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

Achour Mostefaoui, Moumen Hamouna, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 2016 – February 2014

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

Achour Mostéfaouil[*] Hamouma Moumen[**] Michel Raynal[*** ****]

Abstract: This paper presents a new round-based asynchronous consensus algorithm that copes with up to $t < n/3$ Byzantine processes, where n is the total number of processes. In addition of being signature-free and optimal with respect to the value of t, this algorithm has several noteworthy properties: the expected number of rounds to decide is four, each round is composed of two or three communication steps and involves $O(n^2)$ messages, and a message is composed of a round number plus a single bit. To attain this goal, the consensus algorithm relies on a common coin as defined by Rabin, and a new extremely simple and powerful broadcast abstraction suited to binary values. The main target when designing this algorithm was to obtain a cheap and simple algorithm. This was motivated by the fact that, among the first-class properties, simplicity –albeit sometimes under-estimated or even ignored– is a major one.

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

Consensus Byzantin Asynchrone avec $t < n/3$ et $O(n^2)$ Messages en Moyenne

Résumé : *Cet article présente un algorithme de consensus binaire en présence de processus Byzantins. Cet algorithme satisfait plusieurs propriétés d'optimalité.*

Mots clés : *Abstraction, Diffusion, Nombre aléatoire, Processus byzantin, Système asynchrone, Optimalité.*

---

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

1 Introduction

Distributed computing Distributed computing occurs when one has to solve a problem in terms of physically distinct entities (usually called nodes, processors, processes, agents, sensors, etc.) such that each entity has only a partial knowledge of the many parameters involved in the problem. In the following, we use the term *process* to denote any computing entity. From an operational point of view this means that the processes of a distributed system need to exchange information, and agree in some way or another, in order to cooperate to a common goal. If processes do not cooperate, the system is no longer a distributed system. Hence, a distributed system has to provide the processes with communication and agreement abstractions.

Understanding and designing distributed applications is not an easy task [3, 10, 26, 33, 34, 35]. This is because, due to the very nature of distributed computing, no process can capture instantaneously the global state of the application it is part of. This comes from the fact that, as processes are geographically localized at distinct places, distributed applications have to cope with the uncertainty created by asynchrony and failures. As a simple example, it is impossible to distinguish a crashed process from a very slow process in an asynchronous system prone to process crashes.

As in sequential computing, a simple approach to facilitate the design of distributed applications consists in designing appropriate abstractions. More generally, computer science is a science of abstraction and distributed computing is the science of distributed abstractions [18]. With such abstractions, the application designer can think about solutions to solve problems at a higher conceptual level than the basic send/receive communication level.

Communication and agreement abstractions Broadcast abstractions are among the most important abstractions encountered in fault-tolerant distributed computing [3, 10, 12, 26, 33]. Roughly speaking, these abstractions allow processes to disseminate information in such a way that specific provable properties concerning this dissemination are satisfied. One of the most popular of these abstractions is reliable broadcast [8, 12].

As far as agreement abstractions are concerned, *non-blocking atomic commit* [23, 27] and *consensus* [17, 31] are certainly the most important abstractions of fault-tolerant distributed computing. Assuming that each process proposes a value, the consensus abstraction allows the non-faulty processes to agree on the same value, which has to satisfy some validity condition depending on both the proposed values and the failure model [3, 26, 33, 34].

Consensus in asynchronous Byzantine systems This paper is on the consensus problem in asynchronous distributed systems where processes can commit Byzantine failures [25]. This failure type has first been introduced in the context of synchronous distributed systems [25, 31, 34], and then investigated in the context of asynchronous distributed systems [3, 26, 33]. A process has a *Byzantine* behavior when it arbitrarily deviates from its intended behavior: it then commits a Byzantine failure (otherwise we say it is *correct*). This bad behavior can be intentional (malicious) or simply the result of a transient fault that altered the local state of a process, thereby modifying its behavior in an unpredictable way. Let us notice that process crashes (unexpected halting) define a strict subset of Byzantine failures.

Related work Let t denote the model upper bound on the number of processes that can have a Byzantine behavior. It is shown in several papers (e.g., [15, 25, 31, 38]) that Byzantine consensus cannot be solved when $t \geq n/3$, be the system synchronous or asynchronous, be the algorithm allowed to use cryptography or not, or be the algorithm allowed to use random numbers or not. As far as synchronous systems are concerned, it has been been shown in [16] that $(t + 1)$ rounds is a lower bound on the number of communication steps needed to solve Byzantine consensus. It has also been shown in [22] that, using randomization, this bound can be improved to an expected $O(\log n)$ value in synchronous full-information systems where $t \leq \frac{n}{3+\epsilon}$ and $\epsilon > 0$.

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 [17], 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., [4, 13, 21, 32]), assumption on message delivery schedules [8, 38], failure detectors suited to Byzantine systems (e.g., [20, 24]), additional –deterministic or probabilistic– synchrony assumptions (e.g., [8, 15, 28]), or restrictions on the vectors of input values proposed by the processes [19, 29].

This paper is on binary consensus algorithms in asynchronous Byzantine systems where the additional computational power is supplied by a *common coin*. Such an object, introduced by Rabin [32], is a distributed object that delivers the same sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to each process. Some of the first randomized algorithms (such as the ones of Ben-Or [4] or Bracha [6]) use local coins. As a consequence they have an expected number of rounds which is exponential (unless $t = O(\sqrt{n})$), which makes them of theoretical interest, but of limited practical use. As randomized algorithms based on a common coin can have an expected number of rounds which is constant, this paper focuses only on such algorithms.

Protocol	$n >$	signatures	msgs/round	bits/msg	steps/round
Rabin 1983 [32]	$10t$	yes	$O(n^2)$	$O(1)$	2
Berman Garay 1993 [5]	$5t$	no	$O(n^2)$	$O(1)$	2
Friedman Mostefaoui Raynal 2005 [21]	$5t$	no	$O(n^2)$	$O(1)$	1
Bracha 1987 [7]	$3t$	no	$O(n^3)$	$O(\log(n))$	9
Srikanth Toueg 1987 [37]	$3t$	no	$O(n^3)$	$O(\log(n))$	5
Toueg 1984 [38]	$3t$	yes	$O(n^3)$	$O(n)$	3
Canetti Rabin 1993 [11]	$3t$	yes	$O(n^2)$	poly(n)	9
Cachin Kursawe Shoup 2000 [9]	$3t$	yes	$O(n^2)$	$O(\ell)$	2
This paper	$3t$	no	$O(n^2)$	$O(1)$	2 or 3

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

Round-based asynchronous Byzantine algorithms based on a common coin are listed in Table 1. All these algorithms, which address binary consensus, are round-based, and, in each of them, each message carries a round number. Hence, when comparing their message size, we do not consider round numbers. We have the following.

- The first algorithm is such that $n < 10t$, has an $O(n^2)$ message complexity, and requires signatures.
- The algorithms of the two next lines are such that $t < n/5$, and their message complexity is $O(n^2)$. These algorithms are simple, signature-free, and use one or two communication steps per round, but none of them is optimal with respect to t-resilience.
- The algorithms of the next three lines are optimal with respect to t, but have an $O(n^3)$ message complexity. Moreover, [38] uses signed messages, while [7] does not use a common coin, and may consequently execute an exponential number of rounds. Due to their message complexity, and their number of communication steps per round, these algorithms are costly.
- The algorithm proposed in [11], although polynomial, has a huge bit complexity. The algorithm presented in [9] is optimal with respect to t, uses only $O(n^2)$ messages per round, has two communication steps per round, and uses signed messages (the value ℓ in the bit complexity corresponds to the size of RSA signatures). However, as noticed by its authors, "because of public key operations, the computational complexity of this protocol is typically higher than those using authentication codes".

Content of the paper The paper first introduces a simple broadcast abstraction suited to binary values, that we call BV-broadcast. This broadcast focuses on values and not on processes, which means that it does not consider the fact that "this" value has been broadcast by "this" process. BV-broadcast, whose implementation is particularly simple, reduces the power of Byzantine processes, in such a way that a value broadcast only by Byzantine processes is never delivered to the correct processes. This value-oriented broadcast abstraction is then used to solve asynchronous Byzantine binary consensus is asynchronous systems enriched with a common coin. The resulting consensus algorithm is a round-based asynchronous algorithm that has the following noteworthy features.

- The algorithm requires $t < n/3$ and is consequently optimal with respect to t.
- The algorithm uses 2 or 3 communication steps per round (2 steps when the correct processes start a round with the same estimate of the decision value).
- The expected number of rounds to decide is 4.
- The message complexity is $O(n^2)$ messages per round.
- Each message carries its type, a round number plus a single data bit.
- Finally, the algorithm does not require signed messages.

Hence, contrarily to what one could believe from existing asynchronous Byzantine binary consensus algorithms, the formula [quadratic message complexity] $\Rightarrow$ [(use of signatures) $\vee$ $(t < n/5)$] is false. The proposed algorithm shows that $t < n/3$ (as in [9]), quadratic message complexity (as in [5, 9]), and absence of signatures (as in [5, 21]) are not incompatible.

Simplicity is a first class property When designing this algorithm, an important issue was to obtain a *simple* Byzantine consensus algorithm. Our guiding mantra while designing the proposed algorithm was the famous sentence of Einstein "make it as simple as possible, but not simpler". This was not an easy task as simplicity is rarely obtained for free. Let us also remember the following sentence written by Pascal at the end of a letter to a friend "I apologize for having written such a long letter, I had not enough time to write a shorter one". The implication "simple $\Rightarrow$ easy" is rarely true for non-trivial problems [1]. Simplicity requires effort, but is very rewarding. It is a first class scientific property which participates in the beauty of science[1] and makes pedagogy easier.

Roadmap The paper is composed of 5 sections. Section 2 presents the computation model. Section 3 presents and proves correct the binary-value broadcast abstraction, while Section 4 presents and proves correct the consensus algorithm. Finally, Section 5 concludes the paper.

2 Computation Model

Asynchronous processes The system is made up of a finite set Π of $n > 1$ asynchronous sequential processes, namely $\Pi = \{p_1, \ldots, p_n\}$. "Asynchronous" means that each process proceeds at its own pace, which may vary arbitrarily with time, and remains always unknown to the other processes.

Communication network The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. "Asynchronous" means that a message that has been sent is eventually received by its destination process, i.e., there is no bound on message transfer delays. "Reliable" means that the network does not loss, duplicate, modify, or create messages. "Point-to-point" means that there is a bi-directional communication channel between each pair of processes. Hence, when a process receives a message, it can identify its sender.

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

The operation broadcast TAG(m) is a macro-operation which stands for "**for each** $j \in \{1, \ldots, n\}$ send TAG(m) to p_j **end for**".This operation is usually called *unreliable* broadcast (if the sender crashes in the middle of the **for** loop, it is possible that only an arbitrary subset correct processes receives 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 called *faulty*. Otherwise, it is *correct*.

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.

Notation This computation model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$. In the following, this model is both restricted with a constraint on t and enriched with an object providing processes with additional computational power. More precisely, $\mathcal{BZ}\_\mathcal{AS}_{n,t}[n > 3t]$ denotes the model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ where the number of faulty processes is smaller than $n/3$, and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[n > 3t, \text{CC}]$ denotes the model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[n > 3t]$ enriched with the common coin (CC) abstraction as defined in [32].

3 The Binary-Value Broadcast Abstraction

3.1 Binary-value broadcast

Definition This communication abstraction (in short, BV-broadcast) in an all-to-all abstraction that provides the processes with a single operation denoted BV_broadcast(). When a process invokes BV_broadcast TAG(m), we say that it "BV-broadcasts the message TAG(m)". The content of a message m is 0 or 1 (hence the term "binary-value" in the name of this communication abstraction).

[1] To quote Dijkstra: "When we recognize the battle against chaos, mess, and unmastered complexity as one of computing science's major callings, we must admit that "Beauty is our Business"." [14].

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

- BV-Obligation. If at least $(t+1)$ correct processes BV-broadcast the same value v, v is eventually added to the set bin_values_i of each correct process p_i.
- BV-Justification. If p_i is correct and $v \in bin\_values_i$, v has been BV-broadcast by a correct process.
- BV-Uniformity. If a value v is added to the set bin_values_i of a correct process p_i, eventually $v \in bin\_values_j$ at every correct process p_j.
- BV-Termination. Eventually the set bin_values_i of each correct process p_i is not empty.

The following property is an immediate consequence of the previous properties.

Property 1. *Eventually the sets bin_values_i of the correct processes p_i become non-empty and equal, contain all the values broadcast by correct processes and no value broadcast only by Byzantine processes.*

3.2 A BV-broadcast algorithm

A simple algorithm implementing the BV-broadcast abstraction is described in Figure 1. This algorithm is based on a particularly simple "echo" mechanism. Differently from previous echo-based algorithms (e.g., [7, 37]), echo is used here with respect to each value that has been received (whatever the number of processes that broadcast it), and not with respect to each pair composed of a value plus the identity of the process that broadcast this value. In the algorithm of Figure 1, a value entails a single echo, whatever the number of processes that have broadcast this value.

When a process invokes BV_broadcast MSG(v), $v \in \{0, 1\}$, it broadcasts B_VAL(v) to all the processes (line 01). Then, when a process p_i receives (from any process) a message B_VAL(v), (if not yet done) it forwards this message to all the processes (line 03) if it has received the same message from at least $(t + 1)$ different processes (line 02). Moreover, if p_i has received v from at least $(2t + 1)$ different processes, the value v is added to bin_values_i.

```
operation BV_broadcast MSG(v_i) is
(01)  broadcast B_VAL(v_i).

when B_VAL(v) is received
(02)  if (B_VAL(v) received from (t + 1) different processes and B_VAL(v) not yet broadcast)
(03)     then broadcast B_VAL(v)    % a process echoes a value only once %
(04)  end if;
(05)  if (B_VAL(v) received from (2t + 1) different processes)
(06)     then bin_values_i ← bin_values_i ∪ {v}    % local delivery of a value %
(07)  end if.
```

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

Remark It is important to notice that no correct process p_i can know when its set bin_values_i has obtained its final value. (Otherwise, consensus will be directly obtained by directing each process p_i to deterministically extract the same value from bin_values_i). This impossibility is due to the net effect of asynchrony and process failures [17].

3.3 Proof and cost of the algorithm

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

Proof Proof of the BV-Obligation property. Let v be a value such that $(t + 1)$ correct processes invoke BV_broadcast MSG(v). It follows from the predicate of line 02 that each correct process eventually receives these $(t + 1)$ messages, and consequently each correct process broadcasts exactly once the message B_VAL(v) to all the processes. As the channels are reliable, it follows that the predicate of line 05 is eventually satisfied at each correct process, and consequently, the property follows.

Proof of the BV-Justification property. To show this property, we prove that a value BV-broadcast only by faulty processes cannot be added to the set bin_values_i of a correct process p_i. Hence, let us assume that only faulty processes BV-broadcast v. It follows

that a correct process can receive the message B_VAL(v) from at most t different processes. Consequently the predicate of line 02 cannot be satisfied at a correct process. Hence, the predicate of line 05 cannot be satisfied either at a correct process, and the property follows.

Proof of the BV-Uniformity property. If a value v is added to the set bin_values_i of a correct process p_i (local delivery), this process has received B_VAL(v) from at least $(2t+1)$ different processes (line 05), i.e., from at least $(t+1)$ different correct processes. As each of these correct processes has sent this message to all the processes, it follows that the predicate of line 02 is eventually satisfied at each correct process, which consequently broadcasts B_VAL(v) to all. As $n - t \geq 2t + 1$, the predicate of line 05 is then eventually satisfied at each correct process, and the BV-Uniformity property follows.

Proof of the BV-Termination property. As (a) there are at least $(n - t)$ correct processes, (b) each of them invokes BV_broadcast MSG(), (c) $n - t \geq 2t + 1 = (t + 1) + t$, and (d) only 0 and 1 can be BV-broadcast, it follows that there is a value $v \in \{0, 1\}$ that is BV-broadcast by at least $(t + 1)$ correct processes. The proof of the BV-Termination property is then an immediate consequence of the BV-Obligation property. $\square_{Theorem\ 1}$

Cost of the algorithm As far as the cost of the algorithm is concerned, we have the following for each BV-broadcast instance.

- If all correct processes BV-broadcast the same value, the algorithm requires a single communication step. Otherwise, it can require two communication steps.
- Let $c \geq n - t$ be the number of correct processes.

  The correct processes send $c\, n$ messages if they BV-broadcast the same value, and send $2\, c\, n$ messages otherwise.
- In addition to the control tag B_VAL, a message carries a single bit. Hence, message size is constant.

4 The Byzantine Consensus Algorithm

4.1 Byzantine consensus and enriched model

Binary Byzantine consensus The Byzantine consensus problem has been informally stated in the Introduction. Assuming that each correct process p_i proposes a value $v_i \in \{0, 1\}$, each of them has to decide a value such that the following property are satisfied.

- BC-Validity. A decided value was proposed by a correct process.
- BC-Agreement. No two correct processes decide different values.
- BC-One-shot. A correct process decides at most once.
- BC-Termination. Each correct process decides.

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

From binary to multivalued Byzantine consensus Interestingly, asynchronous multivalued Byzantine consensus (i.e., when more than two values can be proposed) can be solved on top of binary Byzantine consensus. Such constructions are described in [13, 30, 36] (see [39] for synchronous systems).

Enriching the basic asynchronous model: Rabin's common coin As indicated in the Introduction, the basic system model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3]$ has to be enriched so that Byzantine consensus can be solved. The additional computational power we consider here is a *common coin* (CC) as defined by Rabin [32]. As already indicated, the corresponding enriched system model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{CC}]$. A common coin can be seen as a global entity that delivers the very same sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to processes, each bit b_r has the value 0 or 1 with probability $1/2$.

More precisely, this oracle provides the processes with a primitive denoted random() that returns a bit each time it is called by a process. In addition to being random, this bit has the following global property: the r^{th} invocation of random() by a correct process p_i returns it the bit b_r. This means that the r^{th}invocations of random() by any pair of correct processes p_i and p_j return them b_r. A common coin is built in such a way that the processes need to cooperate to compute the value of each bit b_r. This is required to prevent Byzantine processes from computing bit values in advance and exploit these values to produce message delivery schedule that would prevent termination[2]. The reader interested in the implementation of such a common coin can consult [2, 9].

[2]As just indicated, this assumption on the common coin computation is due to the fact that Byzantine processes are not prevented from controlling message delivery schedule, thereby rendering random numbers helpless. It would be possible to let Byzantine processes know in advance the sequence of random bits, as long as they cannot control message delivery schedules (this assumption corresponds to the *oblivious scheduler* model [2]).

On randomized consensus When using additional computing power provided by common coins, the consensus termination property can no longer be deterministic. *Randomized consensus* is defined by BC-Validity (Obligation), BC-Agreement, plus the following BC-Termination property [4, 32]: Every non-faulty process decides with probability 1. For round-based algorithms, this termination property is re-stated as follows: For any correct process p_i: $\lim_{r\to+\infty}\big(\text{Probability }[p_i \text{ decides by round } r]\big) = 1$.

4.2 Randomized Byzantine consensus algorithm

Principles and description of the algorithm The consensus algorithm is described in Figure 2. It requires $t < n/3$ and is consequently optimal with respect to the maximal number t of Byzantine processes that can be tolerated. A process p_i invokes propose(v_i) where $v_i \in \{0,1\}$ is the value it proposes. It decides when it executes the statement decide(v) at line 08.

The local variable est_i of a process p_i keeps its current estimate of the decision (initially $est_i = v_i$). The processes proceed by consecutive asynchronous rounds and a BV-broadcast instance is associated with each round. The local variable r_i denotes the current round of process p_i, while the local variable $bin\_values_i[r_i]$ denotes the local read-only variable bin_values_i associated with the BV-broadcast instance used at round r_i.

```
operation propose(v_i)
est_i ← v_i; r_i ← 0;
repeat forever
(01)  r_i ← r_i + 1;
(02)  BV_broadcast EST[r_i](est_i);
(03)  wait until (bin_values_i[r_i] ≠ ∅);
      % bin_values_i[r_i] has not necessarily obtained its final value when the wait statement terminates %
(04)  broadcast AUX[r_i](w) where w ∈ bin_values_i[r_i];
(05)  wait until (∃ a set of (n − t) AUX[r_i](x) messages delivered from distinct processes such that
                  values_i ⊆ bin_values_i[r_i] where values_i is the set of values x carried by these (n − t) messages):
(06)  s ← random();
(07)  if (values_i = {v})  % i.e.,|values_i| = 1 %
(08)    then if (v = s) then decide(v) if not yet done end if;
(09)         est_i ← v
(10)    else est_i ← s
(11)  end if
end repeat.
```

Figure 2: A BV-broadcast-based algorithm implementing binary consensus in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[n > 3t, \text{CC}]$

The behavior of a correct process p_i during a round r_i can be decomposed in three phases.

- Phase 1: lines 01-03. This first phase is an exchange phase. During a round r_i, a process p_i first invokes BV_broadcast EST$[r_i](est_i)$ (line 02) to inform the other processes of the value of its current estimate est_i. This message is tagged EST and associated with the round number r_i (hence the notation EST$[r_i]()$). Then, p_i waits until its underlying read-only BV-broadcast variable $bin\_values_i[r_i]$ is no longer empty (line 03). Due to the BV-Termination property, this eventually happens. When the predicate becomes satisfied, $bin\_values_i[r_i]$ has not yet necessarily its final value, but it contains at least one value $\in \{0,1\}$. Moreover, due to the BV-Justification property, the values in $bin\_values_i[r_i]$ were BV-broadcast by correct processes.
- Phase 2: lines 04-05. The second phase is also an exchange phase during which each correct process p_i invokes broadcast AUX$[r_i](w)$ where w is a value that belongs to $bin\_values_i[r_i]$ (line 04). Let us notice that all the correct processes p_j broadcast a value of their set $bin\_values_j[r_j]$ (i.e., an estimate value of a correct process), while a Byzantine process can broadcast an arbitrary binary value. To summarize, the broadcasts of the second phase inform the other processes of estimate values that have been BV-broadcast by correct processes.

  A process p_i then waits until the predicate of line 05 becomes satisfied. This predicate is used to discard values sent only by Byzantine processes. From an operational point of view, it states that there is a set $values_i$ containing only the values broadcast at line 04 by $(n-t)$ distinct processes, and these values originated from correct processes (which BV-broadcast them at line 02). Said in another way, the set $values_i$ of a correct process p_i cannot contain an estimate value broadcast only by Byzantine processes. Hence, after line 05, we have $values_i \in \{0,1\}$, and for any $v \in values_i$, v is an estimate VB-broadcast by a correct process.
- Phase 3: lines 06-11. This last phase is a local computation phase. A correct process p_i first obtains the common coin value s associated with the current round (line 06).
  - If $|values_i| = 2$, both the value 0 and the value 1 are estimate values of correct processes. In this cases, p_i adopts the value s of the common coin (line 10).

– If $|values_i| = 1$, p_i decides v (the single value present in $values_i$) if additionally $s = v$ (line 08). Otherwise it adopts v as its new estimate (line 09).

The statement decide() used at line08 allows the invoking process p_i to decide but does not stop its execution. A process executes round forever. This facilitates the description and the understanding of the algorithm.

4.3 Proof and cost of the algorithm

Notation Let p_i be a correct process. $values_i^r$ denotes the value of the local variable $values_i$ which makes satisfied the predicate of line 05 at round r.

Lemma 1. *Let $n > 3t$. Consider the situation where, at the beginning of a round r, all the non-faulty processes have the same estimate value v. These processes will never change their estimates, thereafter.*

Proof If all the correct processes (which are at least $n - t > t + 1$) have the same estimate value v at beginning of a round r, they all BV-broadcast EST$[r](v)$ at line 02. It follows that the set $bin\_values_i[r]$ of each correct process p_i contains v (BV-Obligation property) and only v (BV-Justification property). Hence, due to the lines 04, 07, and 09, the estimate est_i of any correct process p_i is set to v, which concludes the proof of the lemma. $\square_{Lemma\ 1}$

Lemma 2. *Let $n > 3t$, and p_i and p_j be two correct processes.* $(values_i^r = \{v\}) \wedge (values_j^r = \{w\}) \Rightarrow (v = w)$.

Proof Let us consider a correct process p_i and assume that $value_i^r = \{v\}$. It follows from the predicate of line 05 that p_i has received the same message AUX$[r](v)$ from at least $(n - t)$ different processes. As at most t processes can be Byzantine, it follows that p_i has received AUX$[r](v)$ from at least $(n - 2t)$ different correct processes, i.e., as $n - 2t \geq t + 1$, from at least $(t + 1)$ correct processes.

Let us consider another correct process p_j such that $value_j^r = \{w\}$. This process received the message AUX$[r](w)$ from at least $(n-t)$ different processes. As $(n-t)+(t+1) > n$, it follows that one correct process p_x sent AUX$[r](v)$ to p_i and AUX$[r](w)$ to p_i. As p_x is correct, it sent the same message to all the processes. Hence $v = w$, which concludes the proof of the lemma. $\square_{Lemma\ 2}$

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

Proof Let us consider the round $r = 1$. It follows from (a) the BV-Justification property of the BV-broadcast (line 02), (b) the wait statement at line 03, and (c) the broadcast by each correct process p_j of a value taken from its set $bin\_values_j[1]$, that, the set $values_i$ computed at line 05 by any correct process p_i contains only estimate values of correct processes. Then, if $values_i = \{v\}$ and v is equal to the value s of the common coin, v is decided. Be the value v decided or not, p_i adopts it as new estimate (line 09). If $values_i = \{0, 1\}$, both values have been proposed by correct processes and p_i adopts the one defined by the common coin as its new estimate (line 10). In all cases, the estimate value of a correct process is equal to a proposed value. Then the same reasoning applies to all other rounds, from which it follows that a decided value is an estimate value that was proposed by a correct process. $\square_{Lemma\ 3}$

Lemma 4. *No two non-faulty processes decide different values.*

Proof Let r be the first round at which processes decide. If two correct processes p_i and p_j decide at round r, they decide the same value, namely, the value of the common coin associated with round r, and update their estimates to the value of the common coin.

Moreover, due to Lemma 2, if p_i decides v during round r, there is no correct process p_j such that $value_i^r = \{w\}$, with $w \neq v$. Hence, if a process p_j does not decide during r, we necessarily have $values_j^r = \{v, w\} = \{0, 1\}$. It follows that such a process p_j executes line 10, and assigns the value of the common coin to its estimate est_j.

Hence, at the beginning of round $(r + 1)$, the estimates of all the correct processes are equal to the common coin, which is itself equal to the decided value v. It then follows from Lemma 1 that they keep this value forever. As a decided value is a an estimate value, only v can be decided. $\square_{Lemma\ 4}$

Lemma 5. *Each non-faulty process decides with probability* 1.

Proof Let us first prove that no correct process remains blocked forever during a round r. There are two **wait** statements. Due to the BV-Termination property, no correct process can block forever at line 03. To show that no correct process can block forever at line 05, we have to show that the predicate of line 05 becomes eventually true at every correct process p_i. This follows from the

following observations: during round r, (a) the set $bin\_values_i[r]$ of each correct process contains only values BV-broadcast by correct processes (BV-Justification), and eventually the sets of all the correct processes are equal (BV-Uniformity); (b) each of the at least $(n-t)$ correct processes p_i broadcasts a message AUX$[r](w)$ such that $w \in bin\_values_i[r]$; (c) each of these messages is eventually received by each correct process.

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

Assuming the claim holds, it follows from Lemma 1 that all the correct processes p_i keep their estimate value $est_i = v$ and consequently the predicate $values_i = \{v\}$ at line 07 is true at every round. Due to the common coin CC, it follows that, with probability 1, there is eventually a round in which random() outputs v. Then, the predicates of the lines 07 and 08 evaluate to true, and all the correct processes decide.

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

- If all the correct processes execute line 10, they all adopt the value of the common coin at the end of round r, and the claim directly follows.
- Similarly, if all the correct processes execute line 09, they adopt the same value v as their new estimate, and the claim follows.
- The third case is when some correct processes execute line 09 and adopt the same value v, while others execute line 10 and adopt the same value s.

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

$\square_{Lemma\ 5}$

Theorem 2. *The algorithm described in Figure 2 solves the randomized binary consensus problem in the system model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{CC}]$.*

Proof BC-Validity follows from Lemma 3. BC-Agreement follows from Lemma 4. BC-One-shot follows from line 08. BC-Termination follows from Lemma 5. $\square_{Theorem\ 2}$

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

Proof As indicated in the proof of Lemma 5, termination is obtained in two phases. First, all the correct processes must adopt the same value v. Second, the outcome of the common coin has to be the same as the commonly adopted value v.

It follows from the proof of Lemma 5 that there is only one situation in which the correct processes do not adopt the same value. This is when the predicate of line 07 is satisfied for a subset of correct processes and not for the other correct processes. Thus, the expected number of rounds for this to happen is 2. As for the second phase, here again, the probability that the value output by the common coin is the same as the value held by all the correct processes is $1/2$. Thus, the expected time for this to occur is also 2. Combining the two phases, the expected termination time is 4 rounds (i.e., a small constant). $\square_{Theorem\ 3}$

Cost of the algorithm As far as the cost of the algorithm is concerned, we have the following, where $c \geq n-t$ denotes the number of correct processes.

- If the correct processes propose the same value, each round requires two communication steps (one in the BV-broadcast and one broadcast), and the expected number of rounds to decide is two. Moreover, the total number of messages sent by correct processes is then $2\ c\ n$.
- If the correct processes propose different values, each round requires three communication steps (two in the BV-broadcast and one broadcast), and the expected number of rounds to decide is four. Moreover, the total number of messages sent by the correct processes is then $4\ c\ n$ per round.
- In addition to a round number, both a message EST$[r]()$ and a message AUX$[r]()$ sent by a correct process carry a single bit. An underlying message B_VAL() has to carry a round number and a bit.
- The total number of bits exchanged by the correct processes is $O(n^2 r \log r)$ where r is the number of rounds executed by the correct processes. Hence, the expected bit complexity is $O(n^2)$.

5 Conclusion

This paper has presented a new consensus algorithm suited to asynchronous systems composed of n processes, and where up to $t < n/3$ processes may have a Byzantine behavior. This algorithm relies on Rabin's common coin and an underlying binary-value broadcast algorithm which guarantees that a value broadcast only by Byzantine processes is never delivered to the correct processes. In addition to being t-resilient optimal, the algorithm, which is round-based and signature-free, uses two or three communication steps per round (this depends on the estimate values of the correct processes at the beginning of a round), and $O(n^2)$ messages per rounds. Moreover, each message carries a round number and a single bit, and the expected number of rounds to decide is four. Finally, as claimed in the Introduction, and in addition to its efficiency-related properties, a very important first class property of the proposed algorithm lies in its *design simplicity*.

References

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

[2] Aspnes J., Lower bounds for distributed coin flipping and randomized consensus. *Journal of the ACM*, 45(3):415-450, 1998.

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

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

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

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

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

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

[9] Cachin Ch., Kursawe K., and Shoup V., Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. *Proc. 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 123-132, 2000.

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

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

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

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

[14] Dijkstra E.W., Some beautiful arguments using mathematical induction. *Algorithmica*, 13(1):1-8, 1980.

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

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

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

[24] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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