mathcal{BAMP}_{n,t}[t < n/3]$ is an underlying Byzantine binary consensus (BBC) algorithm. Let $\mathcal{BAMP}_{n,t}[t < n/3, \text{BBC}]$ denote the system model $\mathcal{BAMP}_{n,t}[t < n/3]$ enriched with a BBC algorithm. BBC algorithms are described in several papers (e.g., [4,7,13,22,33]).

To obtain a multivalued ITB consensus algorithm with an $O(n^2)$ expected message complexity and a constant expected time complexity, we implicitly consider that the underlying BBC algorithm is the one presented in [22].

5.2 An Efficient Algorithm Solving the Multivalued ITB Consensus Problem

The algorithm is described in Figure 3. The multivalued consensus operation that is built is denoted mv_propose(), while the underlying binary consensus operation it uses is denoted bin_propose(). Extremely simple, this algorithm can be decomposed in four phases. The first three phases are communication phases, while the last phase exploits the result of the previous phases to reduce multivalued Byzantine consensus to BBC.

The second and the third phases are two distinct instances of the MV-broadcast abstraction. Not to confuse them, their corresponding broadcast operations are denoted MV_broadcast$_1$(), and MV_broadcast$_2$(), respectively. Similarly, their default values are denoted $\bot_{mv1}$ and $\bot_{mv2}$. It is assumed that the default values $\bot_{rd}$, $\bot_{mv1}$, $\bot_{mv2}$, and $\bot$ (the consensus default value) are all different. The four phases are as follows, where C_PROP denotes the set of values proposed by the non-faulty processes.

```
operation mv_propose($v_i$) is
(1) $rd\_val_i \leftarrow$ RD_broadcast($v_i$);
% ---------------------------------------------------------------
(2) $set1_i \leftarrow$ MV_broadcast$_1(rd\_val_i)$;
    % $p_i, p_j$ non-faulty: $((|set1_i| = 1) \wedge (|set1_j| = 1)) \Rightarrow (set1_i = set1_j)$ %
(3) if $(set1_i = \{w\})$ then $aux_i \leftarrow w$ else $aux_i \leftarrow \perp$ end if;
% ---------------------------------------------------------------
(4) $set2_i \leftarrow$ MV_broadcast$_2(aux_i)$;
    % $p_i, p_j$ non-faulty: $(set2_i = \{w\}) \Rightarrow (w \in set2_j)$ %
% ---------------------------------------------------------------
(5) if $((set2_i = \{w\}) \wedge (w \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}))$
                then $bp_i \leftarrow 1$ else $bp_i \leftarrow 0$ end if;
(6) $bdec_i \leftarrow$ bin_propose($bp_i$);
(7) if $(bdec_i = 1)$ then return($w$) such that $w \in set2_i$ and $w \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$
(8)         else return($\perp$)
(9) end if.
```

Fig. 3. An algorithm implementing multivalued ITB consensus in $\mathcal{BAMP}_{n,t}[n > 3t, \text{BBC}]$

- The first phase consists of an RD-broadcast instance. Each non-faulty process p_i invokes RD_broadcast(v_i), where v_i is the value it proposes to consensus, and stores the returned value in its local variable rd_val_i (line 1). Due to properties of the RD-broadcast abstraction, we have

$$rd\_val_i \in RD\_VAL \text{ where } RD\_VAL \subseteq C\_PROP \cup \{\perp_{rd}\},$$

  and (due to Lemma 2) $|RD\_VAL| \leq 6$. Moreover, the message cost of this phase is the one of the RD-broadcast, i.e., $O(n^2)$.
- The second phase (lines 2 and 3) consists of the first MV-broadcast instance, namely, a process p_i invokes MV_broadcast$_1(rd\_val_i)$ from which it obtains the non-empty set $set1_i$. Due to the properties of the MV-broadcast abstraction, we have

$$set1_i \subseteq MV\_VAL_1,$$

$$\text{where } MV\_VAL_1 \subseteq RD\_VAL \cup \{\perp_{mv1}\} \subseteq C\_PROP \cup \{\perp_{rd}, \perp_{mv1}\}.$$

  Moreover, due to the MV-singleton property, we also have

$$\big((|set1_i| = 1) \wedge (|set1_j| = 1)\big) \Rightarrow (set1_i = set1_j).$$

  Then, according to the value of $set1_i$, p_i prepares a value aux_i it will broadcast in the second MV-broadcast instance. If $set1_i = \{w\}$, $aux_i = w$, otherwise $aux_i = \perp$ (the consensus default value).

  Let $AUX = \cup_{i \in \mathcal{C}}\{aux_i\}$, where $\mathcal{C}$ denotes the set of non-faulty processes. While preserving the $O(n^2)$ message complexity, the aim of the lines 2 and 3 is to ensure the following property

$$AUX = \{v\} \vee AUX = \{\perp\} \vee AUX = \{v, \perp\}, \text{ where } v \in MV\_VAL_1.$$

  Let us notice that, thanks to the MV-Justification property, the set AUX cannot contain a value proposed only by Byzantine processes.

- The third phase (line 4) is a second instance of the MV-broadcast abstraction. The values MV-broadcast by the non-faulty processes are values of the set AUX. So, the set $set2_i$ returned by a non-faulty process p_i is such that

$$set2_i \subseteq MV\_VAL_2 \text{ where } MV\_VAL_2 \subseteq AUX \cup \{\bot_{mv2}\},$$

  and, due to the MV-Inclusion property, the sets returned to any two non-faulty processes p_i and p_j are such that $(set2_i = \{w\}) \Rightarrow (w \in set2_j)$.
- The last phase (lines 5-9) is where the underlying BBC algorithm is exploited. If $set2_i$ contains a single value, that is not a default value, p_i proposes 1 to the underlying BBC algorithm. Otherwise, it proposes 0. Then, according to the value $bdec_i$ returned by the BBC algorithm, there are two cases. If $bdec_i = 1$, p_i return the value of $set2_i$ which is not a default value (line 7). Otherwise, $bdec_i = 0$ and p_i returns the default value $\bot$.

5.3 Proof of the Multivalued ITB Consensus Algorithm and Two Remarks

Theorem 5. *The algorithm described in Figure* 3 *solves the* multivalued ITB consensus *problem in the computing model* $\mathcal{BAMP}_{n,t}[t < n/3, \text{BBC}]$.

Theorem 6. *Let us assume an underlying* BBC *algorithm whose expected message complexity is* $O(n^2)$ *and expected time complexity is constant (e.g., the one presented in* [22]*). When considering the non-faulty processes, the expected message complexity of the multivalued ITB consensus algorithm described in Figure* 3 *is* $O(n^2)$*, and its expected time complexity is constant.*

Remark 1. Let us remark that, if we suppress the invocation of the RD-broadcast abstraction, and replace line 1 by the statement "$rd\_val_i \leftarrow v_i$", the multivalued ITB consensus remains correct. This modification saves the two communication steps involved in the RD-broadcast, but loses the $O(n^2)$ message complexity, which is now $O(kn^3)$ (this follows from Theorem 4 and the fact that $k \in [1..n]$ is the number of distinct values broadcast by correct processes).

Remark 2. The algorithm of Figure 2 uses two instances of the MV-broadcast abstraction. It is an open problem to know if it is possible to design an algorithm based on a single instance of it.

6 Conclusion

This paper presented an asynchronous message-passing algorithm which reduces multivalued consensus to binary consensus in the presence of up to $t < n/3$ Byzantine processes (n being the total number of processes). This algorithm has the following noteworthy features: its message complexity is $O(n^2)$, its time complexity is $O(1)$, and it does not rely on cryptographic techniques. As far as we know, this is the first consensus reduction owning all these properties, while being optimal with respect to the value of t. This algorithm relies on two new all-to-all communication abstractions. These abstractions consider the values that are broadcast, and not the fact that "this" value was broadcast by "this" process. This simple observation allowed us to design an efficient reduction algorithm. (An n-multiplexing of a one-to-all broadcast abstraction would entail an $O(n^3)$ message complexity.) Interestingly, this reduction algorithm uses a single instance of the Byzantine binary consensus, and tolerates message re-ordering by Byzantine processes.

When combined with the binary Byzantine consensus algorithm presented in [22], we obtain the best algorithm known so far (as far as we know) for multivalued Byzantine consensus in a message-passing asynchronous system (where "best" is with respect the value of t, the message and time complexities, and the absence of limit on the computational power of the adversary).

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

References

1. Aguilera, M.K., Frolund, S., Hadzilacos, V., Horn, S., Toueg, S.: Abortable and query-abortable objects and their efficient implementation. In: Proc. 26th Annual ACM Symposium on Principles of Distributed Computing (PODC 2007), pp. 23–32 (2007)
2. Attiya, H., Welch, J.: Distributed computing: fundamentals, simulations and advanced topics, 2nd edn., p. 414 pages. Wiley Interscience (2004)
3. Ben-Or, M.: Another advantage of free choice: completely asynchronous agreement protocols. In: Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC 1983), pp. 27–30. ACM Press (1983)
4. Bracha, G.: Asynchronous Byzantine agreement protocols. Information & Computation 75(2), 130–143 (1987)
5. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. Journal of the ACM 32(4), 824–840 (1985)
6. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 524–541. Springer, Heidelberg (2001)
7. Correia, M., Ferreira Neves, N., Verissimo, P.: From consensus to atomic broadcast: time-free Byzantine-resistant protocols without signatures. Computer Journal 49(1), 82–96 (2006)
8. De Prisco, R., Malkhi, D., Reiter, M.: On k-set consensus problems in asynchronous systems. Transactions on Parallel and Distributed Systems 12(1), 7–21 (2001)
9. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
10. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (1985)
11. Friedman, R., Mostéfaoui, A., Rajsbaum, S., Raynal, M.: Distributed agreement problems and their connection with error-correcting codes. IEEE Transactions on Computers 56(7), 865–875 (2007)
12. Friedman, R., Mostéfaoui, A., 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., 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. Hadzilacos, V., Toueg, S.: On deterministic abortable objects. In: Proc. 32th Annual ACM Symposium on Principles of Distributed Computing (PODC 2013), pp. 4–12 (2013)
15. Kihlstrom, K.P., Moser, L.E., Melliar-Smith, P.M.: Byzantine fault detectors for solving consensus. The Computer Journal 46(1), 16–35 (2003)
16. King, V., Saia, J.: Breaking the $O(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. In: Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC 2011), pp. 420–429. ACM Press (2011)
17. Lamport, L., Shostack, R., Pease, M.: The Byzantine generals problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
18. Liang, G., Vaidya, N.: Error-free multi-valued consensus with Byzantine failures. In: Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC 2011), pp. 11–20. ACM Press (2011)
19. Lynch, N.A.: Distributed algorithms, 872 pages. Morgan Kaufmann Pub., San Francisco (1996)
20. Martin, J.-P., Alvisi, L.: Fast Byzantine consensus. IEEE Transactions on Dependable and Secure Computing 3(3), 202–215 (2006)
21. Milosevic, Z., Hutle, M., Schiper, A.: On the reduction of atomic broadcast to consensus with Byzantine faults. In: Proc. 30th IEEE Int'l Symposium on Reliable Distributed Systems (SRDS 2011), pp. 235–244. IEEE Computer Press (2011)
22. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous Byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In: Proc. 33rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2014), pp. 2–9. ACM Press (2014)
23. Mostéfaoui, A., Rajsbaum, S., Raynal, M.: Conditions on input vectors for consensus solvability in asynchronous distributed systems. Journal of the ACM 50(6), 922–954 (2003)
24. Mostéfaoui, A., Raynal, M.: Signature-free broadcast-based intrusion tolerance: never decide a Byzantine value. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 143–158. Springer, Heidelberg (2010)
25. Mostéfaoui, A., Raynal, M.: Asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, $O(1)$ time, and no signature. Tech Report 2014, 17 pages, IRISA, Université de Rennes (F) (2015), `https://hal.inria.fr/hal-01102496`
26. Mostéfaoui, A., Raynal, M., Tronel, F.: From binary consensus to multivalued consensus in asynchronous message-passing systems. Information Processing Letters 73, 207–213 (2000)
27. Patra, A.: Error-free multi-valued broadcast and Byzantine agreement with optimal communication complexity. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 34–49. Springer, Heidelberg (2011)
28. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27, 228–234 (1980)
29. Rabin, M.: Randomized Byzantine generals. In: Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS 1983), pp. 116–124. IEEE Computer Society Press (1983)
30. Raynal, M.: Communication and agreement abstractions for fault-tolerant asynchronous distributed systems. Morgan & Claypool, 251 pages (2010) ISBN 978-1-60845-293-4
31. Raynal, M.: Fault-tolerant agreement in synchronous message-passing systems, 165 pages. Morgan & Claypool Publishers (2010) ISBN 978-1-60845-525-6
32. Raynal, M.: Concurrent programming: algorithms, principles and foundations, 515 pages. Springer (2013)
33. Toueg, S.: Randomized Byzantine agreement. In: Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC 1984), pp. 163–178. ACM Press (1984)
34. Turpin, R., Coan, B.A.: Extending binary Byzantine agreement to multivalued Byzantine agreement. Information Processing Letters 18, 73–76 (1984)