Asynchronous Byzantine Systems: From Multivalued to Binary Consensus with t < n/3, O(n 2) Messages, O(1) Time, and no Signature

Achour Mostefaoui, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 2024 – January 2015

UMR IRISA

Asynchronous Byzantine Systems: From Multivalued to Binary Consensus with $t < n/3$, $O(n^2)$ Messages, $O(1)$ Time, and no Signature

Achour Mostéfaouil[*] Michel Raynal[** ***]

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 tolerates message reordering by Byzantine processes.

Key-words: Asynchronous message-passing system, Broadcast abstraction, Byzantine process, Consensus, Distributed algorithm, Intrusion tolerance, Multivalued consensus, Optimal resilience, Randomized binary consensus, Signature-free algorithm.

Une réduction du consensus multivalué au consensus binaire en présence d'asynchronisme, de $t < n/3$ processus byzantins, avec un temps constant, $O(n^2)$ messages, et pas de signatures

Résumé : *Cet article présente un algorithme réparti qui, dans un système asynchrone de n processus qui communiquent par passage de messages, et qui comprend jusqu'à t processus byzantins, ramène le problème du consensus multivalué au problème du consensus binaire. Cette réduction est optimale par rapport à t ($t < n/3$), requiert un temps constant et $O(n^2)$ messages, et n'utilise aucun élément cryptographique (i.e., pas de signatures). Elle considère donc un adversaire donc la la puissance de calcul peut être illimitée.*

Mots clés : *Système à passage de messages, Processus byzantin, consensus binaire, consensus multivalué, réduction algorithmique.*

---

[*] LINA, Université de Nantes, 44322 Nantes Cedex, France
[**] Institut Universitaire de France
[***] ASAP : équipe commune avec l'Université de Rennes 1 et Inria

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, 27, 30], and then investigated in the context of asynchronous distributed systems [2, 19, 29]. A process has a *Byzantine* behavior (or commits a Byzantine failure) when it arbitrarily deviates from its intended behavior: it then commits a Byzantine 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., [9, 17, 27, 32]) that Byzantine consensus cannot be solved when $t \geq n/3$, be the system synchronous or asynchronous, be the algorithm allowed to use cryptography or not, or be the algorithm allowed to use random numbers or not.

As far as 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., [3, 7, 13, 22, 28]), assumption on message delivery schedules(e.g., [5, 32]), 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 [33]. Recent works for such synchronous systems can be found in [16, 18, 26]. Reductions for asynchronous systems where the communication is by message-passing can be found in [6, 7, 25]. The case where communication is by read/write registers is investigated in [31]. 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.

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 four 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 and a constant expected time complexity.

The design of the 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 first 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 define 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.

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 loss, duplicate, modify, or create messages. "Point-to-point" means that there is a bi-directional communication channel between each pair of processes. Hence, when a process receives a message, it can identify its sender.

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

The operation broadcast TAG(m) is a macro-operation which stands for "**for each** $j \in \{1, \ldots, n\}$ send TAG(m) to p_j **end for**".This operation is usually called *unreliable* broadcast (if the sender commits a failure in the middle of the **for** loop, it is possible that only an arbitrary subset of processes receives a 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 modify 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 TAG() to a process p_i, then p_i processes only the first message TAG(v) it receives from p_j. This means that, if p_j is Byzantine and sends several messages TAG(v), TAG(v') where $v' \neq v$, etc., all of them except the first one are discarded. Let us observe that this does not prevent multiple copies of the first message TAG() to be received and processed.

Notation 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 simultaneously.

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 In Byzantine consensus, if the non-faulty processes do not propose the same value, they can decide any value. 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 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 property (which is a 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 be can be proposed by processes) has the following property.

Property 1. *The* ITB *binary consensus problem is such that if a value is decided by a non-faulty process, this value can always be a value proposed by a non-faulty process.*

Proof It follows from the C-obligation property that any non-faulty process decides b when all the non-faulty processes proposed b. Otherwise, at least one non-faulty process proposed g, and then any value b or g may be decided by a non-faulty process without violating the C-Non-intrusion property. $\square_{Property\ 1}$

This means that, when considering the ITB binary consensus, $\perp$ can be safely replaced by any of b or g.

[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, 31] 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].

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 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 invokes 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 also say that it "RB-delivers" a value (or a value is RB-delivered). The default value denoted $\perp_{rd}$ cannot be RB-broadcast but can be RB-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 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 $\perp_{rd}$.
- RD-Obligation. If the non-faulty processes RD-broadcast the same value v, none of them RD-delivers the default value $\perp_{rd}$.
- 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 broadcasts the message INIT(v_i), and 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) received from $(n-2t)$ different processes) $\wedge$ (ECHO(v) never broadcast)
(3) **then** broadcast ECHO(v)
(4) **end if**;
(5) **if** $\big(\exists\, x \neq v_i :\ |rd\_pset_i(x)| \geq t+1\big)$ **then** $rd\_del_i \leftarrow \perp_{rd}$ **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** $rd\_del_i \leftarrow \perp_{rd}$ **end if**.

Figure 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 INIT(v) has been received from "enough" processes (namely $(n-2t)$), p_i broadcasts the message ECHO(v) if not yet done. Let us notice that, as $n - 2t \geq t + 1$, this means that INIT(v) has been received from at least one non-faulty process.

- Try-to-deliver phase (lines 5-8). Then, for any value x, a process p_i computes first the set $rd\_pset_i(x)$ composed of the processes from which p_i has received a message INIT(x) or ECHO(x). If there is a value x, different from v_i, that has been received from $(t+1)$ different processes, if it is non-faulty, p_i 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 $\perp_{rd}$ (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 v received from at least $(n-t)$ distinct processes (line 6). Let us notice that, in this case, it is possible that v has been 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 has received the most often in an INIT() or ECHO() message (line 7). If there are at least $(t+1)$ different processes that sent INIT() or ECHO() messages with values different from w (this is captured by the predicate of line 7), 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 $w' \neq w$ has been 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 $\perp_{rd}$.

Remark The predicate of line 6 could be strengthened as follows:

$$\exists\, x:\ (|rd\_pset_i(x)| \geq n-t)\ \wedge\ (\text{INIT}(x) \text{ received from } \geq n-2t \text{ different processes}).$$

This predicate is more constraining than the predicate $(\exists\, x:\ |rd\_pset_i(x)| \geq n-t)$ used to RD-deliver values at p_i. Consequently, it can reduce the number of values which are RD-delivered at p_i.

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 echoes at line* 3*. We have: $(n/3 > t > n/4) \Rightarrow (nb\_echo \leq 2)$ and $(n/4 \geq t) \Rightarrow (nb\_echo \leq 1)$.*

Proof Let us first consider the greatest possible value of t, which corresponds to $n = 3t+1$. Let us observe that a process p_i receives at most one message INIT() from each other process (otherwise, it knows that the sender is Byzantine). Moreover, to broadcast ECHO(v), where $v \neq v_i$, p_i needs to receive INIT(v) from $n-2t = t+1$ different processes (predicate of line 2). It follows, from these observations and the fact that $n = (t+1)+(t+1)+(t-1) < 3(n-2t)$, that p_i can broadcast at most two messages ECHO() carrying distinct values. As p_i broadcasts at most once a message ECHO() carrying a given value v, it follows that $(n/3 > t) \Rightarrow (nb\_echo \leq 2)$.

Let us now consider the case $n = 4t+\alpha \geq 4t$, where $\alpha \geq 0$. In this case $n-2t = 2t+\alpha \geq 2t$. As the broadcast by p_i of a message ECHO(v) requires the reception of INIT(v) from at least $2t+\alpha$ different processes, it is possible that there is such a value v. On the other side, as a process p_i receives at most one message INIT() from each other process, it is not possible for a process p_i to receive $n-2t \geq 2t$ messages carrying a value w, different from both v and v_i. It follows that $(n/4 \geq t) \Rightarrow (nb\_echo \leq 1)$, which concludes the proof of the lemma. $\square_{Lemma\ 1}$

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 $t > n/4$.*

Proof Let us consider a "worst" execution, i.e., an execution in which t processes are Byzantine. Let us call "vote" (for a value $v \neq \perp_{rd}$) a message INIT(v) or ECHO(v) sent by a process.

To be RD-delivered by a non-faulty process p_i, a value (different from $\perp_{rd}$) must be received from $(n-t)$ different processes (line 6). Due to Lemma 1, a non-faulty process can vote for at most three distinct values, while a Byzantine process can vote for any number of values. To maximize the total number of values that can be RD-delivered by the set of non-faulty processes, we consider the worst case where (a) there are t Byzantine

processes, and (b) if a non-faulty process p_i RD-delivers a value v (line 6) it has received for this value $(n-2t)$ votes from non-faulty processes, and a vote from each of the Byzantine processes. In this way, we have to consider the maximal number of votes that can be sent by the non-faulty processes and the fact that only $(n-2t)$ of them are needed to RD-deliver a value. There are three cases.

Case $n > 3t$. At most $3(n-t)$ votes can be sent by the non-faulty processes. It follows that the total number of different values that can be RD-delivered by the set of non-faulty processes is (where the "$<$" comes from $n > 3t$):

$$\frac{3(n-t)}{n-2t} = \frac{3n-6t+3t}{n-2t} = 3 + \frac{3t}{n-2t} < 3 + \frac{3t}{3t-2t} = 6.$$

Hence the total number of RD-broadcast values that can be returned by the whole set of non-faulty processes is at most 5. When adding the default value $\perp_{rd}$, we obtain $c = 6$.

Case $n > 4t$. It then follows from Lemma 1 that there are at most $2(n-t)$ votes sent by non-faulty processes. With the same reasoning as before we obtain (where "$<$" comes from $n > 4t$):

$$\frac{2(n-t)}{n-2t} = \frac{2n-4t+2t}{n-2t} = 2 + \frac{2t}{n-2t} < 2 + \frac{2t}{4t-2t} = 3.$$

Hence the total number of RB-broadcast values that can be returned by the whole set of non-faulty processes is at most 2. When adding the default value $\perp_{rd}$, we obtain $c = 3$.

Finally, for $n = 4t$ we have $\frac{2(n-t)}{n-2t} = \frac{6t}{2t} = 3$. Hence, at most $c = 4$ different values can be returned by the whole set of non-faulty processes when $n = 4t$. $\square_{Lemma\ 2}$

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

Proof The proof of the RD-Integrity property (a non-faulty process RD-delivers at most one value) follows directly from the text of the algorithm. The proof of the RD-Reduction property follows from Lemma 2.

To prove the RD-Justification property we show that a value RB-broadcast only by Byzantine processes cannot be RD-delivered by a non-faulty process p_i. Let us assume the worst case, namely, there are t Byzantine processes and all of them RB-broadcast the same value v. As $n > 3t$, no non-faulty process can receive INIT(v) from $n - 2t \geq t + 1$ different processes. Hence, no non-faulty process broadcasts ECHO(v) at line 3. If follows that no non-faulty process receives messages INIT(v) or ECHO(v) from $(n-t)$ different processes. Hence, the predicate $|rd\_pset_i(v)| \geq n - t$ (line 6) cannot become satisfied at a non-faulty process p_i, which concludes the proof of the RD-Justification property.

To prove the RD-Obligation property, let us assume that all the non-faulty processes RD-broadcast the same value v. We have to show that no non-faulty process RD-delivers $\perp_{rd}$, i.e., executes the statement $rd\_del_i \leftarrow \perp_{rd}$ at line 5 or line 8. Let us first observe that, even if all Byzantine processes propose the same value $w \neq v$, the predicate of line 5 cannot be satisfied for w. Finally, as the (at least $n-t$) non-faulty processes RD-broadcast the same value v, whether the (at most t) Byzantine processes have sent the same or different values in INIT() or ECHO() messages, it is not possible for the predicate of line 8 to become satisfied.

To prove the RD-Termination property, we consider two cases. Let us first consider that some value v is RD-broadcast by at least $n - 2t \geq t + 1$ non-faulty processes. If follows from line 2 that each other non-faulty process broadcasts ECHO(v). Hence, each non-faulty process p_i receives messages INIT() or ECHO() messages, carrying the value v, from at least $(n-t)$ processes. Consequently, the predicate of line 6 becomes eventually satisfied at every non-faulty process, and –if not yet done– each of them RD-delivers a value.

Let us now consider the case where there is no value v that is RD-broadcast by $n-2t$ non-faulty processes. Considering a non-faulty process p_i, let us assume that the predicates of line 5 and line 6 are never satisfied. We show that the predicate of line 8 becomes then eventually satisfied. Let v be the value that p_i received the most often from different processes. By assumption it received this value from g non-faulty processes where $g < n-2t$,

and from b Byzantine processes, where $0 \leq b \leq t$. As $g+b < (n-2t)+b$, at most $(n-2t-1)+b$ processes have sent the value v to p_i. Moreover, as p_i does not terminate at line 5 or line 6, it receives a message from each non-faulty process, i.e., from at least $(n-t)$ processes, from which we conclude that p_i eventually receives messages from at least $(n-t)+b$ different processes. It follows that at least $(n-t+b)-((n-2t-1)+b) = t+1$ different processes have sent to p_i values different from v. Consequently, the predicate of line 8 is eventually satisfied and p_i RD-delivers $\perp_{rd}$. $\square_{Theorem\ 1}$

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 $O(1)$.*

Proof It follows from Lemma 1 that a non-faulty process echoes at most two INIT() messages. This means that, it broadcast at most three messages (its own INIT() message, plus two ECHO() messages). As a broadcast costs n messages, the total number of messages sent by the non-faulty processes is $3n^2$. A single bit of control is required to differentiate INIT() and ECHO() messages.

As far as the time complexity is concerned, a message ECHO() can only be sent when a message INIT() is received. Hence, the time complexity is 2. $\square_{Theorem\ 2}$

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 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 five values plus a default value denoted $\perp_{rd}$, 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 $\perp_{mv}$. This default value cannot be MV-broadcast by a process. Similarly to RD-broadcast, when a process invokes $\mathsf{MV\_broadcast}(v)$, wee 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 $\perp_{mv}$.
- MV-Justification. If a non-faulty process p_i returns a set including a value $v \neq \perp_{mv}$, 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 $\perp_{mv}$).

- MV-Termination. An invocation of 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. $[(set_i = \{v\}) \wedge (set_i = \{w\})] \Rightarrow (v = w)$.

let $mv\_pset1_i(x)$ **denote** the set of processes from which p_i has received MV_VAL1(x);
mv_val2_i is a set of pairs $\langle$process index, value$\rangle$, initially empty.

operation MV_broadcast MSG(v_i) **is**
(1) broadcast MV_VAL1(v_i); wait $\big(\exists\, v$ such that $|mv\_pset1_i(v)| \geq 2t+1\big)$; % v can be $\perp_{mv}$ %
(2) broadcast MV_VAL2(v); wait $\big(|mv\_val2_i| \geq n-t\big)$;
(3) return $(\{x \mid \langle -, x\rangle \in mv\_val2_i\})$.

when MV_VAL1(y) **is received do** % y can be $\perp_{mv}$ %
(4) **if** $\big((|mv\_pset1_i(y)| \geq t+1) \wedge$ (MV_VAL1() not yet broadcast)$\big)$ **then** broadcast 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** $\big((|\cup_x\, mv\_pset1_i(x)| - |mv\_pset1_i(w)| \geq t+1) \wedge$ (MV_VAL1$(\perp_{mv})$ never broadcast)$\big)$
(7) **then** broadcast MV_VAL1$(\perp_{mv})$
(8) **end if**;

when MV_VAL2(x) **is received from** p_j **do** % x can be $\perp_{mv}$ %
(9) wait$(|mv_pset1_i(x)| \geq 2t+1)$;
(10) $mv\_val2_i \leftarrow mv\_val2_i \cup \langle j, x\rangle$.

Figure 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 MV_VAL1(v_i). It then waits until it knows (a) a validated value v (hence it has received 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 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 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 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 MV_VAL1$(\perp_{mv})$ in order no set returned by a non-faulty process be empty. To that end (as at lines 6-7 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 MV_VAL1$(\perp_{mv})$, if not yet done (lines 6-8); 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 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 MV_VAL2(x) from a process p_j, it waits until it has received a message MV_VAL1(x) from at least $(2t+1)$ different processes. This is needed because Byzantine processes can send spurious messages 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 9 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 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 $\perp_{mv}$, a Byzantine process can do it. Let us also remark that $\perp_{mv}$ is the only default value associated with the MV-broadcast abstraction. Hence, for MV-broadcast, $\perp_{rd}$ is a "normal" value, which can be MV-broadcast, as any value different from $\perp_{mv}$.

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.

Proof Let us consider the the following predicate P: "There is a value v and a finite time after which at least $(t+1)$ non-faulty processes have sent the message MV_VAL1(v)". We consider two cases.

Case 1: The predicate P is satisfied. In this case, it follows from line 4 that every non-faulty process eventually broadcasts MV_VAL1(v). As there are at least $n-t \geq 2t+1$ non-faulty processes, the predicate ($\exists\, v$ such that $|mv\_pset1_i(v)| \geq 2t+1$) becomes eventually true at every non-faulty process. Consequently, no non-faulty process can block forever at line 1.

Case 2: The predicate P is not satisfied. Let p_i be a non-faulty process. Let vmr_i be the most often received value by p_i, from different processes (as defined at line 5). Let c be the number of non-faulty processes that sent MV_VAL1(vmr_i). As P is not satisfied, $c \leq t$.

As there are at least $(n-t)$ non-faulty processes, p_i receives at least $(n-t)$ messages MV_VAL1$()$. More precisely, p_i receives these messages from $(n-t+b)$ different processes, where $b \in [0..t]$ is the additional number of –non-faulty or Byzantine– processes that sent a message to p_i. Among these $(n-t+b)$ processes, at most $(c+b)$ messages MV_VAL1$()$ carry the value vmr_i (the upper bound $(c+b)$ is attained when the b additional processes are Byzantine and sent MV_VAL1(vmr_i) to p_i). It follows that at least $(n-t+b)-(c+b) = n-t-c$ of them sent to p_i values different from vmr_i. As $c \leq t$ and $n > 3t$, we have $n-t-c \geq n-2t \geq t+1$. It follows that the predicate of line 6 is eventually satisfied, and p_i broadcasts then the message MV_VAL1$(\perp_{mv})$ (line 7). This reasoning applies to any non-faulty process p_i (possibly with values different from vmr_i). Hence, each non-faulty process p_i eventually receives MV_VAL1$(\perp_{mv})$ from at least $(n-t)$ processes. Finally, as $n-t \geq 2t+1$, the predicate $(|mv\_pset1_i(\perp_{mv})| \geq 2t+1)$ is eventually satisfied at each non-faulty process p_i, which concludes the proof of the second case. $\square_{Lemma\ 3}$

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.

Proof It follows from Lemma 3 that no non-faulty process blocks forever in the wait statement of line 1. Hence, each non-faulty process p_i broadcasts a message MV_VAL2(w_i) at line 2, where the value w_i is such that $(|mv\_pset1_i(w_i)| \geq 2t+1)$. If follows from this predicate that at least $(2t+1)$ processes sent MV_VAL1(w_i)

to p_i from which we conclude that the message MV_VAL1(w_i) has been broadcast by at least $(t+1)$ non-faulty processes. It consequently follows from line 4 that (if not yet done) each non-faulty process broadcasts the message MV_VAL1(w_i). Hence, at each non-faulty process p_j, we eventually have $|mv\_pset1_j(w_i)| \geq 2t+1$. Finally, it follows from the predicate of line 9 that, at line 10, every non-faulty process p_j adds $\langle i, w_i\rangle$ to its local set mv_val2_j.

As there are $(n-t)$ non-faulty processes, it follows that we eventually have $|mv\_pset1_i(w_i)| \geq n-t$ at every non-faulty process, and the predicate of line 2 becomes eventually satisfied, which concludes the proof of the lemma. $\square_{Lemma\ 4}$

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

Proof To prove the MV-Obligation property, let us assume that all non-faulty processes broadcast MV_VAL1(v) at line 1. If follows from the predicate of line 4 that, even if all the Byzantine processes broadcast the same value $w \neq v$, no non-faulty process broadcasts w at line 4, and consequently at most t values different from v can be broadcast in MV_VAL1() messages. Let us consider a non-faulty process p_i in a worst case execution, which occurs when all the Byzantine processes broadcast the same value $w \neq v$. Let us notice that $|mv\_pset1_i(v)|$ monotonically increases from 0 to $(n-t)$, and $|mv\_pset1_i(w)|$ monotonically increases from 0 to t. There are two cases.

- If, during the execution, there are periods where $|mv\_pset1_i(w)| \geq |mv\_pset1_i(v)|$, due to the range of $|mv\_pset1_i(w)|$, the predicate $|mv\_pset1_i(v)| - |mv\_pset1_i(w)| \geq t+1$ returns *false* when it is evaluated by p_i. It follows that, when $|mv\_pset1_i(w)| \geq |mv\_pset1_i(v)|$, p_i cannot broadcast MV_VAL1$(\perp_{mv1})$ at line 7.
- During the period where $|mv\_pset1_i(v)| \geq |mv\_pset1_i(w)|$ (this necessarily occurs, and eventually remains true forever), due to the range of $|mv\_pset1_i(v)|$ and $|mv\_pset1_i(w)|$, and similarly to the previous case, the predicate $|mv\_pset1_i(w)| - |mv\_pset1_i(v)| \geq t+1$, returns *false* when it is evaluated by p_i, and, as before, p_i cannot broadcast MV_VAL1$(\perp_{mv})$ at line 7.

It follows that no non-faulty process broadcasts MV_VAL1$(\perp_{mv})$. As only (at most t) Byzantine processes may broadcast MV_VAL1$(\perp_{mv})$, the predicate $(|mv\_pset1_i(\perp_{mv})| \geq 2t+1)$ evaluated at line 9 can never be satisfied at a non-faulty process p_i. Hence, the set mv_val2_i of a non-faulty process p_i cannot contain a pair $\langle -, \perp_{mv}\rangle$, and consequently p_i cannot return a set including $\perp_{mv}$, which concludes the proof the MV-Obligation property. $\square_{Lemma\ 5}$

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

Proof To prove the MV-Justification property, we show that a value MV-broadcast only by Byzantine processes cannot be added to the set mv_val2_i of a non-faulty process p_i. Let us consider the worst case where there are t Byzantine processes and all broadcast MV_VAL1(v), where v, is not MV-broadcast by non-faulty processes. Due to the predicate of line 4, no non-faulty process p_i echoes the value v. Hence, v can be received only from the t Byzantine processes. It follows that the predicate $(|mv\_pset1_i(v)| \geq 2t+1)$ of line 9 cannot be satisfied at a non-faulty process p_i, and consequently $\langle -, v\rangle$ cannot be added to mv_val2_i. $\square_{Lemma\ 6}$

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

Proof If a non-faulty process p_i returns $set_i = \{w\}$, it follows from the predicate of line 2 that mv_val2_i contains $(n-t)$ different pairs $\langle$process index,$w\rangle$. Hence, p_i has received the message MV_VAL2(w) from $(n-t)$ different processes.

Before a non-faulty process p_j returns a set set_j, it added $(n-t)$ different pairs to mv_del2_j. As a non-faulty process processes at most one message MV_VAL2() from every other process (see the discarding of erroneous messages from Byzantine processes in section 2.1), it follows that p_j has received and processed messages MV_VAL2() from $(n-t)$ different processes, i.e, from at least $n-2t \geq t+1$ non-faulty processes.

As $(n-t)+(t+1) > n$, it follows that there is a non-faulty process p_k that sent the same message MV_VAL2() to both p_i and p_j. Hence, it sent MV_VAL2(w) to p_j, and the pair $\langle k, w\rangle$ is a pair of mv_val2_j, which concludes the proof of the lemma. $\square_{Lemma\ 7}$

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

Proof The proof follows from Lemmas 3 and 4 (MV-Termination), Lemma 5 (MV-Obligation), Lemma 6 (MV-Justification), and Lemma 7 (MV-Inclusion). $\square_{Theorem\ 3}$

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* $O(1)$.

Proof As far as the messages MV_VAL1() are concerned we have the following. Each non-faulty process broadcasts its initial value v_i (line 1), broadcasts at most once any value $v \neq v_i$ it receives (line 4), and broadcasts at most once the default value $\perp_{mv}$ (line 7). It follows that these broadcasts issued by the non-faulty processes generate at most $(k+1)n^2$ MV_VAL1() messages. As each non-faulty process broadcasts a message MV_VAL2() at most once (line 2), it follows that at most n^2 MV_VAL2() messages are sent by the non-faulty processes. Hence, the total number of messages sent by non-faulty processes belongs to $O(n^2)$.

As there are only two message types, a message has to carry a single bit of control information.

As far as the time complexity is concerned, we have the following. A message MV_VAL1() sent by a non-faulty process at line 1 entails at most one forwarding of the same message at line 4 by a non-faulty process. Moreover, the sending of a message MV_VAL2() by a non-faulty process at line 2 is entailed by "enough" reception of messages MV_VAL1(). It follows that the longest sequence of causally related messages is 3. $\square_{Theorem\ 4}$

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, 32]).

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 $\perp_{mv1}$ and $\perp_{mv2}$. It is assumed that the default values $\perp_{rd}$, $\perp_{mv1}$, $\perp_{mv2}$, and $\perp$ (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 ← RD_broadcast(v_i);
%  ------------------------------------------------------------------------
(2)  set1_i ← MV_broadcast_1(rd_val_i);
     %  p_i, p_j non-faulty: ((|set1_i| = 1) ∧ (|set1_j| = 1)) ⇒ (set1_i = set1_j)  %
(3)  if (set1_i = {w}) then aux_i ← w else aux_i ← ⊥ end if;
%  ------------------------------------------------------------------------
(4)  set2_i ← MV_broadcast_2(aux_i);
     %  p_i, p_j non-faulty: (set2_i = {w}) ⇒ (w ∈ set2_j)  %
%  ------------------------------------------------------------------------
(5)  if ((set2_i = {w}) ∧ (w ∉ {⊥_rd, ⊥_mv1, ⊥_mv2, ⊥})) then bp_i ← 1 else bp_i ← 0 end if;
(6)  bdec_i ← bin_propose(bp_i);
(7)  if (bdec_i = 1) then return(w) such that w ∈ set2_i and w ∉ {⊥_rd, ⊥_mv1, ⊥_mv2, ⊥}
(8)                  else return(⊥)
(9)  end if.
```

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

- The first phase consists of an RB-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

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

  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$_1(rd_del_i)$ from which it obtains the non-empty set $set1_i$. Due to the properties of the MV-broadcast abstraction, we have

  $$set1_i \subseteq MV\_VAL_1, \text{ where } MV\_VAL_1 \subseteq RD\_VAL \cup \{\bot_{mv1}\} \subseteq C\_PROP \cup \{\bot_{rd}, \bot_{mv1}\}.$$

  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 $aux_i = \bot$ (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

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

  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

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

Proof The C-termination property follows from the following properties.

- The RD-termination property (line 1).
- The MV-Termination property applied to both instances of MV-broadcast (lines 2 and 4).
- The C-Termination property of the underlying BBC algorithm.
- The fact that, when BBC returns 1:
  - at least one non-faulty process p_i proposed 1 to BBC, and this process is such that $set2_i = \{w\}$ where $w \notin \{\bot_{rd}, \bot_{mv1}, \bot_{mv2}, \bot\}$, and
  - due to the MV-inclusion property of $\mathsf{MV\_broadcast}_2()$, the set $set2_j$ of any non-faulty process p_j contains w, and this is the only non-bottom value in $set2_j$.

Proof of the C-Obligation property. Let us consider the case where all non-faulty processes propose the same value v. It follow from the RD-Obligation property that they all return v from their invocation of $\mathsf{RD\_broadcast}(v)$ (line 1). Hence, they all invoke $\mathsf{MV\_broadcast}_1(v)$ (line 2), from which, due to the MV-Obligation and MV-Justification properties, they all obtain the set $set1_i = \{v\}$. It follows that they all $\mathsf{MV\_broadcast}_2(v)$, from which (for the same reason) they still obtain the same set $set2_i = \{v\}$ (line 4). Consequently all non-faulty processes propose 1 to the underlying BBC algorithm and (due to its C-Obligation property) they all decide 1 (line 6). Hence, they all return v at line 7.

Proof of the C-Non-intrusion property. To show that a value proposed only by Byzantine processes cannot be decided, let us consider the worst case, namely, there are t Byzantine processes and all of them propose a value w, which is not proposed by a non-faulty process. It follows from the RD-Justification of the RD-broadcast abstraction that no non-faulty process p_i obtains w from its invocation of $\mathsf{RD\_broadcast}()$ (line 1). Similarly, due to the MV-Justification of the MV-broadcast abstraction, no non-faulty process p_i can obtain w from its invocations of $\mathsf{MV\_broadcast}_1()$ (line 2) and $\mathsf{MV\_broadcast}_2()$ (line 4). As $v \notin set2_i$, a non-faulty process p_i cannot decided v.

Proof of the C-agreement property. If the value decided by the underlying BBC algorithm is 0, all non-faulty processes decide $\bot$ (line 8). If the value 1 is decided, and all non-faulty processes proposed 1, all their sets $set2_i$ are singletons, which –due to the MV-singleton property– contain the same value w. It follows from lines 5 and 7 that all non-faulty processes decide w.

Let us now consider the last case, where the value 1 is decided by the underlying BBC algorithm, while two non-faulty processes p_i and p_j are such that p_i proposed 1 and p_j proposed 0. As p_i has proposed 1 we have $set2_i = \{w\}$, where w is not a default value. As both p_i and p_j are non-faulty, it follows from the MV-Inclusion property of the second MV-broadcast that $w \in set2_j$. Moreover, as non-faulty processes can MV-broadcast only w or $\bot$ at line 4, it follows from the MV-Inclusion property that w is the only non-default value in $set2_j$. Consequently, p_j decides w at line 7. $\square_{Theorem\ 5}$

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.*

Proof It follows from Theorem 2 that the number of messages sent by the non-faulty processes at line 1 is $O(n^2)$. Due to the RD-reduction property of the RD-broadcast abstraction, there is constant c (where $c \leq 6$, see Lemma 2) such that at most c different values are RD-delivered by the RD-broadcast abstraction. It then follows from Theorem 4 that the number of messages sent in the two MV-broadcasts (lines 2 and 4) by the non-faulty processes is $O(n^2)$. Finally, as the expected number of messages sent by the non-faulty processes in the BBC algorithm is $O(n^2)$, the multivalued ITB consensus algorithm inherits this message complexity. It follows that the expected number of messages sent by the non-faulty processes belongs to $O(n^2)$.

The fact that expected time complexity is constant follows directly from the fact that the time complexities of both the RB-broadcast and MV-Broadcast abstraction are constant (Theorem 2 and Theorem 4), and the underlying BBC algorithm has an expected time complexity that is constant. $\square_{Theorem\ 6}$

Remark 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 $O(kn^3)$, where $k \in [1..n]$ is the number of distinct values broadcast by correct processes.

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 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).

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

[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] Cachin Ch., Kursawe K., Petzold F., and Shoup V., Secure and efficient asynchronous broadcast protocols. *Proc. 21st Annual International Cryptology Conference CRYPTO'01*, Springer LNCS 2139, pp. 524-541, 2001.

[7] 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.

[8] De Prisco R., Malkhi D., and 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., and 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., and 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., and 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. 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] Hadzilacos V. and Toueg S., On deterministic abortable objects. *Proc. 32th Annual ACM Symposium on Principles of Distributed Computing (PODC'13)*, pp. 4-12, 2013.

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

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

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

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

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

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

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

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

[24] Mostéfaoui A. and Raynal M., Signature-free broadcast-based intrusion tolerance: never decide a Byzantine value. *Proc. 14th Int'l Conference On Principles Of Distributed Systems (OPODIS'010)*, Springer LNCS 6490, pp. 144-159, 2010.

[25] 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.

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

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

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

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

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

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

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

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