A New Insight into Local Coin-Based Randomized Consensus

Achour Mostefaoui, Matthieu Perrin, Michel Raynal

A New Insight into Local Coin-Based Randomized Consensus

Achour Mostéfaoui[†] Matthieu Perrin[†] Michel Raynal[⋆,‡]
[†] LS2N, Université de Nantes, 44322 Nantes Cedex, France
[⋆] IRISA, Université de Rennes 35042 Rennes Cedex, France
[‡] Department of Computing, Polytechnic University of Hong Kong
Achour.Mostefaoui@univ-nantes.fr Matthieu.Perrin@univ-nantes.fr raynal@irisa.fr

This submission is a regular paper.

***Abstract*—This paper presents a binary randomized consensus algorithm for n-process asynchronous message-passing systems in which (1) up to $t < n/2$ processes may crash, and (2) an asynchrony adversary can read the content of the messages and re-order their deliveries according to their values.**

In addition to its simplicity, the main property of this algorithm lies in the fact it allows the processes to decide in a constant number of rounds when $t \in O(\sqrt{n})$. To attain this goal the algorithm combines Ben-Or's randomized consensus algorithm (PODC 1983) and the condition-based approach to solve consensus (JACM 2003). (Among other points, if the most frequent proposed value appears more than t times than the other value, decision requires three communication steps.) An improvement of the proposed algorithm is given for the case where $t < n/4$.

We conjecture that if $t > O(\sqrt{n})$ it is no longer possible to implement a randomized consensus algorithm ensuring a constant number of communication steps despite unfair channels.

***Keywords*-Binary consensus, Condition, Local coin, Message-passing asynchronous system, Process crash failure,**

I. Introduction

Consensus in crash-prone message-passing systems: Consensus is one of the most fundamental problems (from both practical and theoretical point of views) of fault-tolerant distributed computing. This paper considers n-process systems, where communication is by asynchronous message-passing and where up to t processes may crash.

In a consensus instance, each process is assumed to invoke once the operation propose(), that allows it to propose a value (called *proposed* value), and returns it a value (called *decided* value). If only 0 and 1 can be proposed, consensus is *binary*, otherwise it is *multivalued*. In the context we consider, consensus is defined by the following properties.

- C-Validity. A decided value is a proposed value.
- C-Agreement. No two processes decide different values.
- C-Termination. If a process does not crash, it decides a value.

C-Validity relates the output to the inputs, C-Agreement relates the outputs of all processes, and C-Termination is the natural liveness property.

One of the most celebrated results of fault-tolerant distributed computing is the famous FLP impossibility result [12], namely, there is no deterministic consensus algorithm in the asynchronous message-passing model in which even a single process may crash ($t = 1$).

Circumventing FLP - enriched models: Several approaches have been proposed to circumvent the previous impossibility. One consists in enriching the system with additional *synchrony assumptions* [10], [11]. Another approach consists in enriching the system with additional devices providing the processes with information on failures. This is the *failure-detector*-based approach [8], for which is known the weakest failure detector (named Ω), i.e., the weakest information on failures that allows consensus to be solved despite asynchrony and $t < n/2$ [9]. A third approach, named *condition*-based approach, consists in reducing the space of input vectors that the processes can collectively propose (each entry of the vector containing the value proposed by a process) [16]. For example, it is possible to solve consensus under the hypothesis that more than $\frac{3}{4}$ of the processes propose the same value (which is called the C_{mfv} condition).

Yet another approach that has been proposed consists in enriching each process with a random number generator, and weakening the termination property as follows. C-Termination becomes (where R stands for "randomized"):

- RC-Termination. If a process does not crash, it decides with probability 1.

Randomized binary consensus algorithms belong to the family of Las Vegas randomized algorithms, which means that (1) their safety properties are always ensured, and (2) while it is technically possible that an execution does not terminate, it requires an infinite sequence of unlucky random number which makes termination certain in practice. Algorithms have been proposed since the early eighties [6], [23]. A more extensive presentation of these algorithms can be found in [24].

Hybrid consensus algorithms have also been designed, which use additional assumptions of several kinds. For example, algorithms based on failure detectors and randomization are presented in [1], [21], and algorithms based on failure detectors and conditions are described in [17]. The advantage of hybrid algorithms lies in their greater assumption coverage [22].

In all consensus algorithms, the processes execute a sequence of asynchronous rounds, each round including one

	B83	AW04	AAKS14	Algo1	Algo2
t-resilience	$t < n/2$	$t < n/2$	$t < n/2$	$t < n/2$	$t < n/4$
Com. steps for the common coin	0	3	$O(n(\log n)^3)$	0	0
Bias of the common coin	$1/2^n$	$1/4$	$1/4$	$1/2^n$	$1/2^n$
Message re-ordering	allowed	forbidden	allowed	allowed	allowed
Expected number of com. steps	$O(2^n)$	36	$O(n(\log n)^3)$	$\frac{3}{1-\mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right)}$	$\frac{2}{1-\mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right)}$
When $\alpha = 1$ and $t < \sqrt{n}$			$O(t(\log t)^3)$	$\simeq 9$	$\simeq 6$

Table I: Comparison and content of the paper

or more communication steps. Let us also notice that it is possible to design multivalued consensus algorithms on top of binary consensus algorithms despite asynchrony and process crashes (with a constant or bounded number of communication steps) [19], [20], [25].

Binary coin-based consensus algorithms: As defined in [5], a *common coin with bias ρ*, is a coin shared by all processes, that returns the same value $v \in \{0, 1\}$ to all of them with probability at least ρ. This means it can return different values to different processes with probability at most $1 - 2\rho$. Hence, a common coin with bias $1/2$ is perfect in the sense it returns the same value to all processes.

A coin is *local* to a process, if it is used only by this process, and returns the value 0 with some probability pb and the value 1 with probability $1 - pb$, where pb is known. Hence, n local coins such that $pb = 1/2$, provides us, without additional computation or communication –i.e. for free–, with a common coin whose bias is $\rho = \frac{1}{2^n}$.

In shared memory systems, the step complexity for a process is the number of its own steps. The step complexity of a program is the total number of steps taken by all the processes. In message-passing systems, a process depends on messages sent to and received from the other processes. A communication steps represents the time between the sending and the receiving of a message. One can count these steps even though the considered system is asynchronous. For instance, if a process broadcasts a message and waits for a given number of responses, the time complexity would be 2 (one round-trip). In message-passing consensus algorithms, once a process has terminated its execution, it usually broadcasts a message containing the decided value, allowing other processes to terminate within only one more communication step. We define the step complexity of an execution as the number of communication steps taken by the first process that terminated.

Attiya and Censor [4] proved the total step complexity of randomized consensus is $O(n^2)$ in an asynchronous shared memory system. There is abundant literature on common coins in shared memory but the lower bound on time complexity cannot be directly translated to message-passing. By the pigeonhole theorem, such a lower bound translates to $\Omega(n)$ communication steps. The closest known message-passing algorithm to this bound is the one presented in [2] that has a step complexity of $O(n \log^3(n))$.

The time complexity of a consensus algorithm built on top of a common coin is inversely proportional to the bias of the common coin when the number t of possible failures is high. The algorithms proposed in [2], [4] build a common coin with a very small bias. This induces big constants (around 10^5) in the step complexity which makes them impractical. Hence the importance of having a constant bias or a "small" t. As shown in [5], it is possible at the price of some communication steps and assumptions on the asynchrony adversary to build a common coin with known bias from local coins. On the other hand, the implementation of common coins with a constant bias has been extensively studied in the shared memory model [3], [4] for both the crash and the Byzantine failure models. The most efficient translation into the message-passing crash-failure model of these results [2] leads to a time complexity in $O(n(\log n)^3)$ when $t < n/2$ (the implementation needs at least $O(n^2)$ local coin flips). In this paper we focus on the case where a constant number of communication steps can be reached even when the asynchrony adversary can re-order messages according to their content. Let us call AAKS14 any common-coin based consensus algorithm used with the common coin of [2]. Such an algorithm needs $\Omega(n(\log n)^3)$ communication steps as each round of the algorithm needs a common coin flip. Let us also consider the randomized binary message-passing consensus algorithms presented in [6] (denoted B83), and in [5] (denoted AW04). We have the following (see also Table I).

- B83 uses local coins only, and its expected number of communication steps is 2^n. It allows the asynchrony adversary to re-order message receptions according to their content. (This is due to the fact that, as it is based on local coins only, any message re-ordering strategy yields the same result when the processes obtain the same value from their local coins.)
- According to its authors, AW04 is a simplified version of an algorithm proposed in [7] suited to Byzantine process failures. It considers local coins where the probability that the output is 0 is $pb = 1/n$, and consequently the probability that the output is 1 is $1 - 1/n$, from which it constructs a common coin whose bias is $1/4$, i.e., this common coin is such that both the probability to output 0 and the probability to output 1 are $\geq 1/4$. This construction requires three communication steps. Then, using an appropriate communication pattern (called CORE), AW04 implements binary consensus with an

expected number of communication steps equal to 36 (the expected number of rounds is $1/\rho = 4$, each round involving nine communication steps). Moreover, while the adversary can re-order message receptions, it is not allowed to read their content (re-ordering messages based on their content would allow the adversary to control the output of the common coin, resulting in an exponential number of communication steps).

These algorithms strive to produce a round in which the vector defined by the current estimates of the decision value of the processes is the vector $[0, \dots, 0]$ or the vector $[1, \dots, 1]$. When this is attained, decision trivially follows. These two vectors can be considered as *fixed points* (or attractors), and the algorithms have to attain one of them. To this end, they use random numbers so that the path defined by the sequence of the vectors of the decision current estimates associated with each round can be seen as a "distributed random walk" ending in $[0, \dots, 0]$ or $[1, \dots, 1]$.

Content of the paper: This paper presents a constant expected time consensus algorithm based on local coins. It combines the condition-based approach with randomization to obtain a hybrid binary consensus algorithm, denoted Algo1 in Table I that is based on both a condition and local coins. In the condition-based approach, all vectors contained in the condition are *fixed points* in which the random walk terminates. Hence, the "termination space" contains all vectors that respect the condition, which is much bigger than the set $\{[0, \dots, 0], [1, \dots, 1]\}$.

This algorithm, which assumes $t < n/2$ (a necessary condition in asynchronous message-passing), is pretty simple. Each round is made up of 3 communication steps, the first one tries to benefit from the condition, the second one can be seen as a "cleaning" step, while the third one is a "decision/adoption/random help" step. If the input vector belongs to the condition, decision is obtained in 3 communication steps (and randomization is unnecessary).When $t < \alpha\sqrt{n}$, with $0 < \alpha < \sqrt{n}$ (a reasonable case in practice as, while failures can happen, they are not frequent), the expected number of communication steps to decide is $3\big(1 - \mathsf{erf}\big(\frac{\alpha}{\sqrt{2}}\big)\big)^{-1}$, where erf denotes the Gauss error function. For $\alpha = 1$, this number is smaller than 10, divided in 3.2 rounds of three communication steps. For $\alpha \leq 0.67$ (e.g. $n = 4$ and $t = 1$), the algorithm needs less than the 2 rounds required in average by the best known algorithm using a common coin [15] regardless the number of faults.

It has already been observed that in practice when failures are rare, local coin protocols can terminate within few rounds [14]. Especially, when $t = \sqrt{n}$, a common coin can be built whatever is the message scheduler. In the present paper, this becomes a particular case of the proposed hybrid algorithm. Moreover, due to the fact that there is no explicit common coin construction there is a gain in the complexity of the consensus algorithm as there is no protocol stacking as it can be seen in [5] in Table I.

A main property of the algorithm lies in the strong asynchrony adversary it tolerates, namely the adversary controls all aspects regarding asynchrony and failure. In particular, it is allowed to read messages and re-order their reception according to their content. The only restrictions of the adversary are that (1) processes are bound to respect their protocole and (2) the adversary cannot predict nor chose the values of the local coins drawn by the processes. B83 copes with this adversary but has an expected number of communication steps which is exponential, while AW04 has a constant expected number of communication steps but does not tolerate the previous strong adversary. The other contribution consists of a second algorithm (denoted Algo2), which converges faster, but requires a stronger assumption on t, namely $t < n/4$.

As far as we know, these are the first randomized consensus algorithms that need so few communication steps despite content-based message re-ordering when $t < O(\sqrt{n})$. The problem of designing a randomized consensus algorithm whose expected number of communication steps is constant (or sublinear) when $t > O(\sqrt{n})$ and the adversary can re-order messages according to their content is still open.

Let us notice that, if the adversary is not allowed to read the content of the messages, it is possible to apply the condition-based approach to AW04, and obtain an algorithm with a reduced number of communication steps.

Roadmap: The paper is composed of 6 sections. Section II presents the distributed computation model and the condition-based approach. Section III presents the binary condition-helped randomized consensus algorithm Algo1, and Section IV proves the bound on its expected number of communication steps to decide. Section V presents Algo2, which reduces the number of communication steps per round from three to two when $t < n/4$. Finally, Section VI concludes the paper.

II. Distributed Computing Model and the Condition-based Approach

A. Basic Distributed Computing Model

As the constraint $t < n/2$ is a necessary requirement to solve consensus in asynchronous message-passing systems, the distributed computing model described below is denoted $CAMP_{n,t}[t < n/2]$[1].

Process model: The computing model is composed of a set of n sequential processes denoted p_1, ..., p_n. Each process is asynchronous which means that it proceeds at its own speed, which can be arbitrary and remains always unknown to the other processes.

A process may halt prematurely (crash failure), but executes correctly its local algorithm until it crashes (if it ever does). The model parameter t denotes the maximal number of processes that may crash in a run. A process that

[1] $CAMP$ stands for Crash-tolerant Asynchronous Message Passing

crashes in a run is said to be *faulty*. Otherwise, it is *correct* or *non-faulty*. Given a run, $Correct$ will denote the set of processes that are correct in this run.

Communication: Each pair of processes communicate by sending and receiving messages through a bidirectional channel. Hence, the communication network is a complete network: any process p_i can directly send a message to any process p_j (including itself). A process p_i invokes the operation "send $\text{TYPE}(m)$ to p_j" to send to p_j the message m, whose type is TYPE. The operation "receive()" allows p_i to receive a message.

The unreliable macro-operation broadcast $\text{TYPE}(m)$ is a shortcut for "**for each** $j \in \{1, ..., n\}$ **do** "send $\text{TYPE}(m)$ to p_j". If the invoking process crashes when it executes it, an arbitrary subset of processes receive the message.

Each channel is reliable (neither loss, corruption, nor creation of messages), not necessarily first-in/first-out, and asynchronous (while the transit time of each message is finite, there is no upper bound on message transit times).

Asynchrony adversary: It is not assumed any restriction on the ability of the adversary controlling message asynchrony to read their content, and use this information to re-order message receptions at any process.

B. Enriching the Basic Computing Model with Local Coins

Local coin: The computability power of each process p_i is enriched with a local function denoted random(). When a process p_i invokes it, it obtains either the value 0 or the value 1, each with probability 0.5. The local functions random() are independent from the others, which means they implement for free a common coin with bias $1/2^n$.
Notation The model $CAMP_{n,t}[t < n/2]$ enriched with local coins is denoted $CAMP_{n,t}[t < n/2, \text{LC}]$.

C. The Condition-based Approach

The condition-based approach to solve consensus in failure-prone asynchronous systems was introduced in [16].

Definition: Let an *input vector* be a vector $I[1..n]$ such that $I[i]$ contains the value proposed by p_i. In the case of binary consensus there are 2^n possible input vectors. Let us call *condition* any subset of input vectors.

The *condition-based* approach to solve asynchronous consensus consists in identifying subsets of input vectors, such that if the actual input vector belongs to the considered subset, consensus can be solved. Such a subset is called a *t-legal condition*. If adding another vector to the condition makes it not t-legal, the condition is *maximal*.

Let $\mathsf{dist}(I1, I2)$ be the Hamming distance between the input vectors $I1$ and $I2$ (number of entries in which they differ), and $\mathsf{nb}(a, I)$ be the number of entries of I whose value is a. It is shown in [16] that a subset of input vectors C is a t-legal condition if, and only if, there is a function $h() : C \mapsto \mathcal{V}$ (where $\mathcal{V}$ is the set of values that can be proposed, in our case $\mathcal{V} = \{0, 1\}$) such that

- $\forall I \in C : \ \mathsf{nb}(h(I), I) > t$,
- $\forall I1, I2 \in C : \ \big(h(I1) \neq h(I2)\big) \Rightarrow \big(\mathsf{dist}(I1, I2) > t\big)$.

The intuition that underlies the condition-based approach is the following. Given a condition C, each of its input vectors allows a proposed value to be selected as the decided value. This value is extracted from the input vector with the function $h()$, i.e., $h(I)$ is the value decided from I. When looking at the definition of t-legality, the first item states that, to be decided, a value must be "present enough" in the input vector, while the second item states that input vectors from which different values are decided must be "far enough apart" to prevent ambiguity. A relation linking the consensus condition-based approach and error-correcting code was established in [13].

The condition: C_{mfv}: The condition used in this paper is the condition C_{mfv} introduced in [16]. This condition favors the most present value. Given a vector I, let $\mathsf{first}(I)$ be its most frequent value, and $\mathsf{second}(I)$ its second most frequent value. C_{mfv} is defined as follows (by definition $\mathsf{nb}(\mathsf{second}(I), I) = 0$ when I contains a single value).

$$C_{mfv} = \{I \in \mathcal{V}^n \text{ s.t. } \big(\mathsf{nb}(\mathsf{first}(I), I) - \mathsf{nb}(\mathsf{second}(I), I)\big) > t\}$$

Symmetric condition: The condition C_{mfv} is symmetric in the sense that it does not depend on the specific values in a vector, it depends only on their occurrence numbers. In the case of binary consensus, changing all 0 in 1 and all 1 in 0 in a vector I produces a vector I' that belongs to the condition.

As an example, let us consider the condition C_{max}, where the greatest value appearing in an input vector I must appear more than t times, namely $C_{max} = \{I \in \mathcal{V}^n \text{ such that } \mathsf{nb}(\max(I), I) > t\}$. This condition is t-legal but not symmetric.
Notation: In the rest of the paper, C_{mfv} is abbreviated as C.

D. Characterization of C_{mfv} for Binary Input Values

Let us assume that the vector I only contains values 0 and 1. Hence, $\mathsf{nb}(0, I) + \mathsf{nb}(1, I) = n$.

Theorem 1: $(I \in C_{mfv}) \Leftrightarrow \big(\sum_{i=1}^{n} I[i] < \frac{n-t}{2}\big) \vee \big(\sum_{i=1}^{n} I[i] > \frac{n+t}{2}\big)$.

Proof: Direction "⇒".

- Case $\mathsf{nb}(1, I) - \mathsf{nb}(0, I) > t)$. Replacing $\mathsf{nb}(0, I)$ by $n - \mathsf{nb}(1, I)$, gives $\mathsf{nb}(1, I) - n + \mathsf{nb}(1, I) > t$, i.e., $2 \times \mathsf{nb}(1, I) > n + t$.
- Case $\mathsf{nb}(0, I) - \mathsf{nb}(1, I) > t)$. Replacing $\mathsf{nb}(0, I)$ by $n - \mathsf{nb}(1, I)$, gives $n - \mathsf{nb}(1, I) - \mathsf{nb}(1, I) > t$, i.e., $n - t > 2 \times \mathsf{nb}(1, I)$.

Direction "⇐". As $\mathsf{nb}(0, I) + \mathsf{nb}(1, I) = n$, the previous reasoning works in both directions. ■

```
operation propose $(v_i)$ is % $v_i \in \{0,1\}$ %
    $est1_i \leftarrow v_i$; $r_i \leftarrow 0$;
    while true do
    $r_i \leftarrow r_i + 1$;
—— % Phase 1: Exchange of current estimate values ————————————————-
      broadcast EST $(r_i, est_i)$;
      wait $\big($EST $(r_i, -)$ received from $(n-t)$ processes$\big)$;
      for $x \in \{0,1\}$ do $nb_i[x] \leftarrow$ number of messages (EST $(r_i, x)$ received) end for:
      if $(nb_i[1] \geq nb_i[0])$ then $aux1_i \leftarrow 1$ else $aux1_i \leftarrow 0$ end if;
———— % Here: $P1(r) \overset{def}{=} \big((est(r-1) \in C) \Rightarrow (\forall\ i,j:\ aux1(r)[i] = aux1(r)[j])\big)$
—— % Phase 2: Exchange of current $aux1$ values ————————————————
      broadcast AUX1 $(r_i, aux1_i)$;
      wait $\big($AUX1 $(r_i, -)$ received from $(n-t)$ processes $\big)$;
      $rec2_i \leftarrow$ multiset of the values $aux1$ received;
      if $\big(rec2_i = \{v, v, \ldots, v\}\big)$ then $aux2_i \leftarrow v$ else $aux2_i \leftarrow \bot$ end if;
———— % Here: $P2(r) \overset{def}{=} \big(\forall\ i,j:\ \big((aux2(r)[i] = v \neq \bot) \Rightarrow (aux2(r)[j] \in \{v, \bot\})\big)\big)$
—— % Phase 3: Try to decide a value from the $aux2$ values (decide/adopt/random) ————————
      broadcast AUX2 $(r_i, aux2_i)$;
      wait (AUX2 $(r_i, -)$ received from $(n-t)$ processes);
      $rec3_i \leftarrow$ multiset of the values $aux2$ received;   % The values in $rec3_i$ are from either $\{0, \bot\}$, or $\{1, \bot\}$
———— % Here: $P3(r) \overset{def}{=} \forall i,j : i \neq j : (rec3_i(r) = \{v, \ldots, v\}$ where $v \neq \bot)$ XOR $(rec3_j(r) = \{\bot, \ldots, \bot\})$
      case $(\exists v \neq \bot : \mathsf{nb}(v, rec3_i) > t)$            do broadcast DECIDE$(r_i, v)$; return$(v)$ % decision
           $(\exists v \neq \bot : 0 < \mathsf{nb}(v, rec3_i) \leq t)$    do $est_i \leftarrow v$                        % adoption
           otherwise                                         do $est_i \leftarrow$ random()                  % random help
      end case
    end while.
```

Algorithm 1: Condition-helped binary consensus in the model $CAMP_{n,t}[t < n/2, \text{LC}]$ (code for p_i)

III. A Simple Consensus Algorithm Based on a Condition and Local Coins

A. Binary Consensus in $CAMP_{n,t}[t < n/2, \text{LC}]$

Algorithm 1 (denoted Algo1 in Table I) is a simple asynchronous binary consensus algorithm for the randomized model $CAMP_{n,t}[t < n/2, \text{LC}]$. It is a round-based algorithm, each round being composed of three communication phases (each including an all-to-all communication step). As indicated in the introduction, Algo1 accepts any of 2^n possible input vectors, and terminates in one round when the input vector I belongs to the condition. Otherwise, Algo1 strives -with the help of randomization- to produce a vector of decision estimates that belongs to the condition.

Local variables at a process: Each process p_i manages the following local variables.

- r_i: current round number.
- est_i: current local estimate of the decision value.
- $nb_i[x]$, where $x \in \{0,1\}$: number of messages $\text{EST}(r_i, x)$ received during the current round.
- $aux1_i$ ($aux2_i$): auxiliary variable used in the first (second) phase of each round. They contain either a proposed value or the default value $\bot$.
- $rec2_i$ ($rec3_i$): multiset of the values received during the second (third) phase of the current round. (A multiset –also called bag– is a "set" that may contain several copies of a same value. As an example, while $\{a, a, b, c, c, c\}$ and $\{a, b, c\}$ are the same set, they are different multisets.)

  The function $\mathsf{nb}(v, rec3_i)$ returns the occurrence number of v in the multiset $rec3_i$.

Notion of a ghost execution: Let us consider an execution E of the algorithm, in which P is the set of processes that decide and Q is the set of processes that crash before deciding. We associate with E a *ghost execution*[2] E' in which (i) no process crashes nor terminates, and (ii) the processes decide the same value v as in E. As E' is an extension of E, all safety properties verified by E' are also verifies by E. Thanks to this property, ghost executions allow us to reason only on executions where no crash occurs, assuming correct processes terminate, which significantly simplifies the explanations in this section and Section IV.

The ghost execution E' of E is an extension of E defined as follows, where τ is a finite time after which all the processes that decide in E, have decided.

- Let $p_i \in P$. Process p_i behaves as in E until it decides, and, after it has decided, pauses until time τ.
- Let $p_i \in Q$. Process p_i behaves as in E until its crash, and then pauses until time τ.
- After time τ, all the processes are resumed, and execute forever.

Additional notations: Considering a global observer point of view, the following notations are used in this section and Section IV.

- $aux1(r)$ and $aux2(r)$: vectors associated with round r, such that $aux1(r)[i]$ (resp., $aux2(r)[i]$) is the current

[2] This is similar to the notion of a ghost variable in program verification.

value of $aux1_i$ (resp., $aux2_i$) at round r in E'.

- $rec3_i(r)$: value of the multiset $rec3_i$ after the assignment at line 14 of round r in E'.
- $est(r)$: vector defined by the estimate values computed at the end of round r in E' (lines 16 or 17). Moreover, let $est(0)$ denote the input vector I.

Process behavior: As already indicated, from a structural point of view, each round is composed of three phases, each including a communication step. An initial phase, which strives to exploit the condition C if the input vector $est(r)$ associated with the current round belongs to the condition, is followed by two phases which are close to the ones used in the B83 algorithm [6], and the ones used in the failure detector-based algorithm presented in [18]. The effect of each phase is captured by a round invariant (denoted $P1(r)$, $P2(r)$, and $P3(r)$ in the following) that form the main lemmas to prove correctness of the algorithm (Properties 2 to 4).

- Phase 1 (lines 4-7).
  A process p_i first broadcasts its current estimate (message $\text{EST}(r_i, est_i)$, and waits until it has received a message $\text{EST}(r_i, -)$ from at least $(n-t)$ processes. Then, it determines the most often received value $x \in \{0,1\}$ in the current round, and assigns it to $aux1_i$ (if both values are equally received, the value 1 is arbitrarily selected). At this point of round r, we have the following global property $P1(r) \stackrel{def}{=}$

  $$(est(r-1) \in C) \Rightarrow (\forall\, i,j:\ aux1(r)[i] = aux1(r)[j])$$

- Phase 2 (lines 8-11).
  The all-to-all communication pattern, is the same in the three phases. The processes exchange the values of their variables $aux1_i$, which (due to $(P1(r_i))$ are identical if the input vector of the estimates at the beginning of the round belongs to the condition C.
  Then, if a process p_i receives the same value $v \in \{0,1\}$ from $(n-t)$ processes, it assigns it to $aux2_i$. Otherwise, it assigns the default value $\perp$ to $aux2_i$. The meaning of $aux2_i = v \neq \perp$ is "p_i champions v to be decided"; The meaning of $aux2_i = \perp$ is "p_i has not enough information to champion a value". At this point of a round r, we have the following global property $P2(r) \stackrel{def}{=}$

  $$\forall\, i,j:\ (aux2(r)[i] = v \neq \perp) \Rightarrow (aux2(r)[j] \in \{v, \perp\})$$

- Phase 3 (lines 12-18).
  After the exchange of their $aux2_i$ values, each process p_i stores the values it has received in its multiset $rec3_i$. Then, it strives to decide, without compromising consensus agreement. To this end it does the following.
  - If $rec3_i$ contains more than t copies of a non-$\perp$ value v, p_i decides it by invoking $\mathsf{return}(v)$ (line 15). However, before deciding p_i broadcasts a message $\text{DEC}(r_i, v)$. This message is interpreted by receivers as a digest containing three messages: $\text{EST}(r_i{+}1, v)$, $\text{AUX1}(r_i{+}1, v)$, and $\text{AUX2}(r_i{+}1, v)$. This is required to prevent a process p_j from waiting forever messages that will never be sent by p_i.
  - If $rec3_i$ contains a non-$\perp$ value v which appears at most t times, p_i adopts it as its new estimate (line 16).
  - Finally, if $rec3_i$ contains only $\perp$, p_i invokes $\mathsf{random}()$ and considers the value returned as its new estimate (line 17).

At this point of a round r, we have the following global property (mutex), where $v \neq \perp$: $P3(r) \stackrel{def}{=} \forall i,j, i \neq j$:

$$(rec3_i(r) = \{v, \ldots, v\}) \oplus (rec3_j(r) = \{\perp, \ldots, \perp\})$$

B. Proof of the Algorithm

Property 2: $P1(r)$:
$(est(r-1) \in C) \Rightarrow (\forall\ i,j:\ aux1(r)(i) = aux1(r)[j])$.

Proof: If the vector $est(r-1) \in C$, there is $v \in \{0,1\}$ such that $\big(\mathsf{nb}(v, est(r)) - \mathsf{nb}(1-v, est(r))\big) > t$. It follows that, even if a process p_i misses the message $\text{EST}(r, v)$ from up to t processes, it receives more messages carrying v than messages carrying $(1-v)$ (line 5). Hence, we have $aux1_i(r) = v$ (line 7). As this is true at any process that terminates round r, the property follows. ■

Property 3: $P2(r)$:
$\forall\ i,j:\ \big((aux2(r)[i] = v \neq \perp) \Rightarrow (aux2(r)[j] \in \{v, \perp\})\big)$.

Proof: Let p_i be such that, at round r, we have $aux2_i = v \neq \perp$. It follows from lines 9-11 that p_i received $\text{AUX}(r, v)$ from a set $Q1$ including at least $(n-t)$ processes.

Any process p_j that executes line 11 during round r, received a message $\text{AUX}(r, -)$ from a set $Q2$ including at least $(n-t)$ processes. As $n > 2t$, $|Q1| + |Q2| \geq 2n - 2t > n$, from which we conclude that there is a process $p_k \in Q1 \cap Q2$. As p_k sends the same message to p_i and p_j, it follows that $v \in rec2_j$ line 10). Then, due to line 11, we have $aux2_j \in \{v, \perp\}$ at round r. ■

The next corollary follows from Property $P2(r)$.

Corollary 1: $\forall\ i,j\ :\ \big((aux2(r)[i] = v \neq \perp) \wedge (aux2(r)[j] = w \neq \perp)\big) \Rightarrow (v = w)$.

Property 4: $P3(r)\ :\ \forall i,j\ :\ (rec3_i(r) = \{v, \ldots, v\}$ (where $v \neq \perp$) $\oplus$ $(rec3_j(r) = \{\perp, \ldots, \perp\})$ (*mutex*).

Proof: Let us assume that, at round r, $rec3_i$ contains $(t{+}1)$ copies of the same non-$\perp$ value v, while $rec3_j$ contains only copies of $\perp$. This means that p_i received the message $\text{AUX2}(r, v)$ from a set $Q1$ of $(t+1)$ processes, while p_j received the message $\text{AUX2}(r, \perp)$ from a set $Q2$ of $(n-t)$ processes. Similarly to the proof of Property $P2(r)$, we have $|Q1| + |Q2| = (t{+}1) + (n{-}t) > n$, from which we conclude that there is a process $p_k \in Q1 \cap Q2$. As this process sent the same message $\text{AUX2}(r, -)$ to p_i and p_j, $rec3_j$ contains a copy of v, which concludes the proof. ■

Theorem 5: Algorithm *1* solves binary consensus in the system model $\mathit{CAMP}_{n,t}[t < n/2, \mathit{LC}]$. Moreover, if the input vector belongs to the condition C, A process decides in three communication steps.

Proof: Let us first consider the case where the input vector $I = est(0) \in C$. Then, due to the property $P1(1)$, all the local variables $aux1_i$ are equal to the same proposed value v (the value obtained from the condition C). It directly follows from the text of algorithm that any multiset $rec2_i$ contains only copies of v. Due to Corollary 1, no local variable $aux2_i$ is different from v. It then follows that any multiset $rec3_i$ can contain v only, from which we conclude that the processes decide the same proposed value v, and decision occurs in three communication steps.

Let us now consider that $I \notin C$. In this case, both 0 and 1 have been proposed, from which follows the validity property.

Let r be the first round during which a process p_i decides, and v the value it decides. It then follows from Property $P2(r)$ and $P3(r)$ that any other process p_j that executes round r is such that $rec3_j$ contains v. Hence, if p_j decides at round r it decides v (line 15), and if it progresses to round $(r+1)$, it assigns v to est_j (line 16), which means that, from round $(r+1)$, no process can have $(1-v)$ as estimate value. The agreement property follows from this observation.

If there is a round r such that $est(r) \in C$, we are in the case of the first paragraph, and decision is obtained at the latest in round r. The proof that there is a round r such $est(r) \in C$ is the same as the proof of the Termination of the B83 algorithm (which implicitly considers the very poor condition $C' = \{[0,\dots,0],[1,\dots,1]\} \subsetneq C$). Let p be the probability that at end of a round r, $est(r) \in C$. We have $p > 0$. Let $P(r)$ be the probability that there is a round $r' \le r$ such that $est(r') \in C$. We have $P(r) = p + (1-p)p + (1-p)^2 p + \dots + (1-p)^{r-1} p = 1-(1-p)^r$. It follows that $\lim_{r \to +\infty} P(r) = 1$ which concludes the proof. (Let us notice that, when $est(r) \in C$, the re-ordering of message reception by the adversary is inoperative.) ■

IV. Expected Number of Communication Steps: Theoretical View

The previous section presented a local coin-based binary consensus algorithm, which benefits from the fact that, at some round r, the input vector defined by the estimate values at the beginning of r belongs to the symmetric condition C. This section, which constitutes the second and noteworthy contribution of the paper, shows that its expected number of communication steps is $3\big(1-\mathsf{erf}\big(\frac{\alpha}{\sqrt{2}}\big)\big)^{-1}$ (for $\alpha = 1$, it is smaller than 10) when $t < \alpha\sqrt{n}$.

Notations

- $\mathsf{erf}(x)$: Gauss error function $\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$.
- $\lambda(\alpha) = \frac{1}{1-\mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right)}$.
- $[\bot] = [\bot,\dots,\bot]$.
- Reminder: $\mathbb{P}()$ is the probability function, and $P_k(n) = \mathbb{P}\left(est(r) \in C \mid \mathsf{nb}(0, aux2(r)) = k\right)$.

The lemma that follows explores the best strategy for the asynchronous adversary during round r. According to property $P1(r+1)$, the adversary must avoid that $est(r) \in C$, otherwise termination will be reached in round $r+1$. Lemma 1 proves that the best strategy for the asynchrony adversary, consists in ensuring that all processes invoke $\mathsf{random}()$ at the end of round r, or in other words $aux2(r) = [\bot]$. This lemma actually defines the "worst case" scenario in terms of probability, from a decision point of view. Note that the proof does not use the hypothesis that the adversary cannot read messages, which implies that it is also valid in models where the adversary can re-order messages based on their content.

Lemma 1:
$\forall r : \mathbb{P}\left(est(r) \in C\right) \ge \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\bot]\right)$.

Proof: The first part of the proof consists in showing that, for any round r, $\mathbb{P}\left(est(r) \in C \mid aux2(r) \neq [\bot]\right) \ge \mathbb{P}\left(est \in C \mid aux2(r) = [\bot]\right)$. To this end, let us consider the case where, at round r, $aux2(r)[i] \in \{\bot, 0\}$ for all p_i (the case $aux2(r)[i]_i \in \{\bot, 1\}$ is symmetric), and prove the result by induction on the number of processes p_i such that $aux2(r)[i] = 0$.

We have $P_0(n) = \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\bot]\right)$. Suppose (by induction) that, for some k such that $0 \le k < n$, $P_k(n) \ge \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\bot]\right)$. By the characterization of the condition for the binary consensus, we have:

$$\begin{aligned} P_k(n) =& \,\mathbb{P}\left(\textstyle\sum_{i=1}^{n} est(r)[i] < \frac{n-t}{2} \;\middle|\; \mathsf{nb}(0, aux2(r)) = k\right) \\ &+ \mathbb{P}\left(\textstyle\sum_{i=1}^{n} est(r)[i] > \frac{n+t}{2} \;\middle|\; \mathsf{nb}(0, aux2(r)) = k\right). \end{aligned} \tag{1}$$

As the processes play a symmetric role, we can assume without loss of generality that the processes p_i such that $aux2(r)[i] = 0$ are processes $p_{n-k+1}, \dots, p_n$. Therefore, we have

$$\begin{aligned} P_k(n) = \quad & \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k} est(r)[i] < \frac{n-t}{2}\right) \\ & + \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k} est(r)[i] > \frac{n+t}{2}\right). \end{aligned} \tag{2}$$

We now use the law of total probabilities to isolate the value drawn by process p_{n-k}. There are two possibilities with probabilities $\mathbb{P}(est(r)[n-k] = 0) = \mathbb{P}(est(r)[n-k] = 1) = \frac{1}{2}$, which gives

$$\begin{aligned} 2P_k(n) = \quad & \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k-1} est(r)[i] < \frac{n-t}{2}\right) \\ & + \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k-1} est(r)[i] > \frac{n+t}{2}\right) \\ & + \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k-1} est(r)[i] < \frac{n-t}{2}\right) \\ & - \mathbb{P}\left(\frac{n-t}{2} - 1 \le \textstyle\sum_{i=1}^{n-k-1} est(r)[i] < \frac{n-t}{2}\right) \\ & + \mathbb{P}\left(\textstyle\sum_{i=1}^{n-k-1} est(r)[i] > \frac{n+t}{2}\right) \\ & + \mathbb{P}\left(\frac{n+t}{2} - 1 < \textstyle\sum_{i=1}^{n-k-1} est(r)[i] \le \frac{n+t}{2}\right). \end{aligned} \tag{3}$$

We can now piece together the parts forming $P_{k+1}(n)$. Moreover, as $\sum_{i=1}^{n-k-1} est(r)[i]$ is an integer, the intervals –of length 1– of the two remaining parts contain one value only. This gives:

$$\begin{array}{rl} 2(P_k(n) - P_{k+1}(n)) & \\ = & \mathbb{P}\left(\sum_{i=1}^{n-k-1} est(r)[i] = \left\lfloor \frac{n+t}{2} \right\rfloor\right) \\ & -\mathbb{P}\left(\sum_{i=1}^{n-k-1} est(r)[i] = \left\lfloor \frac{n-t}{2} \right\rfloor - 1\right). \end{array} \quad (4)$$

If $k > \frac{n-t+1}{2}$, then $\mathbb{P}\left(\sum_{i=1}^{n-k-1} est(r)[i] = \left\lfloor \frac{n+t}{2} \right\rfloor\right) = 0$ and $P_k(n) - P_{k+1}(n) \leq 0$. Otherwise, the probability that $\sum_{i=1}^{n-k-1} est(r)[i] = x$ is $2^{-(n-k-1)}\binom{n-k-1}{x}$. If $k \geq t$, we have $\frac{n-k-1}{2} \leq \left\lfloor \frac{n-t}{2} \right\rfloor - 1 \leq \left\lfloor \frac{n+t}{2} \right\rfloor$. As $\binom{n-k-1}{x}$ is decreasing when x varies from $\frac{n-k-1}{2}$ to $n-k-1$, we have $P_k(n) - P_{k+1}(n) \leq 0$. In the last case, $k < t$. We use the identity $\binom{n-k-1}{\lfloor \frac{n+t}{2} \rfloor} = \binom{n-k-1}{n-k-1-\lfloor \frac{n+t}{2} \rfloor}$. Then, $n-k-1-\left\lfloor \frac{n+t}{2} \right\rfloor \leq \left\lfloor \frac{n-t}{2} \right\rfloor - 1 \leq \frac{n-k-1}{2}$. As $\binom{n-k-1}{x}$ is increasing when x varies from 0 to $\frac{n-k-1}{2}$, we have $P_k(n) - P_{k+1}(n) \leq 0$.

In all cases, this inequality concludes the induction: $P_{k+1}(n) \geq P_k(n) \geq \mathbb{P}\left(est(r) \in C \mid [aux2(r) = \perp]\right).$

Finally, the second part of the proof consists in using the previous equations and invoking the law of total probability. To this end, let SET be the set of vectors of size n, where $x/\perp$ means "x or $\perp$", $(0/\perp)^n \cup (1/\perp)^n$ (i.e., SET contains all vectors made up of 0 and $\perp$, plus all vectors made up of 1 and $\perp$). We have:

$$\begin{array}{rl} \mathbb{P}\left(est(r) \in C\right) & \\ = & \sum_{V \in SET} \mathbb{P}(aux2(r) = V) \\ & \quad \times \mathbb{P}\left(est(r) \in C \mid aux2(r) = V\right)) \\ \geq & \sum_{V \in SET} \mathbb{P}(aux2(r) = V) \\ & \quad \times \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\perp]\right) \\ \geq & \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\perp]\right). \end{array} \quad (5)$$

■

The next lemma proves that, even when the asynchrony adversary uses its best strategy (given by Lemma 1) at round r, there is still a lower-bounded probability to terminate at round $r+1$ when $t < \alpha\sqrt{(n)}$. To obtain a lower bound, let us now consider the size n of the vectors (number of processes) and look at how n impacts $\mathbb{P}\left(est(r) \in C\right)$ when it increases.

Lemma 2: Let $\alpha \in]0, \sqrt{n}[$ and $t < \alpha\sqrt{n}$. The probability that, for any r, $est(r) \in C$ is lower-bounded when $n \to \infty$. Namely, $\lim_{n\to\infty} \mathbb{P}\left(est(r) \in C\right) \geq \frac{1}{\lambda(\alpha)}$.

Proof: Let $0 < \alpha < \sqrt{n}$, and let us denote by $\sigma = \frac{1}{2}$ the standard deviation of the local coins and by $\mu = \frac{1}{2}$ their expected value. By Lemma 1, we have :

$$\begin{array}{rl} \mathbb{P}\left(est(r) \in C\right) & \\ \geq & \mathbb{P}\left(est(r) \in C \mid aux2(r) = [\perp]\right); \\ \geq & \quad \mathbb{P}\left(\sum_{i=1}^{n} est(r)[i] < \frac{n-\alpha\sqrt{n}}{2} \mid aux2(r) = [\perp]\right) \\ & +\mathbb{P}\left(\sum_{i=1}^{n} est(r)[i] > \frac{n+\alpha\sqrt{n}}{2} \mid aux2(r) = [\perp]\right) \\ \geq & \quad \mathbb{P}\left(\frac{\sum_{i=1}^{n} est(r)[i] - n\mu}{\sigma\sqrt{n}} < -\alpha \mid aux2(r) = [\perp]\right) \\ & +\mathbb{P}\left(\frac{\sum_{i=1}^{n} est(r)[i] - n\mu}{\sigma\sqrt{n}} > \alpha \mid aux2(r) = [\perp]\right). \end{array} \quad (6)$$

Under the hypothesis $aux2(r) = [\perp]$, the $est(r)[i]$ for $1 \leq i \leq n$ are independent random variables following the same uniform Bernoulli law. Therefore, by the central limit theorem, and as the function $\mathsf{erf}()$ is odd, we have

$$\begin{array}{rl} \lim_{n\to\infty} \mathbb{P}\left(est(r) \in C\right) \geq & \frac{1}{2}\left(1 + \mathsf{erf}\left(\frac{-\alpha}{\sqrt{2}}\right)\right) + \\ & \left(1 - \frac{1}{2}\left(1 + \mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right)\right)\right) \\ \geq & 1 - \mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right) = \frac{1}{\lambda(\alpha)}. \end{array} \quad (7)$$

■

Theorem 6: Let $\alpha \in]0, \sqrt{n}[$. Assuming $t < \alpha\sqrt{n}$, the expected number of communication steps for Algorithm *1* to terminate is bounded. Moreover, this number tends to $3\lambda(\alpha)$when $n \to \infty$.

Proof: The random draws done at each round are independent, so the probability to terminate at each round is lower-bounded by a geometric distribution of parameter $P_0(n)$, with an expected value of $\frac{1}{P_0(n)}$. By Lemma 2, $\lim_{n\to\infty} P_0(n) = \frac{1}{\lambda(\alpha)}$. Therefore, $\lim_{n\to\infty} \frac{1}{P_0(n)} = \lambda(\alpha)$.

By definition of the limit, for $\varepsilon = 1$, there exists a $N_\varepsilon \in \mathbb{N}$ such that, for all $n > N_\varepsilon$, $\lambda(\alpha) - \varepsilon < \frac{1}{P_0(n)} < \lambda(\alpha) + \varepsilon$. Therefore, the expected number of rounds is bounded by $\mathsf{max}\left(\lambda(\alpha) + 1, \mathsf{max}_{n \leq N_\varepsilon} P_0(n)\right)$ and its limit when n grows to infinity is $\lambda(\alpha) = \left(1 - \mathsf{erf}\left(\frac{\alpha}{\sqrt{2}}\right)\right)^{-1}$.

As each round is made up of three communication steps, the theorem follows. ■

V. An Improved Algorithm for $t < n/4$

The algorithm: Algorithm 2 (denoted Algo2 in Table I) is a version of Algo1 customized for $t < n/4$ (model $CAMP_{n,t}[t < n/4, \text{LC}]$). It shows that, decreasing t-resilience from $t < n/2$ to $t < n/4$, allows the saving of one communication step per round. To make the understanding easier, the lines with the same statement in both algorithms are identified with the same number, and the lines that are modified are postfixed by the letter "M". There is no new line.

The modified lines: In Algo2, the second phase of Algo1 is suppressed (lines 10-13), which entails the disappearance of the default value $\perp$. Consequently, lines 14-16 are modified as follows.

```
operation propose ($v_i$) is % $v_i \in \{0,1\}$ %
(1)    $est1_i \leftarrow v_i$; $r_i \leftarrow 0$;
(2)    while true do
(3)    $r_i \leftarrow r_i + 1$;
——  % Phase 1: Exchange of current estimate values ————————————————
(4)      broadcast EST ($r_i, est_i$);
(5)      wait (EST ($r_i, -$) received from $(n-t)$ processes);
(6)      for $x \in \{0,1\}$ do $nb_i[x] \leftarrow$ number of messages (EST ($r_i, x$) received) end for:
(7)      if ($nb_i[1] \geq nb_i[0]$) then $aux1_i \leftarrow 1$ else $aux1_i \leftarrow 0$ end if;
———— % Here: $P1(r) \overset{def}{=} \big((est(r-1) \in C) \Rightarrow (\forall\, i,j:\ aux1(r)[i] = aux1(r)[j])\big)$
——    % Phase 2+3: Exchange of current $aux1$ values and try to decide ————————————
(8)      broadcast AUX1 ($r_i, aux1_i$);
(9)      wait (AUX1 ($r_i, -$) received from $(n-t)$ processes );
(14.M)  $rec3_i \leftarrow$ multiset of the values $aux1$ received;   % The values in $rec3_i$ are from the set $\{0,1\}$
———— % Here: $P3'(r) \overset{def}{=}$
              $\forall i,j,v,r:\quad \big(\mathsf{nb}(v, rec3_i(r)) \geq n-t\big) \Rightarrow \big(\mathsf{nb}(v, rec3_j(r)) \geq n-2t\big)$
                          $\wedge\ \big(\mathsf{nb}(v, rec3_i(r)) \geq n-2t\big) \Rightarrow \big(\mathsf{nb}(1-v, rec3_j(r)) < n-2t\big).$
(15.M)  case ($\exists v: \mathsf{nb}(v, rec3_i) = n-t$)                  do broadcast DECIDE($r_i, v$); return($v$)  % decision
(16.M)        ($\exists v: n-2t \leq \mathsf{nb}(v, rec3_i) < n-t$)     do $est_i \leftarrow v$                            % adoption
(17)          otherwise                                             do $est_i \leftarrow$ random()                     % random help
(18)      end case
(19)   end while.
```

Algorithm 2: Improved version for the model $CAMP_{n,t}[t < n/4, \text{LC}]$ (code for p_i)

- Line 14.M. During a round r, the multiset $rec3_i$ contains now $(n-t)$ elements, each being 0 or 1.

  A central part of Algo1 is the mutual exclusion captured by predicate $P3(r)$. This predicate ensures two things: (i) if, during a round r, p_i decides a value v, another process p_j either decides v or adopts v as new estimate. In this case, the value $(1-v)$ disappears. As Algo2 cannot use the default value $\perp$ used by Algo1, its predicate, denoted $P3'(r)$, must explicitly ensure that $(1-v)$ cannot be adopted by process p_j when v is decided or adopted by a process p_i. To this end, $P3'(r)$ is defined as follows:

  $$\forall i,j,v,r:\quad \big(\mathsf{nb}(v, rec3_i(r)) \geq n-t\big) \Rightarrow \big(\mathsf{nb}(v, rec3_j(r)) \geq n-2t\big)$$
  $$\wedge\ \big(\mathsf{nb}(v, rec3_i(r)) \geq n-2t\big) \Rightarrow \big(\mathsf{nb}(1-v, rec3_j(r)) < n-2t\big).$$

  For commodity, the first line and the second line of this predicate are abbreviated $A1(i,j,r,v) \Rightarrow (B1(i,j,r,v)\ \wedge\ B2(i,j,r,v))$, and $A2(i,j,r,v) \Rightarrow B2(i,j,r,v)$, respectively. As we are about to see, the first part of $P3'(r)$ is related to safety (if p_i decides v, no process can decide or adopt $(1-v)$, and no process draws a random number), while its second part is related to liveness (to this end if p_i adopts v, no process can adopt $(1-v)$).
- Line 15.M. If $rec3_i$ contains “a lot of” occurrences of v, where –according to $A1(i,j,r,v)$– “a lot of” means $(n-t)$, p_i decides it. Thanks to the predicate $B1(i,j,r,v)$, we know that no other process will exploit random numbers at line 17. The dices no longer need to be cast.
- Line 16.M. If $rec3_i$ contains “enough” occurrences of v, where –according to $B1(j,r,v)$– “enough” means at least $(n-2t)$, p_i adopts v as new estimate, before proceeding to the next round. This is consistent with the fact if a process decides at round r, p_i adopts its value.

  But, the predicate $B1(j,r,v)$ can be satisfied, while no process decides during the current round r. In this case it is crucial that, if p_i adopts a value v, no other process p_j adopts the value $(1-v)$ (if different processes could adopt distinct values at line 16.M of a round r, they could never attain a round r such that $est(r) \in C$, thereby compromising termination). Such a bad scenario is prevented from occurring thanks to the second predicate. If p_i adopts v (predicate $A2(i,j,r,v)$), $(1-v)$ cannot be adopted by another process p_j (predicate $B2(i,j,r,v)$).

The proof of Algo2 is similar to one of Algo1. We just prove here $P3'(r)$.

Proof of Property $P3'(r)$: Let us assume a round r at which $\mathsf{nb}(v, rec3_i(r)) > n-t$. This means that there is a set $Q1$ including least $(n-t)$ processes that broadcast the message AUX1(r,v). It follows that any process p_j receives the message AUX1(r,v) from a set $Q2$ including at least $(n-2t)$ processes. Hence, $B1(i,j,r,v)$ is satisfied, i.e., no process p_j can decide or adopt the value $(1-v)$, and no process draws a random number.

Assuming $n = 4t + x$, where $x \geq 1$, let us now consider that the predicate $\mathsf{nb}(v, rec3_i(r)) \geq n-2t$ is satisfied. It follows that p_i received the message AUX1(r,v) from a set including at least $n-2t = 2t+x$ processes. As $n = 4t+x$, any process p_j can receive the message AUX1$(r, 1-v)$ from at most $n-(2t+x) = n-2t$ processes, from which the predicate $B2(i,j,r,v)$ follows. ■

VI. Conclusion

The contributions of the paper: This paper has presented a binary consensus algorithm based on local coins with a constant expected number of rounds when $t < O(\sqrt{n})$, despite a asynchrony adversary that can re-order message delivery according to their content. To our knowledge, this is the first algorithm with a so small number of rounds and such a strong asynchrony adversary. To this end, the algorithm benefits from the condition-based approach (it is actually the first algorithm relying on both conditions and randomization to solve consensus).

Independently of its theoretical value, this algorithm has a practical interest. Let us remark that, while failures do occur, they are rare. Hence, systems where n is small (e.g., $n = 17$), and subject to few failures (e.g., $t \leq 4$) can benefit from this algorithm.

Extending and generalizing the previous results: Several research directions are worth investigating. From a theoretical point of view, a main (and difficult) issue consists in knowing if there is or not an algorithm with a constant expected number of rounds when $t > O(\sqrt{n})$ (we conjecture the answer is "no"). Another issue, consists in extending the proposed approach to Byzantine process failures.

Acknowledgments

This work has been partially supported by the French ANR project 16-CE40-0023-03 DESCARTES (devoted to layered and modular structures in distributed computing).

References

[1] Aguilera M.K. and Toueg S., Failure detection and randomization: a hybrid approach to solve consensus. *SIAM Journal of Computing*, 28(3):890-903 (1998)

[2] Alistarh D., Aspnes J., King V. and Saia J. Communication-efficient randomized consensus. *Proc. 28th International Symposium on Distributed Computing (DISC'14)*, Springer LNCS 8784, pp. 61-75 (2014)

[3] Aspnes J. and Herlihy M., Fast randomized consensus using shared memory. *Journal of algorithms* vol. 11(3) pp. 441-461. (1990)

[4] Attiya H. and Censor K., Tight bounds for asynchronous randomized consensus. *Journal of the ACM* vol. 55(5), p. 20 (2008)

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

[6] 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)

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

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

[9] Chandra T., Hadzilacos V., and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)

[10] Dolev D., Dwork C., and Stockmeyer L., On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77-97 (1987)

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

[12] 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)

[13] 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)

[14] Moniz H., Neves N.F., Correia M., and Veríssimo P., RITAS: Services for randomized intrusion tolerance. *IEEE Trans. Dependable and Secure Computing*, 8(1): 122-136 (2011)

[15] Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of ACM*, 62(4), Article 31, 21 pages (2015)

[16] 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)

[17] Mostéfaoui A., Rajsbaum S., Raynal M. and Travers C., The Combined power of conditions and information on failures to solve asynchronous set agreement. *SIAM Journal of Computing*, 38(4):1574-1601 (2008)

[18] Mostéfaoui A. and Raynal M., Solving consensus using Chandra-Toueg's unreliable failure detectors: a general quorum-based approach. *Proc. 13th Int'l Symposium on Distributed Computing (DISC'99)*, Springer-Verlag 1693, pp. 49-63 (1999)

[19] Mostéfaoui A. and Raynal M., Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54(5):501-520 (2017)

[20] 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)

[21] Mostéfaoui A., Raynal M., and Tronel F., The best of both worlds: A hybrid approach to solve consensus. *Proc. Int'l Conference on Dependable Systems and Networks (DSN'00)*, IEEE Computer Society Press, pp. 513-522 (2000)

[22] Powell D., Failure mode assumptions and assumption coverage. *Proc. of the 22nd Int'l Symposium on Fault-Tolerant Computing (FTCS-22)*, IEEE Computer Society Press, pp.386-395 (1992)

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

[24] Raynal M., *Fault-tolerant message-passing distributed systems : an algorithmic approach*. Springer, 560 pages (2018)

[25] Zhang J. and Chen W., Bounded cost algorithms for multivalued consensus using binary consensus instances. *Information Processing Letters*, vol 109(17), pp. 1005-1009 (2009)