Technical Report

UCAM-CL-TR-593
ISSN 1476-2986

Number 593

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Paxos at war

Piotr Zieliński

June 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Paxos at War

Piotr Zieliński
University of Cambridge
Computer Laboratory
piotr.zielinski@cl.cam.ac.uk

Abstract

The optimistic latency of Byzantine Paxos can be reduced from three communication steps to two, without using public-key cryptography. This is done by making a decision when more than $(n + 3f)/2$ acceptors report to have received the same proposal from the leader, with n being the total number of acceptors and f the number of the faulty ones. No further improvement in latency is possible, because every Consensus algorithm must take at least two steps even in benign settings. Moreover, if the leader is correct, our protocol achieves the latency of at most three steps, even if some other processes fail. These two properties make this the fastest Byzantine agreement protocol proposed so far.

By running many instances of this algorithm in parallel, we can implement Vector Consensus and Byzantine Atomic Broadcast in two and three steps, respectively, which is two steps faster than any other known algorithm.

1 Introduction

Consider a distributed system with many independent processes. In the Consensus problem [2, 18], all correct processes are expected to eventually decide on the same value, proposed by one of the processes. *Uniform Consensus* is a stronger variant of the problem, where the decisions made by *all* processes (including the faulty ones) must be the same. Keidar and Rajsbaum [11] proved that any Uniform Consensus algorithm needs at least two communication steps to decide, even if there are no failures.

For the asynchronous crash-stop model, where processes can fail only by crashing, there are several known algorithms that meet this lower bound [10, 14, 20]. This paper presents an algorithm that achieves this bound in the asynchronous Byzantine model, where faulty processes can exhibit arbitrary behaviour.

Paxos [14] is one of the algorithms achieving the two-steps latency in the crash-stop model. To deal with malicious participants, two solutions have been proposed. Attiya and Bar-Or [1] use unmodified Paxos with emulated robust shared memory, which requires *eight* communication steps in failure-free runs. The other method, designing a Byzantine version of Paxos, was first proposed by Castro and Liskov [5] and then generalized by Lampson [17]. It not only achieves *three* communication steps in optimistic settings, but also does so without public-key cryptography in failure-free runs.

	latency		
algorithm	all	leader	public-key
Malhki and Reiter [19]	9		yes
Malhki and Reiter [19]	6		yes
Dwork et al. [9]	4		yes
Kihlstrom [12]	4		yes
Doudou et al. [8]	4		yes
Bracha [4]	9		no
Dwork et al. [9]	7		no
Castro and Liskov [5]	3	3	no
Kursawe [13]	2	—	no
this paper	2	3	no

Figure 1: Comparison of several Byzantine agreement algorithms [7]. Latency is given for two scenarios: with all processes correct and with the current leader correct. The "—" denotes that the algorithm handles the latter scenario pessimistically. The last column states whether public-key cryptography is used in failure-free runs.

In this paper, we present an improved version of this algorithm, which takes only *two* communication steps to decide in failure-free runs. It is the first Byzantine version of Paxos to achieve this early-deciding property. (The existence of such an algorithm was suggested by Lamport [16].) Similarly to the original algorithm [5], we do not use public-key cryptography in failure-free runs. In runs with failures, our algorithm may be used with or without public-key cryptography, depending on which of the two variants presented in this paper is used.

Several non-Paxos-based Byzantine agreement protocols have been proposed [8, 9, 12, 19]. They all require at least four communication steps in failure-free runs, which is two more than our algorithm (Figure 1). The only exception is the optimistic Byzantine agreement algorithm by Kursawe [13], which also takes only two communication steps to decide in failure-free runs. However, if *any* of the processes has failed, a new pessimistic Byzantine agreement protocol must be started. On the other hand, our algorithm needs only one more communication step in such circumstances, provided that the leader is correct.

The Validity property ensured by the algorithm from [13] is too weak to implement other useful abstractions such as Atomic Broadcast [8]. The Validity property offered by our algorithm is stronger and enables us to implement Vector Consensus [8] with no additional message delay. This abstraction can in turn be used to construct an Atomic Broadcast protocol that requires only one additional communication step [8].

Independently of this work, Vulovic et al. [22] have recently discovered a similar improvement to the original Byzantine Paxos algorithm [5]. However, their algorithm can avoid using public-key cryptography only if more than $\frac{n+3f}{2}$ acceptors are correct.

The paper is structured in the following way. Section 2 describes the system model and formally states the Byzantine Consensus problem. Section 3 introduces the notation. The algorithm is presented in Section 4 and is used in Section 5 to implement fast Vector

Symbol	Description
a, b, c, ...	Acceptors (c for correct acceptors)
v, w	Elements of $\mathcal{V}$
x, y	Elements of $\mathcal{V} \cup \{\perp\}$
r, s, t	Round numbers ($s < r$)
V, U	Set variables

Figure 2: Conventional variable names used in the paper

Symbol	Description
p_r	Proposer for round r
n	Number of acceptors, $n = \|\mathcal{A}\|$
f	Upper bound on the number of faulty acceptors,
$\mathcal{V}$	Set of all values.
$\mathcal{P}$	Set of all processes.
$\mathcal{A}$	Set of all acceptors, $\|\mathcal{A}\| = \|\mathcal{C}\| + \|\mathcal{F}\| = n$.
$\mathcal{C}$	Set of all correct acceptors, $\|\mathcal{C}\| \geq n - f$.
$\mathcal{F}$	Set of all faulty acceptors, $\|\mathcal{F}\| \leq f$.
$\mathcal{S}$	Set of all round numbers $s < r$, that is, $\mathcal{S} = \{\, s : s < r \,\}$.

Figure 3: Symbols used in the paper

Consensus [8] and thus Atomic Broadcast for Byzantine settings. Section 6 provides a short summary and shows some possible directions of future research. The appendices contain a formal proof of the correctness of our algorithm.

2 System model and problem statement

The system consists of a (possibly infinite) set $\mathcal{P}$ of processes, which communicate using authenticated asynchronous channels. Processes can be either *correct* or *faulty*. It is normally assumed that only correct processes behave according to the specification, whereas the faulty ones can exhibit an arbitrary behaviour. In this paper, however, we will consider a model in which all processes behave according to the specification, but channels from faulty processes do not offer any guarantees; they can create, modify, and lose any number of messages. Although these two models are equivalent, the latter one makes proving and reasoning much easier. Also, since all processes are "internally correct", it allows us to solve the *Uniform* Consensus problem. Since Uniform Consensus restricts the behaviour of all processes (including the maliciously faulty ones), it is impossible to implement it in a model with "internally faulty" processes.

Processes belong to two, not necessarily distinct, classes: proposers and acceptors. The algorithm consists of many rounds r; in each of them, the proposer p_r responsible for this round issues a proposal and sends it to the acceptors, who try to make it a decision. If the first round fails, then the second one is started, then the third, and so on. The proposers for different rounds are not necessarily distinct. The proposer p_1 of the first

round is called the leader. This is because in normal, failure-free runs, p_1 will propose a value, which will become the decision, and no other rounds will be started. We denote the set of all proposable values by $\mathcal{V}$.

The set $\mathcal{A}$ of acceptors consists of exactly n processes, whose task is to decide on the value proposed by the proposer. We denote the set of correct acceptors by $\mathcal{C}$ and the set of faulty acceptors by $\mathcal{F}$. Although the algorithm does not have access to the sets $\mathcal{C}$ and $\mathcal{F}$ (i.e., it does not know which acceptors are correct), we introduce these two sets because of their usefulness in proofs.

As a convention, we use a, b, ... for acceptors (c for correct ones), p for proposers, v and w for values, V and U for set variables (Section 3.2), and r, s, t for round numbers. We use round numbers as superscripts and process names as subscripts (round proposers p_r are the only exception). Figures 2 and 3 summarize the use of symbols and conventions.

For safety properties, we assume that channels from correct processes cannot create or distort messages. We also assume that at most f acceptors are faulty ($|\mathcal{F}| \leq f$). For liveness properties, we make the following additional assumptions. Firstly, for infinitely many round numbers r, the proposer p_r is correct (recall that these p_r are not necessarily different). Secondly, all channels from correct processes are reliable, that is, all messages sent will eventually be received. Third, these channels will eventually be timely, that is, their transmission latency will, from some point onwards, be smaller than some parameter d. A run in which all processes are correct and the network is always timely is called *nice*.

Each proposer p_r has a constant singleton set $input^r = \{w\}$ that contains its initial proposal w. We assume that $input^r \subseteq range^r \subseteq \mathcal{V}$, where $range^r$ is a pre-agreed set of possible proposals of process p_r. The purpose of $range^r$ is to restrict the potential damage of a faulty p_r issuing nonsensical proposals. For example, $range^r$ can be the set of all possible proposals signed by some external trusted entity. We define I^r to be $input^r$ for correct proposers p_r and $range^r$ for the faulty ones.

The variant of Uniform Consensus considered in this paper is similar to the proposers-acceptors-learners version in [16] and Lazy Consensus in [23]. Formally, the following properties must be satisfied:

Agreement. No two acceptors decide on different values.

Validity. Each decision belongs to I^r for some round r.

Termination. All correct acceptors eventually decide.

3 Notation

3.1 Actions

We specify the algorithm as a collection of actions of the form $(name, guard, body)$. Here, *name* is the named action name, *guard* is a boolean expression, and *body* is an atomic operation which is performed when the action is *executed*. We assume the following about action execution. For safety properties: an action can be executed only if it is enabled (*guard* is true). For liveness properties: an action enabled forever will eventually be executed *at least* once. For performance properties: every enabled action that has not been executed yet will be executed immediately.

Name	Guard	Body
Spread_p	$x \in V$ and $p = \text{owner}\, V$	broadcast "$x \in V$"
Update_p	received "$x \in V$" from owner V	$V_p \leftarrow x$

Figure 4: Variable replication code for a process p.

3.2 Set variables

A *set variable* V is an initially empty set containing at most one element. It is stored at a designated process called its *owner*. The variable V can be (atomically) updated only by executing the assignment "$V \leftarrow x$", which is a shortcut for "if $V = \emptyset$, then set V to $\{x\}$". Since V can contain at most one element, $x \in V$ is equivalent to $V = \{x\}$.

For any x, the expression "$x \in V$" is *stable*, which means that once it becomes true it will remain so forever. Stability is an extremely useful property, because it allows us to avoid using temporal formulae in derivations and proofs. This way we can restrict our attention to a single moment of time. For example, consider an action that can be executed only if $x \in V$ in the past. Thanks to the stability of V we can replace the guard "there is a time t in the past at which $x \in V$" with a simpler one: "$x \in V$ (now)".

Since set variables can have at most one element, any variable $V \neq \emptyset$ can no longer change; we say it is *frozen*. Any set variable V can be made frozen by executing "$V \leftarrow x$" for any x, in particular for x being a special value $\perp \notin \mathcal{V}$. This model is equivalent to the one proposed by Lampson [17], with Lampson's x being $\{x\}$ here, *nil* being $\emptyset$, and *out* being $\{\perp\}$, but it is mathematically easier to deal with.

3.3 Replication

A set variable V_p owned by process p is a copy of V if it is updated only as described in Figure 4. If $x \in V$, then the owner of V broadcasts "$x \in V$". Upon reception of "$x \in V$", each process p updates its set variable V_p by executing $V_p \leftarrow x$.

If the owner of V is correct, this replication mechanism has two properties. Firstly, a copy cannot contain elements absent from the original, that is $V_p \subseteq V$ for any p. Secondly, each element of V will eventually become a member of V_p. In the rest of this paper, the details of the replication mechanism are not important, as long as these two properties are preserved.

We adopt the convention that, for any V, variable V_p is the copy of V owned by p. Since V_p is a set variable, it can also have copies. For example, V_{pq} is a copy of V_p owned by q. The same process can be applied over and over again, leading to multi-index copies V_{pqr}, V_{pqrs}, ... owned by the last process in the index sequence. To avoid requiring an infinite amount of memory, we assume that copies are created only if there is a possibility that they might be needed by the algorithm. Finally, note that the existence of V_p does not imply the existence of V. If V_p does not exist, then V_p is simply an independent variable.

3.4 Arrays

We will often need to consider more than one set variable at once. For this purpose, we assume that if the $i-th$ index in the variable name is a set X, then the expression denotes an array of all variables with the $i-th$ index in X. For example, $V_{\mathcal{A}p\mathcal{C}}$ denotes a two-dimensional array $A_{\mathcal{AC}}$ with $A_{ac} = V_{apc}$ for all $a \in \mathcal{A}$ and $c \in \mathcal{C}$. (Recall that $\mathcal{A}$ is the set of all acceptors and $\mathcal{C}$ is its subset containing only the correct ones.)

In our notation, an array has a given property iff all its entries have that property (e.g., $V_{\mathcal{A}p} \neq \emptyset$ iff $V_{ap} \neq \emptyset$ for all $a \in \mathcal{A}$). Similarly, two arrays with the same dimensions are in relation iff their corresponding elements are (e.g., $V_{\mathcal{A}p} \subseteq V_{\mathcal{A}}$ iff $V_{ap} \subseteq V_a$ for all $a \in \mathcal{A}$).

Only the *size* of the set $\mathcal{C}$ of correct acceptors is known to the algorithm ($|\mathcal{C}| \geq n - f$). Therefore, we will often be interested in the *number* of entries V of a given array A_P that satisfy a given condition Z. By $A_P(Z)$, we denote the number of processes $p \in P$ for which A_p satisfies Z. Formally,

$$A_P(Z) = |\{\, p \in P : A_p \text{ satisfies } Z \,\}|.$$

The condition Z can be one of "w", "$\hat{w}$", "$\circ$", and "$\bullet$", with the following meanings:

$$Z = w \stackrel{\text{def}}{\iff} A_p = \{w\}, \qquad Z = \hat{w} \stackrel{\text{def}}{\iff} A_p \subseteq \{w\},$$
$$Z = \circ \stackrel{\text{def}}{\iff} A_p = \emptyset, \qquad Z = \bullet \stackrel{\text{def}}{\iff} A_p \neq \emptyset.$$

Condition "w" tests whether w belongs to A_p now, whereas "$\hat{w}$" tests whether w can belong to A_p in the future. The other two conditions test whether A_p is empty ("$\circ$") or not ("$\bullet$").

For example, for each p, the expression $V_{\mathcal{C}p}(\bullet)$ denotes the number of nonempty V_{cp} that belong to correct acceptors c. As another example, $V_{\mathcal{CA}}(w)$ is a one-dimensional array $B_{\mathcal{A}}$ such that for every $a \in \mathcal{A}$ the value B_a is the number of correct acceptors c for which $w \in V_{ca}$. Therefore, $V_{\mathcal{CA}}(w) > f$ means that for every acceptor a, the number of entries V_{ca} corresponding to correct acceptors c is bigger than f.

We can treat actions also as set variables. For any action Action_a, we assume that "$\mathsf{Action}_a \leftarrow \perp$" is executed whenever Action_a is executed. This allows us to write expressions like $\mathsf{Action}_{\mathcal{A}}(\bullet) > f$, which is true if Action_a has been executed by more than f acceptors a. We additionally assume that if Action_a acts as a boolean expression, then it is equivalent to $\mathsf{Action}_a \neq \emptyset$. This convention is consistent with the intuitive meaning of "Action_a" in that context, which is "Action_a has been executed".

3.5 Diagrams

We illustrate the paper with a number of diagrams that depict example executions of the algorithm (e.g, Figure 5). As usual, arrows denote messages. Solid (black) arrows are normal messages. Dashed (red) ones represent messages sent (or directly influenced) by a faulty process. Such messages can be corrupted in any way, in particular, they can even be lost by the channel and never reach their destination. Small black circles ($\bullet$) represent executions of one or more actions, whose names are written below or above the column of "$\bullet$"s. The diagrams might create the illusion of tight synchronization between

the processes. It is therefore important to remember that the purpose of this apparent synchronization is merely to help the reader visualize the most common executions, not to introduce any additional synchrony assumptions into the algorithm.

4 Byzantine Paxos

The Byzantine Paxos algorithm progresses in a sequence of (possibly overlapping) *rounds*, starting with round 1. In each round r, the proposer p_r selects a value and passes it to acceptors, who try first to *weakly* and *strongly* accept it, and then to make it a decision (Section 4.1). Because of a faulty proposer, round r might not progress. In this case, acceptors can independently *locally freeze* it and start the next round with another proposer (Section 4.2).

To ensure Agreement and Validity, the value w proposed by p_r must be provably *good*, that is, *admissible* and *valid*. Admissibility means that no decision other than w can be made in any round $s < r$. Validity requires that w should belong to I^s for some $s \leq r$, that is, should be either $input^s$ of one of the correct proposers p_s, or $range^s$ of a faulty p_s. Only good values can be weakly accepted and only weakly accepted values can become a decision, which implies goodness of all decisions. Therefore, the algorithm satisfies Validity. Also, since all decisions made at the same round are identical, the algorithm satisfies Agreement. Testing goodness of a given w is covered by Section 4.3. Choosing a w whose goodness can be verified by all acceptors is discussed in Section 4.4.

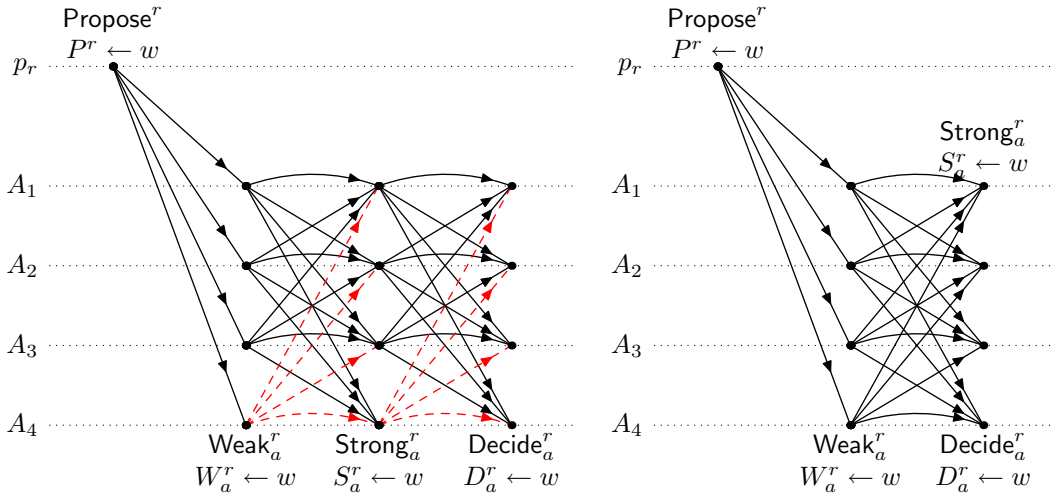
If the network is timely and p_r is correct, then the decision will be reached within three communication steps after p_r proposed. If in addition all acceptors are correct, then the number of steps required is reduced to two. Since we assume infinitely many correct proposers in p_1, p_2, ..., and the network being eventually timely, the former (three-step) property implies Termination of the algorithm. The latter (two-step) property implies that in nice runs (all processes correct and the network timely), the algorithm decides in two communication steps, without using public-key cryptography.

4.1 Single round

The structure of each round r shown in Figure 5(a) is similar to the Bracha broadcast [4]. First, proposer p_r selects a provably good $w \in \mathcal{V}$ (Sections 4.3 and 4.4) and sends it to the acceptors. At each acceptor a, the proposal w goes through three acceptance levels: *weak*, *strong*, and *decision*. Figure 5(b) shows that if all acceptors a are correct, then the decision is made one step earlier – at the same time as strong acceptance.

Figures 6 and 7 present the algorithm and the variables it uses. In action $\mathsf{Propose}^r$, process p_r chooses a provably good w and writes it to the set variable P^r by executing $P^r \leftarrow w$. The details of proving goodness will be presented later; for the moment, note that $input^1$ is always good in the first round, so p_1 can propose it without any proof. If the run is nice, all acceptors will decide in the first round; no proofs, and thus no public-key cryptography, will be used.

Proposal w is propagated from P^r to its copies P^r_a at all acceptors a, where it is weakly accepted by action Weak^r_a. As a result, if the proposer p_r is correct, we have $P^r_a \subseteq P^r \subseteq \{w\}$ and subsequently $W^r_a \subseteq \{w, \perp\}$ for all $a \in \mathcal{A}$. The possibility of $\perp \in W^r_a$ arises from the fact that a can locally freeze round r at any time by writing $\perp$ to W^r_a and



(a) A_4 is faulty

(b) All acceptors are correct

Figure 5: An example execution of a single round with $n = 4$ and $f = 1$.

Name	Guard	Body
Propose^r	provably good w	$P^r \leftarrow w$
Weak^r_a	good $w \in P^r_a$	$W^r_a \leftarrow w$
Strong^r_a	$W^r_{\mathcal{A}a}(w) > \frac{n+f}{2}$	$S^r_a \leftarrow w$
Decide^r_a	$W^r_{\mathcal{A}a}(w) > \frac{n+3f}{2}$	$D^r_a \leftarrow w$
Decide^r_a	$S^r_{\mathcal{A}a}(w) > 2f$	$D^r_a \leftarrow w$

Figure 6: Algorithm code for a single round r.

Symbol	Description
I^r	A non-replicated constant. For correct proposers, $I^r = input^r$ contains only the initial proposal of p_r. For faulty ones, $I^r = range^r$, the pre-agreed range of possible initial proposals of p_r.
P^r	An individual set variable owned by p_r, which contains the value w proposed by p_r and a proof that w is good at that round.
W^r_a	A *weak* approximation of the final decision at acceptor a. If p_r is correct, then there is a good $w \in \mathcal{V}$ such that $W^r_{\mathcal{A}} \subseteq \{w, \perp\}$.
S^r_a	A *strong* approximation of the final decision at acceptor a. There is a good $w \in \mathcal{V}$ such that $S^r_{\mathcal{A}} \subseteq \{w, \perp\}$ even if p_r is faulty.
D^r_a	The final decision at acceptor a. If $w \in D^r_a$, then for any round $r' > r$, proposer $p_{r'}$ will eventually be able to prove that $D^r_{\mathcal{A}} \subseteq \{w\}$.

Figure 7: Summary of round r variables.

Name	Guard	Body
Freeze_a^r	$D_a^r = \emptyset$ and $timer_a^r > \Delta$	$W_a^r \leftarrow \bot, \quad S_a^r \leftarrow \bot$
Freeze_a^r	$\mathsf{Freeze}_{\mathcal{A}a}^r(\bullet) > f$	$W_a^r \leftarrow \bot, \quad S_a^r \leftarrow \bot$
Clock_a^r	$\mathsf{Freeze}_{\mathcal{A}a}^s(\bullet) > 2f$ for all $s < r$	start round r timer $timer_a^r$
Decide_a^r	$D_{\mathcal{A}a}^r(w) > f$	$D_a^r \leftarrow w$

Figure 8: Code responsible for freezing round r.

S_a^r (see Section 4.2). On the other hand, a faulty p_r can propose different values of w to different acceptors, leading to $W_a^r \not\subseteq \{w, \bot\}$.

To deal with faulty proposers that propose different values to different acceptors, we introduce *strong acceptance*. Value w is strongly accepted at acceptor a (action $\mathsf{Strong}_\mathsf{a}^\mathsf{r}$) if more than $\frac{n+f}{2}$ acceptors report to have weakly accepted it. This ensures weak acceptance by a majority of correct acceptors. Therefore, $S_a^r \subseteq \{w, \bot\}$ for all $a \in \mathcal{A}$, even if p_r is faulty.

Finally, acceptor a decides on a weakly accepted w (Decide_a^r) if the following guarantee can be made. For any future round $r' > r$, the proposer $p_{r'}$ will be able to prove that $D_b^r \subseteq \{w\}$ for all $b \in \mathcal{A}$, even if acceptor a is faulty (Section 4.3). This is implied by either of the following conditions:

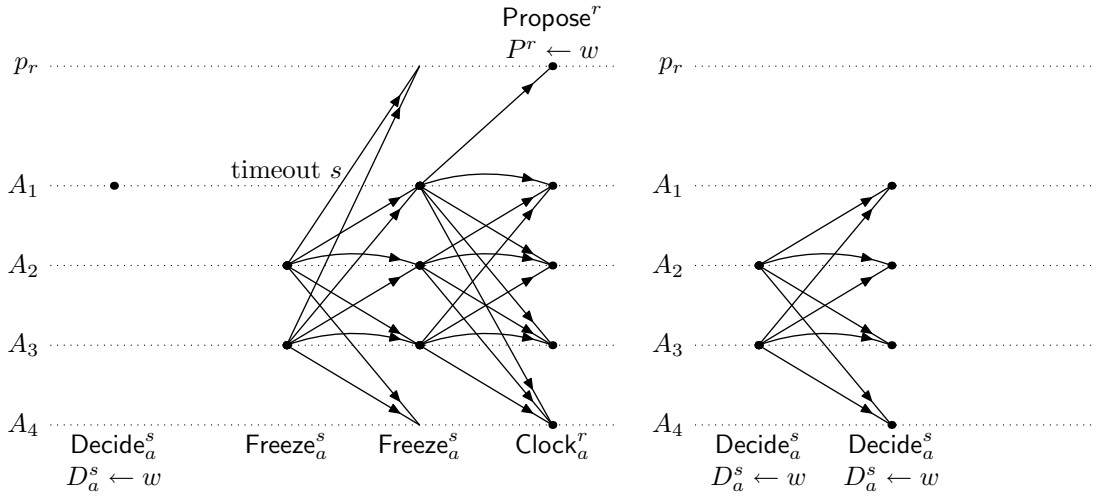
1. Condition $S_{\mathcal{A}a}^r(w) > 2f$, which states that more than $2f$ acceptors report to have strongly accepted w, comes from the original Byzantine Paxos algorithm [5, 17]. It ensures that if the network is timely and p_r correct, then the decision will be reached in three communication steps (Lemma B.12).

2. Condition $W_{\mathcal{A}a}^r(w) > \frac{n+3f}{2}$, which states that more than $\frac{n+3f}{2}$ acceptors report to have weakly accepted w, is new and ensures better performance in failure-free runs. It guarantees that the decision will be reached in *two* communication steps if less than $\frac{n-3f}{2} > 0$ acceptors are faulty (Lemma B.13).

4.2 Freezing rounds

A round with a faulty proposer may not make any progress. This can happen, for example, when the proposer sends different proposals to acceptors, or does not send them at all. Therefore, if an acceptor suspects that the current round r does not make progress, it may (independently of other acceptors) *locally freeze* it, which will allow the next round to start. Acceptor a locally freezes round r (Freeze_a^r) by executing "$W_a^r \leftarrow \bot$" and "$S_a^r \leftarrow \bot$", which ensures that W_a^r and S_a^r are non-empty. A round is considered *frozen* if it has been locally frozen by all correct acceptors.

If an acceptor a has not decided at round r ($D_a^r = \emptyset$) and its timeout has elapsed ($timer_a^r > \Delta$), then it locally freezes the round. Since this method cannot tell a faulty proposer from a slow network, a situation can arise in which some (correct) acceptors decide at round r and the others do not. In this case, the next round should start, because otherwise the algorithm might not make progress. As explained in Section 4.3, the proposer of that round cannot select a good proposal without round r being frozen first. Therefore, in some situations, it might be necessary for an acceptor a to locally



(a) More than f correct acceptors locally freeze

(b) More than f correct acceptors decide

Figure 9: Example executions of the freezing code from Figure 8, with $n = 4$, $f = 1$, and faulty A_4.

freeze round r, even if it has already decided ($D^r_a \neq \emptyset$). On the other hand, having all acceptors freeze the round after the timeout has elapsed (whether they have made a decision or not) would cause the next round to start unnecessarily.

We solve this problem by also making an acceptor locally freeze round r if more than f acceptors report to have done so (Figure 8). This ensures that at least one correct acceptor has already locally frozen round r, so faulty acceptors cannot jeopardize progress by freezing all the rounds. At the same time, if more than f correct acceptors execute Freeze^r, then eventually all acceptors will do so (Figure 9(a)). On the other hand, if at most f correct acceptors ever locally freeze round r, then more than $n - f - f > f$ correct acceptors decide at that round. The new form of Decide^r_a from Figure 8, which decides on w if more than f acceptors report to have done so, ensures that in this case all acceptors will eventually decide (Figure 9(b)). Safety is guaranteed, because any set of more than f acceptors must include a correct one.

The Decide^r_a action in Figure 8 is not necessary in the original Paxos algorithm. This is because there every process that has not learned about the decision can start a new round r, by requesting that all rounds $s < r$ should be frozen. However, in Byzantine settings, we cannot allow a single process to freeze a round, because then, malicious processes could easily prevent progress by freezing all rounds. This is also the reason why, in this paper, we do not differentiate between acceptors and learners [14].

Acceptor a starts the round r timer (Clock^r_a) when it knows that more than f correct acceptors have frozen all rounds $s < r$. This guarantees that eventually all correct acceptors will locally freeze all rounds $s < r$, which has two important consequences. Firstly, the proposer p_r will eventually be able to issue the proposal for round r. Secondly, all acceptors a will eventually execute Clock^r_a, which ensures a synchronization between the Clock^r_a actions at different acceptors a.

4.3 Predicate Good

In this section, we will construct a predicate $\mathsf{Good}(w)$ that is true only for good values w. Recall that w is good if it is admissible (no decision other than w can be made in any round $s < r$) and valid (w belongs to I^s for some $s \leq r$). Let $\mathcal{S} = \{\, s : s < r \,\}$ be the set of all round numbers smaller than r. The predicate $\mathsf{Good}(w)$ is computed using two arrays $\bar{W}^{\mathcal{S}}_{\mathcal{A}}$ and $\bar{S}^{\mathcal{S}}_{\mathcal{A}}$ such that $\bar{X}^{\mathcal{S}}_{\mathcal{C}} \subseteq X^{\mathcal{S}}_{\mathcal{C}}$ for each $X \in \{W, S\}$. In other words, we assume that $\bar{X}^s_c \subseteq X^s_c$ for every $s < r$ and every *correct* acceptor c. For example, $\bar{X}^{\mathcal{S}}_{\mathcal{A}}$ can be a copy $X^{\mathcal{S}}_{\mathcal{A}p}$ of $X^{\mathcal{S}}_{\mathcal{A}}$ at process p. Since Good depends on $\bar{X}^{\mathcal{S}}_{\mathcal{A}}$, it should be written formally as $\mathsf{Good}[\bar{W}^{\mathcal{S}}_{\mathcal{A}}, \bar{S}^{\mathcal{S}}_{\mathcal{A}}]$ or $\mathsf{Good}[\bar{X}^{\mathcal{S}}_{\mathcal{A}}]$ but in this section we abbreviate it to Good. The construction of Good ensures that if all $\bar{X}^{\mathcal{S}}_{\mathcal{C}}$ are non-empty, then there is at least one w for which $\mathsf{Good}(w)$ holds.

4.3.1 Computing Poss^s and Acc^s

In order to compute $\mathsf{Good}(w)$, we will first construct two sets for each $s < r$: Poss^s and Acc^s. The set Poss^s contains all possible decisions that can be made at round s. In other words, if $\mathsf{Poss}^s = K$, then $D^s_a \subseteq K$ will hold forever for every acceptor a. The set Acc^s contains only values that have been weakly accepted at round s, that is, $w \in \mathsf{Acc}^s$ implies $w \in W^s_a$ for at least one a.

The sets Poss^s and Acc^s can be defined as

$$\mathsf{Poss}^s = \big\{\, w \in \mathcal{V} \;\; : \;\; \bar{S}^s_{\mathcal{A}}(\hat{w}) > f \;\; \vee \;\; \bar{W}^s_{\mathcal{A}}(\hat{w}) > \tfrac{n+f}{2} \,\big\}, \quad \mathsf{Acc}^s = \{\, w \in \mathcal{V} : \bar{W}^s_{\mathcal{A}}(w) > f \,\}.$$

The definition of Acc^s is simple. If $w \in \bar{W}^s_a$ for more than f acceptors a, then $w \in \bar{W}^s_c$ for at least one correct acceptor c. Therefore, $w \in \bar{W}^s_c \subseteq W^s_c$ implies that c has weakly accepted w.

The definition of Poss^s is more complicated. Appendix A.3 shows that $\bar{S}^s_a(w) \leq f$ implies $S^s_{\mathcal{A}b}(w) \leq f + f = 2f$ for all acceptors b at any time. Similarly, $\bar{W}^s_a(w) \leq \frac{n+f}{2}$ implies $W^s_{\mathcal{A}b}(w) \leq \frac{n+f}{2} + f = \frac{n+3f}{2}$. Therefore, if $w \notin \mathsf{Poss}^s$, no acceptor b can ever decide on w in round s.

4.3.2 Computing $\mathsf{Good}(w)$

We define $\mathsf{Good}(w)$ to be true if at least one of the following conditions is met:

G1: $w \in I^r$ and $\mathsf{Poss}^s = \emptyset$ for all $s < r$,

G2: there is $s < r$ such that $w \in \mathsf{Acc}^s$ and $\mathsf{Poss}^t \subseteq \{w\}$ for all t with $s \leq t < r$.

The former condition implies that $w \in I^r$ and that no decision can be made by any round $s < r$. Since $w \in I^s$ for $s = r$, this means that w is good at round r.

The latter condition implies that w has been weakly accepted at round s, so it is good (i.e., valid and admissible) at that round. Validity at round s implies validity at round r. Admissibility at round s together with the fact that no decision different than w can be made at rounds t with $s \leq t < r$, implies admissibility of w at round r.

Appendix A.3 shows that if the entries $\bar{S}^s_c$ and $\bar{W}^s_c$ corresponding to all correct acceptors c are non-empty, then $|\mathsf{Poss}^s| \leq 1$ and $\mathsf{Poss}^s \subseteq \mathsf{Acc}^s$. We will show that this implies that $\mathsf{Good}(w)$ holds for at least one w. If all $\mathsf{Poss}^{\mathcal{S}} = \emptyset$, then $\mathsf{Good}(w)$ for $w = \mathit{input}^r$ because of (**G1**). Otherwise, let s be the highest round number for which $\mathsf{Poss}^s \neq \emptyset$ and w be the only element of Poss^s. Since $w \in \mathsf{Poss}^s \subseteq \mathsf{Acc}^s$, condition (**G2**) implies $\mathsf{Good}(w)$.

4.4 Proving goodness

Every process $q \in \mathcal{A} \cup \{p_r\}$ keeps arrays $\bar{W}^{\mathcal{S}r}_{\mathcal{A}q}$ and $\bar{S}^{\mathcal{S}r}_{\mathcal{A}q}$ such that $\bar{X}^{\mathcal{S}r}_{\mathcal{C}q} \subseteq X^{\mathcal{S}}_{\mathcal{C}}$. These arrays can be used to compute $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}q}]$ as described in Section 4.3. Note that $\bar{X}^{\mathcal{S}r}_{\mathcal{C}q}$ and thus the predicate $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}q}]$ may be different for different q. As a result, a value considered good by p_r will not necessarily be considered good by all acceptors. In this section, we will show how p_r can choose w which will eventually be considered good by all acceptors, so that they will all weakly accept it.

The general version of the algorithm is shown in Figure 10(c). There, proposer p_r can propose only a value w for which $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}](w)$ holds. Each acceptor b maintains a copy $\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_rb}$ of $\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}$. Action $\mathsf{Collect}^{sr}_{ab}$ copies from $\bar{X}^{sr}_{ap_rb}$ to $\bar{X}^{sr}_{ab}$ all elements x for which acceptor b can confirm that $x \in X^s_a$ assuming a is correct. This ensures that $\bar{X}^{sr}_{ab} \subseteq X^s_a$ for all correct a, so $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}b}](w)$ holds only for good w. Moreover, if p_r is correct then every $x \in \bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}$ will eventually be a member of $\bar{X}^{\mathcal{S}r}_{\mathcal{A}b}$. Therefore (Lemma B.6), eventually $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}b}](w)$ will hold for the value w proposed by p_r, so Weak^r_b will be executed.

The definition of $\mathsf{Confirmed}$ and the method of calculating $\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}$ at p_r depend on whether or not public-key cryptography can be used, and are discussed in the next two sections. In both cases, three properties must be guaranteed. First, eventually all $\bar{X}^{\mathcal{S}r}_{\mathcal{C}p_r}$ must be non-empty so that $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}](w)$ will hold for at least one w. Second, the confirmation procedure must be safe, that is, $\mathsf{Confirmed}^r_b(x \in X^s_c) \implies x \in X^s_c$ for all correct acceptors c. Third, liveness requires that if $x \in \bar{X}^{sr}_{ap_r}$ for a correct p_r, then $\mathsf{Confirmed}^r_b(x \in X^s_a)$ will eventually hold. Observe that in the simple solution with $\bar{X}^{sr}_{ap_r} = X^s_{ap_r}$ and $\mathsf{Confirmed}^r_b(x \in X^s_a) \equiv x \in X^s_{ab}$, the last property fails. This is because a faulty acceptor a can easily cause $X^s_a = X^s_{ap_r} = \{x\}$, but $X^s_{ab} = \{y\}$ for some $x \neq y$.

Condition (**G1**) from Section 4.3.2 shows that in order to compute Good, acceptors must also be able to confirm that the proposed value w belongs to I^r. This can be achieved by checking whether $w \in range^r$, because faulty proposers p_r have $I^r = range^r$, whereas proposals from the correct ones do not have to be checked.

4.4.1 With public-key cryptography

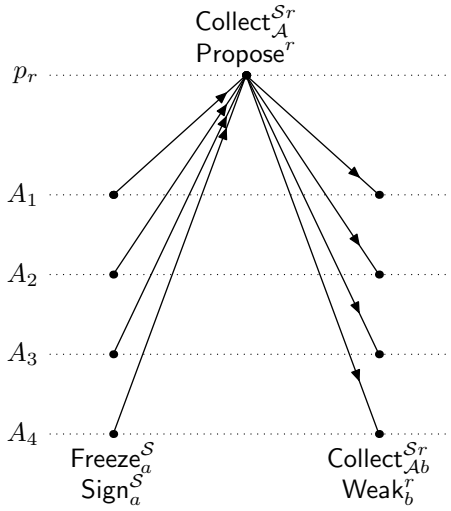
With public-key cryptography, we can make acceptors sign their X^s_a variables, so that confirming "$x \in X^s_a$" would amount to checking the signature. Since such signed X^s_a are needed only for rounds $r > s$, we can defer the signing until round s has been frozen. As a result, no public-key cryptography is used in nice runs.

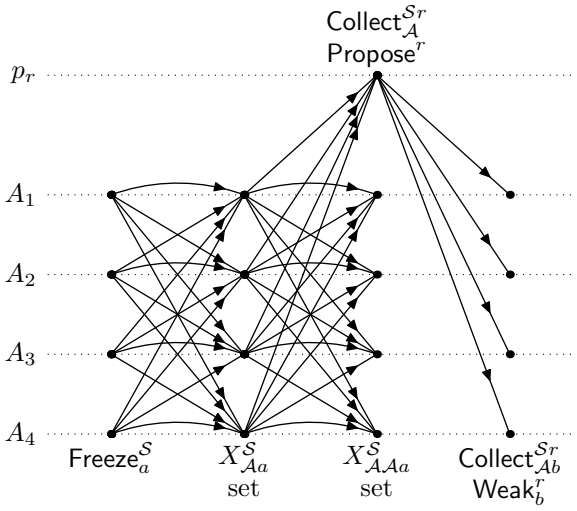
For any set variable V, consider a tuple $\langle x, V \rangle_\sigma$ signed by the owner of V. By signing $\langle x, V \rangle_\sigma$, the owner of V certifies that $x \in V$. Therefore, as long as the owner of V is correct, the existence of $\langle x, V \rangle_\sigma$ implies $x \in V$.

In Figure 10(a), we use auxiliary set variables $\hat{X}^s_a$ to store tuples $\langle x, X^s_a \rangle_\sigma$ signed by a in action Sign^s_a. Proposer p_r executes "$\bar{X}^{sr}_{ap_r} \leftarrow x$" in $\mathsf{Collect}^{sr}_a$ when its copy $\hat{X}^s_{ap_r}$ of $\hat{X}^s_a$ contains $\langle x, X^s_a \rangle_\sigma$. Acceptor b confirms that $x \in X^s_a$ if $\langle x, X^s_a \rangle_\sigma \in \hat{X}^s_{ap_rb}$, where $X^s_{ap_rb}$ is a copy of $X^s_{ap_r}$. For correct acceptors a, this ensures that $x \in X^s_a$.

4.4.2 Without public-key cryptography

Confirming that $x \in X^a_s$ without the proving power of public-key cryptography is more difficult, but possible. In Figure 10(b), the original vectors $X^{\mathcal{S}}_{\mathcal{A}}$ are replicated at all





Name	Guard	Body
$\mathsf{Collect}^{sr}_{a}$	$\langle x, X^s_a\rangle_\sigma \in \hat{X}^s_{ap_r}$	$\bar{X}^{sr}_{ap_r} \leftarrow x$
Sign^{s}_{a}	$x \in X^s_a$	$\hat{X}^s_a \leftarrow \langle x, X^s_a\rangle_\sigma$

$\mathsf{Confirmed}^r_b(x \in X^s_a) \stackrel{\text{def}}{=} \langle x, X^s_a\rangle_\sigma \in \hat{X}^s_{ap_rb}$

(a) with public-key crypto

Name	Guard	Body
$\mathsf{Collect}^{sr}_{a}$	$X^s_{a\mathcal{A}p_r}(x) > 2f$	$\bar{X}^{sr}_{ap_r} \leftarrow x$

$\mathsf{Confirmed}^r_b(x \in X^s_a) \stackrel{\text{def}}{=} X_{a\mathcal{A}b}(x) > f$

(b) without public-key crypto

Name	Guard	Body
$\mathsf{Propose}^{r}$	$\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}](w)$	$P^r \leftarrow w$
$\mathsf{Collect}^{sr}_{ab}$	$x \in \bar{X}^{sr}_{ap_rb}$ and $\mathsf{Confirmed}^r_b(x \in X^s_a)$	$\bar{X}^{sr}_{ab} \leftarrow x$
Weak^{r}_{b}	$w \in P^r_b$ and $\mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}b}](w)$	$W^r_b \leftarrow w$

(c) common code

Figure 10: The code ensuring that only good proposals are issued and accepted.

acceptors a, leading to copies $X^{\mathcal{S}}_{\mathcal{AA}}$. These vectors are replicated again, leading to copies $X^{\mathcal{S}}_{\mathcal{AAA}}$ and $X^{\mathcal{S}}_{\mathcal{AA}p_r}$. An acceptor b confirms that $x \in X^s_a$ if $x \in X^s_{adb}$ for more than f acceptors d, which implies that for correct a the value x indeed belongs to X^s_a. Since $\bar{X}^{sr}_{ap_r}$ is set to x if $x \in X^s_{adp_r}$ for more than $2f$ acceptors d, every $x \in \bar{X}^{sr}_{ap_r}$ will eventually belong to $\bar{X}^{sr}_{ab}$, even if a is faulty.

4.5 Example

In the previous sections, we described the structure of a single round (Section 4.1), freezing a round and starting a new one (Section 4.2), and ensuring that the proposed value is good (Sections 4.3 and 4.4). This section will show how these mechanisms work together.

If the proposer of the first round is correct and the network is timely, then a decision will be reached at this round (Figure 5(b)). Figure 5(a) shows that this will happen even if some of the acceptors are faulty. As a result, the inter-round protocols described in Sections 4.2–4.4 are not necessary in this case.

These mechanisms become useful if the first round proposer is faulty. Figure 11 depicts several scenarios in which proposer p_1 and acceptor A_3 are both faulty. In Figure 11(a), p_1 behaves like a correct proposer, except that it sends a wrong proposal to A_1, which weakly accepts it. The faulty acceptor A_3 sends its correct $W^1_{A_3}$ to all acceptors except for A_1, which makes them strongly accept the correct proposal w. As a result, the faulty acceptor A_3 can make A_2 and A_4 decide on w, while preventing A_1 from doing so. However, the Decide action from Figure 8 will allow A_1 to decide on w as well.

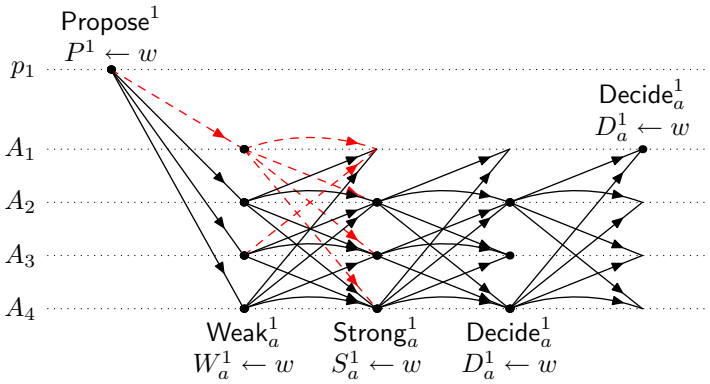
The scenario in Figure 11(b) is similar, except that only A_4 is allowed to decide, which makes the Decide action from Figure 8 inapplicable. Instead, acceptors A_1 and A_2 will eventually time out and locally freeze the first round, which will cause A_4 to do so as well. As a result, p_2 will issue a second round proposal. Both A_2 and A_4 report to have strongly accepted w, and p_2 knows that at least one of them is correct, so the second round proposal must be w. Since p_2 is correct, its proposal will eventually become a decision, despite A_3 being faulty. Note that although (some) acceptors decide both in the first and the second round, these decisions are the same.

Figure 11(c) shows a scenario, in which p_1 sends a (different) wrong proposal to every acceptor, so no value is ever strongly accepted. As a result, the second round proposer p_2 knows that no decision will ever be reached in the first round, so it can propose any value $v \in input^2$. As in the previous example, this value will eventually be decided on, despite malicious efforts of A_3.

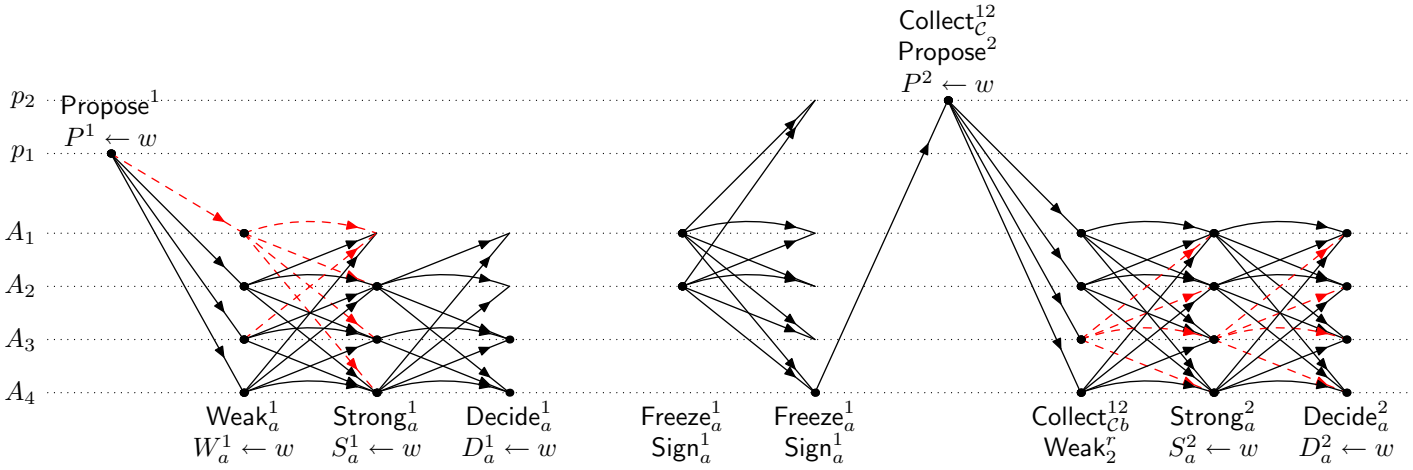
5 Applications: Vector Consensus and Atomic Broadcast

Byzantine Paxos, as described in this paper, can be used to implement Vector Consensus [8]. There, each acceptor a proposes a value $prop[a]$ and the agreement should be reached on a vector $dec[\mathcal{A}]$, with the following Validity conditions:

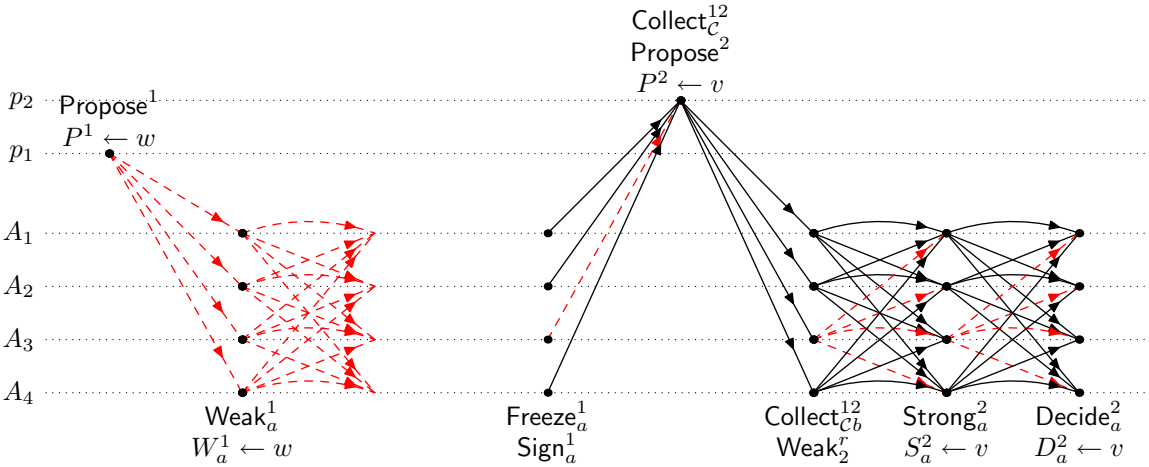
V1: For each correct acceptor c, we have $dec[c] \in \{\triangle, prop[c]\}$, where $\triangle$ is a special symbol meaning "no value".



(a) decision in the first round



(b) identical decisions in the first and the second round



(c) decision in the second round only

Figure 11: An example run with faulty proposer p_1 and acceptor A_3.

Name	Guard	Body
$E_a.\mathsf{Init}_b$	—	$input^1 = \{prop[a]\}$, $range^1 = \mathcal{V}$, $p_1 = a$ $input^r = range^r = \{\triangle\}$ for all $r > 1$
$E_a.\mathsf{Done}_b$	$w \in D_b^r$	set $dec[a]$ at b to w
$E_a.\mathsf{Freeze}_b^1$	... and $E_{a'}.\mathsf{Decide}_b^1$ for at least $n - f$ executions $E_{a'}$	...

Figure 12: Vector Consensus code for execution E_a at acceptor b.

V2: We have $dec[c] = prop[c]$ for at least $n - f - |\mathcal{F}| \geq n - 2f$ correct acceptors c.

V3: If the network is timely, then $dec[c] = prop[c]$ for all correct acceptors c.

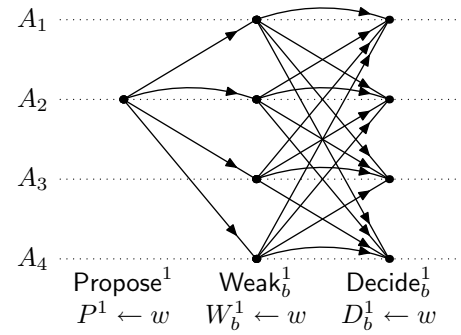
The Byzantine-Paxos-based Vector Consensus algorithm in Figure 12 has a number of advantages over the one presented in [8]. In nice runs, it makes the decision in two steps as opposed to at least four, without using public key cryptography. The bound $n - f - |\mathcal{F}|$ in **V2** is stronger than $n - 2f$ from [8]. In the most common scenario, where none of the acceptors are faulty, $n - f - |\mathcal{F}| = n - f$ acceptors a are guaranteed to have $dec[a] = prop[a]$. Condition **V3** implies that in nice runs $dec[a] = prop[a]$ for all acceptors a.

For each acceptor a, consider an independent execution E_a of Byzantine Paxos running in parallel with all the other executions. The sequence of proposers p_1, p_2, ... for this execution has $p_1 = a$ and contains every acceptor (including a) infinitely many times. For example, we can set $p_r = r \bmod n + 1$ for $r > 1$. Proposer p_1 is allowed to propose any value ($range^1 = \mathcal{V}$) and its initial proposal $input^1$ is $\{prop[a]\}$. Other acceptors can propose only $\triangle$, that is, $input^r = range^r = \{\triangle\}$ for all $r > 1$. The decision reached in E_a is copied to $dec[a]$ at all acceptors b by action $E_a.\mathsf{Done}_b$. The choice of $input^r$ and $range^r$ ensures **V1**.

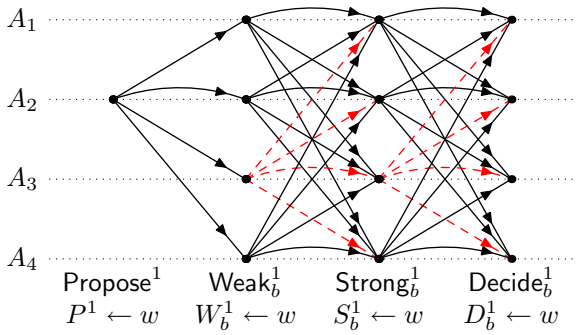
Note that any execution E_c corresponding to a correct acceptor c has a correct first proposer $p_1 = c$. Therefore, if the network is timely, every acceptor a will execute $E_c.\mathsf{Decide}_a^1$ within three communication steps, before Freeze_a^1. As a result, E_c will decide on p_1's proposal ($prop[c]$), which implies **V3**.

We can satisfy **V2** by guaranteeing that at least $n - f$ executions E_a decide in the first round. This can be achieved by not allowing acceptors to locally freeze the first round until they have executed Decide^1 for at least $n - f$ executions $E_{a'}$. This will eventually happen, because at least $n - f$ execution E_a have a correct first-round proposer p_1.

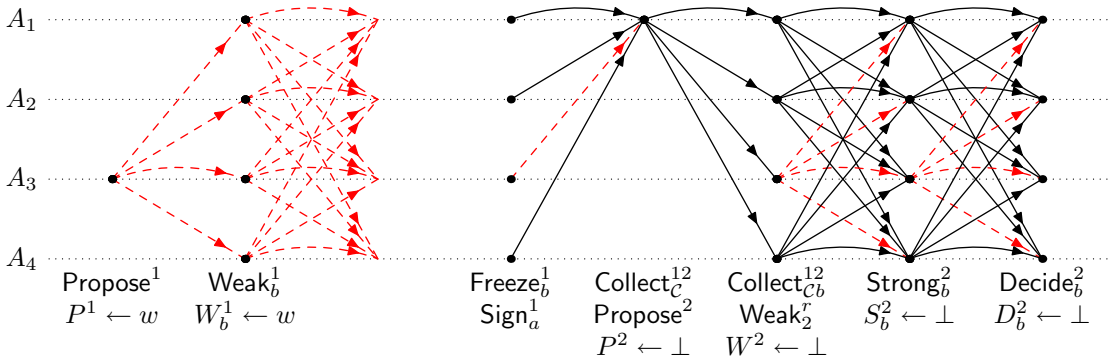
Our Vector Consensus algorithm does not introduce any additional delay to Byzantine Paxos, so in nice runs all executions E_a decide in two communication steps on the value $prop[a]$ proposed by $p_1 = a$. Figure 13(a) shows one of the executions E_a in this scenario. However, if some processes are faulty, then reaching a decision may require more than one round, which normally involves waiting for the first round to time out. In order to improve the performance in such scenarios, we will remove the timeout condition from the guard of $E_a.\mathsf{Freeze}_b^1$ and locally freeze the first round as soon as at least $n - f$ executions E_a have decided. Note that this change may affect **V3** by preventing some of the current acceptors from deciding in the first round.



(a) execution E_2, all correct



(b) execution E_2, acceptor A_3 faulty



(c) execution E_3, acceptor A_3 faulty, $p_2 = A_1$

Figure 13: An example execution of Vector Consensus.

Fortunately, only **V1** and **V2** are required to implement Atomic Broadcast [8]. The method presented in [8] adds one communication step to the underlying Vector Consensus protocol, which in our case leads to the latency of three communication steps in nice runs. By repeating Vector Consensus every $\delta < d$ (where d is the message transmission time) units of time, one can use the techniques for benign settings [21, 24] to achieve the delivery latency of $2d+\delta$. Two-step delivery latency can be achieved in benign settings [24] using one-step Consensus for the most common proposal. In Byzantine settings, however, proposers may be malicious, so Consensus requires two communication steps in any case, making the method from [24] not applicable.

Figures 13(b) and 13(c) depict a scenario with faulty acceptor A_3. Figures 13(b) shows that an execution with a correct proposer (such as A_2) decides in three steps. On the other hand, execution E_3 with a faulty proposer A_3 does not decide in the first round (Figure 13(c)). Instead, it is frozen when all the other executions have decided, and the second round, with a correct proposer A_1, begins.

6 Conclusion and future work

In this paper, we have reduced the latency of the Byzantine Paxos algorithm [5, 17] from three to two communication steps in *nice* runs. At the same time, we still guarantee the

decision in three steps if the leader is correct. As in the original algorithm, each proposal goes through three acceptance stages: weak acceptance, strong acceptance, and decision. The main contribution of this paper lies in showing that the middle stage can be bypassed if more than $\frac{n+3f}{2}$ acceptors have weakly accepted the same value.

If not all processes are correct, it may be necessary to execute more than one round. For this case, we presented two variants of the inter-round protocol. The first variant is simpler than the second but requires computationally expensive public key cryptography. However, these protocols are only needed when failures occur, so their costs may not be very important.

Finally, we proposed new implementations of Vector Consensus and Atomic Broadcast, which use many instances of our Byzantine Paxos algorithm running in parallel. This way, we have achieved a latency that is two communication steps smaller than that in [8]. Another advantage of our solution is that it does not use public-key cryptography.

Our Byzantine Paxos algorithm can be generalized in at least two ways. Firstly, by introducing a third kind of process that can fail only by crashing, it should be possible to reformulate the algorithm presented in this paper to satisfy the lower bounds from [15]. Secondly, one can probably replace f by a safety bound s and a liveness bound l such that the algorithm is safe if $|\mathcal{F}| \leq s$ and live if $|\mathcal{F}| \leq l$. Finally, we hope the ranked register [3, 6] can be generalized to handle Byzantine failures and be used to deconstruct the algorithm presented in this paper in a modular way.

References

[1] H. Attiya and A. Bar-Or. Sharing memory with semi-Byzantine clients and faulty storage servers. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems*, Italy, October 2003.

[2] Michael Barborak, Miroslaw Malek, and Anton Dahbura. The Consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[3] Romain Boichat, Partha Dutta, Svend Frolund, and Rachid Guerraoui. Deconstructing Paxos. Technical Report 200232, École Polytechnique Fédérale de Lausanne, January 2001.

[4] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Symposium on Principles of Distributed Systems (PODC '84)*, pages 154–162, New York, USA, August 1984. ACM, Inc.

[5] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, New Orleans, Louisiana, February 1999. USENIX Association.

[6] G. Chockler and D. Malkhi. Active disk Paxos with infinitely many processes. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, july 2002.

[7] Miguel Correia, Nuno Ferreira Neves, L. C. Lung, and Paulo Veríssimo. Low complexity byzantine-resilient consensus. DI/FCUL TR 03–25, Department of Informatics, University of Lisbon, August 2003.

[8] Assia Doudou and André Schiper. Muteness detectors for Consensus with Byzantine processes. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, pages 315–316, New York, June 1998.

[9] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[10] M. Hurfin and M. Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999.

[11] I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound, 2002.

[12] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. Solving Consensus in a Byzantine environment using an unreliable fault detector. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, pages 61–75, 1997.

[13] Klaus Kursawe. Optimistic asynchronous Byzantine agreement. Technical Report RZ 3202 (#93248), IBM Research, January 2000.

[14] Leslie Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, December 2001.

[15] Leslie Lamport. Lower bounds on Consensus. Unpublished note, March 2002.

[16] Leslie Lamport. Lower bounds on asynchronous Consensus. In André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 22–23. Springer, 2003.

[17] Butler Lampson. The ABCD of Paxos. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. ACM Press, 2001.

[18] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[19] Dahlia Malkhi and Michael Reiter. Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW97)*, pages 116–124, Rockport, MA, 1997.

[20] André Schiper. Early Consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, April 1997.

[21] Pedro Vicente and Luís Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *Proceedings of 21st Symposium on Reliable Distributed Systems (SRDS 2002), 13-16 October 2002, Osaka, Japan*. IEEE Computer Society, 2002.

[22] Marko Vulovic, Partha Dutta, and Rachid Guerraoui. Personal communication.

[23] André Schiper Xavier Défago. Specification of replication techniques, semi-passive replication, and lazy consensus. Technical Report 07, École Polytechnique Fédérale de Lausanne, 2002.

[24] Piotr Zieliński. Latency-optimal Uniform Atomic Broadcast algorithm. Technical Report 582, Computer Laboratory, University of Cambridge, February 2004. Available at `http://www.cl.cam.ac.uk/TechReports/`.

A Safety

As explained in Section 3.2, the stability of the membership in set variables allows one to consider each sequence of implications/inequalities at a single moment of time. On several occasions, however, we will use a special $*$-relations, such as "$\overset{*}{\Longrightarrow}$", where the right-hand side can be calculated for *any* moment of time. For example, for any set variable V we have $w \in V \overset{*}{\Longrightarrow} V \subseteq \{w\}$ because the set V can only grow and cannot contain more than one element. Note that the two arguments of a $*$-relation are treated differently, so the relation is not symmetric. In particular, $A \overset{*}{\Longrightarrow} B$ is not equivalent to $B \overset{*}{\Longleftarrow} A$ and the same applies to other pairs such as $\overset{*}{\leq}$ and $\overset{*}{\geq}$.

Let $V_{\mathcal{A}}$ and $U_{\mathcal{A}}$ be two arrays of set variables such that $U_c \subseteq V_c$ for all $c \in \mathcal{C}$. For example, for any process p, we can have $U_{\mathcal{A}} \stackrel{\text{def}}{=} V_{\mathcal{A}p}$. Then, for any $c \in \mathcal{C}$ and $w \in \mathcal{V}$,

$$w \in U_c \implies w \in V_c \overset{*}{\Longrightarrow} V_c \subseteq \{w\} \implies U_c \subseteq \{w\}.$$

Hence,

$$U_{\mathcal{C}}(w) \leq V_{\mathcal{C}}(w) \overset{*}{\leq} V_{\mathcal{C}}(\hat{w}) \leq U_{\mathcal{C}}(\hat{w}) = U_{\mathcal{C}}(w) + U_{\mathcal{C}}(\circ). \tag{1}$$

Also,

$$V_{\mathcal{A}}(w) \geq V_{\mathcal{C}}(w) \underbrace{\geq}_{(1)} U_{\mathcal{C}}(w) = U_{\mathcal{A}}(w) - U_{\mathcal{F}}(w) \geq U_{\mathcal{A}}(w) - t \geq U_{\mathcal{A}}(w) - f. \tag{2}$$

Lemma A.1. *For any array $V_{\mathcal{A}}$, there is at most one x for which $V_{\mathcal{C}}(x) > \frac{1}{2}|\mathcal{C}|$.*

Proof. If $V_{\mathcal{C}}(x) > \frac{1}{2}|\mathcal{C}|$ holds for $x \in \{y, z\}$, where $y \neq z$, then

$$|\mathcal{C}| \geq V_{\mathcal{C}}(y) + V_{\mathcal{C}}(z) > \tfrac{1}{2}|\mathcal{C}| + \tfrac{1}{2}|\mathcal{C}| = |\mathcal{C}|. \qquad \square$$

We will prove the safety of our algorithm by induction on the round number. Assuming that all theorems from this section hold for all rounds $s < r$, we will prove them for round r.

A.1 Definitions

Here, we repeat some important definitions from the paper. A value w is *admissible* at round r if no decision other than w can be made in any round $s < r$. A value is *valid* if it belongs to I^s for some $s \leq r$. Finally, a value is *good* if it is both valid and admissible.

Recall that

$$\mathsf{Acc}^r = \{\, w \in \mathcal{V} : \bar{W}^r_{\mathcal{A}}(w) > f \,\}$$

and $\mathsf{Poss}^r = \mathsf{PS}^r \cup \mathsf{PW}^r$, where

$$\mathsf{PS}^r = \big\{\, w \in \mathcal{V} : \bar{S}^r_{\mathcal{A}}(\hat{w}) > f \,\big\}, \quad \mathsf{PW}^r = \big\{\, w \in \mathcal{V} : \bar{W}^r_{\mathcal{A}}(\hat{w}) > \tfrac{n+f}{2} \,\big\}.$$

Finally, the predicate $\mathsf{Good}(w)$ to be true if at least one of the following conditions is met:

$$w \in I^r \text{ and } \mathsf{Poss}^s = \emptyset \text{ for all } s < r, \tag{3}$$

$$\text{there is } s < r \text{ such that } w \in \mathsf{Acc}^s \text{ and } \mathsf{Poss}^t \subseteq \{w\} \text{ for all } t : s \leq t < r. \tag{4}$$

A.2 Predicate $\mathsf{Good}(w)$

Lemma A.2. *If* $\mathsf{Confirmed}^r_b(w \in X^s_c)$ *for* $c \in \mathcal{C}$*, then* $w \in X^s_c$*.*

Proof. With public-key cryptography:

$$\mathsf{Confirmed}^r_b(w \in X^s_c) \implies \langle w, X^s_c \rangle_\sigma \in \hat{X}^s_{cp_rb} \implies w \in X^s_c.$$

Without public-key cryptography:

$$\mathsf{Confirmed}^r_b(w \in X^s_c) \implies X^s_{c\mathcal{A}b}(w) > f \implies X^s_{c\mathcal{C}b}(w) > 0 \implies X^s_{c\mathcal{C}}(w) > 0 \implies w \in X^s_c.$$

□

Lemma A.3. *If* $\bar{X}^{\mathcal{S}}_{\mathcal{C}} \subseteq X^{\mathcal{S}}_{\mathcal{C}}$*, then* $\mathsf{Good}[\bar{X}^{\mathcal{S}}_{\mathcal{C}}](w)$ *implies that* w *is admissible.*

Proof. The assertion implies one of the following:

(1) For all rounds $s < r$ we have $\mathsf{Poss}^s = \emptyset$. By Lemma A.7 for round s, no decision is ever made at any round $s < r$.

(2) There is $s < r$ such that $w \in \mathsf{Acc}^s$ and $\mathsf{Poss}^t \subseteq \{w\}$ for all $t : s \leq t < r$. On the one hand, Lemma A.6 applied to Acc^s implies that no decision other than w is ever made at any round $t < s$. On the other hand, Lemma A.7 applied to Poss^t implies the same for $t : s \leq t < r$. □

Lemma A.4. *If* $\bar{X}^{\mathcal{S}}_{\mathcal{C}} \subseteq X^{\mathcal{S}}_{\mathcal{C}}$*, then* $\mathsf{Good}[\bar{X}^{\mathcal{S}}_{\mathcal{C}}](w)$ *implies that* w *is valid.*

Proof. The assertion implies one of the following:

(1) Value $w \in I^r$. Then, $w \in I^r \subseteq \bigcup_{t \leq r} I^t$.

(2) Value $w \in \mathsf{Acc}^s$ for some $s < r$. Then,

$$w \in \mathsf{Acc}^s \implies \bar{W}^s_{\mathcal{A}}(w) > f \implies \bar{W}^s_{\mathcal{C}}(w) > 0 \implies W^s_{\mathcal{C}}(w) > 0 \implies w \in W^s_b$$

for some b, so from the inductive assumption (this lemma for round s) we have $w \in \bigcup_{t \le s} I^t \subseteq \bigcup_{t \le r} I^t$. □

Lemma A.5. *If $w \in W^r_a$ for any $a \in \mathcal{A}$, then w is good.*

Proof. First, for any x and any $c \in \mathcal{C}$ we have

$$x \in X^{sr}_{ca} \implies \mathsf{Confirmed}^r_a(x \in X^s_c) \underbrace{\implies}_{\text{Lemma A.2}} x \in X^s_c.$$

Therefore $\bar{X}^{\mathcal{S}r}_{\mathcal{C}a} \subseteq X^{\mathcal{S}}_{\mathcal{C}}$, so Lemmas A.3 and A.4 prove the assertion. □

A.3 Sets Poss^s and Acc^s

Assume that for any $X \in \{S, W\}$ we have $\bar{X}^r_c \subseteq X^r_c$ for all $c \in \mathcal{C}$. Therefore, we can use rules (1) and (2) from the beginning of Section A.

Lemma A.6. *If $w \in \mathsf{Acc}^r$, then w is good at round r.*

Proof. $W^r_{\mathcal{A}}(w) \ge \bar{W}^r_{\mathcal{A}}(w) - f > 0$, so Lemma A.5 implies the assertion. □

Lemma A.7. *If $w \notin \mathsf{Poss}^r$, then no acceptor a ever executes $\mathsf{Decide}^r_a(w)$.*

Proof. We will use the $*$-version of $\le$ defined at the beginning of Appendix A. To obtain a contradiction, assume that the left-hand side of "$\overset{*}{\le}$" is taken at the time when a hypothetically executes $\mathsf{Decide}^r_a(w)$, and the right-hand side is taken at the "current" time when $w \notin \mathsf{Poss}^r$. Since $w \notin \mathsf{PW}^r$,

$$W^r_{\mathcal{A}a}(w) \le W^r_{\mathcal{C}a}(w) + f \le W^r_{\mathcal{C}}(w) + f \overset{*}{\le} \bar{W}^r_{\mathcal{C}}(\hat{w}) + f \le \bar{W}^r_{\mathcal{A}}(\hat{w}) + f \le \tfrac{n+f}{2} + f = \tfrac{n+3f}{2}.$$

Similarly, $w \notin \mathsf{PS}^r$ implies

$$S^r_{\mathcal{A}a}(w) \le S^r_{\mathcal{C}a}(w) + f \le S^r_{\mathcal{C}}(w) + f \overset{*}{\le} \bar{S}^r_{\mathcal{C}}(\hat{w}) + f \le \bar{S}^r_{\mathcal{A}}(\hat{w}) + f \le f + f = 2f.$$ □

A.4 Validity

Lemma A.8. *If $w \in S^r_a$, then $w \in W^r_b$ for some acceptor b.*

Proof.

$$w \in S^r_a \implies W^r_{\mathcal{A}a}(w) > \tfrac{n+f}{2} \implies W^r_{\mathcal{A}}(w) > \tfrac{n+f}{2} - f > 0 \implies w \in W^r_b.$$ □

Lemma A.9. *If $w \in D^r_a$, then $w \in W^r_b$ for some acceptor b.*

Proof.

$$w \in D^r_a \implies \begin{cases} W^r_{\mathcal{A}a}(w) > \frac{n+3f}{2} \implies W^r_{\mathcal{A}}(w) > \frac{n+3f}{2} - f > 0 \implies w \in W^r_b, \text{ or} \\ S^r_{\mathcal{A}a}(w) > 2f \implies S^r_{\mathcal{A}}(w) > 2f - f \ge 0 \end{cases} \underbrace{\implies}_{\text{Lemma A.8}} w \in W^r_b.$$ □

Lemma A.10. *If $w \in D^r_a$, then w is good.*

Proof. Lemma A.9 implies that $w \in W^r_b$ for some $b \in \mathcal{A}$, so by Lemma A.5 w is good. □

Corollary A.11 (Validity). *If $w \in D^r_a$, then w is valid.*

A.5 Agreement

Lemma A.12. *If* $W^r_{\mathcal{A}a}(v) > \frac{n+f}{2}$ *and* $W^r_{\mathcal{A}b}(w) > \frac{n+f}{2}$*, then* $v = w$*.*

Proof. For $(z, d) \in \{(v, a), (w, b)\}$, we have

$$W^r_{\mathcal{C}}(z) \geq W^r_{\mathcal{C}d}(z) \geq W^r_{\mathcal{A}d}(z) - |\mathcal{F}| > \tfrac{n+f}{2} - |\mathcal{F}| = \tfrac{1}{2}n + \tfrac{1}{2}f - \tfrac{1}{2}|\mathcal{F}| - \tfrac{1}{2}|\mathcal{F}| \geq \tfrac{1}{2}n - \tfrac{1}{2}|\mathcal{F}| = \tfrac{1}{2}|\mathcal{C}|.$$

Lemma A.1 implies that $w = v$. □

Corollary A.13. *If* $v \in S^r_a$ *and* $w \in S^r_b$ *for* $v, w \in \mathcal{V}$*, then* $v = w$*.*

Lemma A.14. *If* $v \in D^r_a$ *and* $w \in D^r_b$*, then* $w = v$*.*

Proof. For $(z, d) \in \{(v, a), (w, b)\}$, we have

$$W^r_{\mathcal{A}d}(z) > \tfrac{n+3f}{2} \qquad \text{or} \qquad S^r_{\mathcal{A}d}(z) > 2f.$$

The former case obviously implies $W^r_{\mathcal{A}d'}(z) > \frac{n+f}{2}$. In the latter case,

$$S^r_{\mathcal{A}}(z) \geq S^r_{\mathcal{C}}(z) \geq S^r_{\mathcal{C}d}(v) \geq S^r_{\mathcal{A}d}(v) - f > 2f - f \geq 0,$$

hence $z \in S^r_{d'}$ for some acceptor d', so $W^r_{\mathcal{A}d'}(z) > \frac{n+f}{2}$ as well. The assertions follows from Lemma A.12. □

Theorem A.15 (Agreement). *If* $w \in D^r_a$ *and* $v \in D^s_b$ *for some* $s \leq r$*, then* $v = w$*.*

Proof. If $s = r$, then it follows from Lemma A.14. Otherwise ($s < r$) from Lemma A.10. □

B Liveness

Lemma B.1. *If* V *is a set variable such that "*$V \leftarrow x$*" can be executed for at most one* x*, then "*$V \leftarrow x$*" results in* $x \in V$*.*

Proof. The "$\leftarrow$" operation is the only way of modifying V, so $V \subseteq \{x\}$ at any time. Executing "$V \leftarrow x$" results in $V = \{x\}$ both if $V = \emptyset$ and if $V = \{x\}$. □

Corollary B.2. *If "*$V \leftarrow x$*" can be executed only if* $x \in U$*, where* V *and* U *are set variables, then "*$V \leftarrow x$*" results in* $x \in V$*.*

Corollary B.3. *If* V *is copy of a variable owned by a correct process, then then "*$V \leftarrow x$*" results in* $x \in V$*.*

In addition to the normal implication "$A \implies B$", we introduce three "timed" implications "$A \overset{ev}{\implies} B$", "$A \overset{d}{\implies} B$", and "$A \overset{\Delta}{\implies} B$", which all state that A implies that B will eventually be true.

The first implication ("$\overset{ev}{\implies}$") is used when the delay is caused by waiting for an enabled action to be executed (e.g., $W^r_{\mathcal{A}a}(w) > \frac{n+f}{2} \overset{ev}{\implies} \mathsf{Strong}^r_a$). If the network is timely, we assume that such an implication introduces no delay.

The second implication ("$\overset{d}{\Longrightarrow}$") is used when a single message passing is involved. In this paper, we use it exclusively for reasoning about updating copies of a set variable V owned by a correct process p:

$$x \in V \overset{ev}{\Longrightarrow} \mathsf{Spread}_p \overset{d}{\Longrightarrow} \mathsf{Update}_{\mathcal{A}} \underbrace{\Longrightarrow}_{\text{Corollary B.3}} x \in V_{\mathcal{A}}$$

If the network is timely, we assume that such an implication introduces a delay smaller than d. Therefore, when reasoning about the performance, it is sufficient to count the number of "$\overset{d}{\Longrightarrow}$" operations in the sequence of implications.

Finally, the third implication ("$\overset{\Delta}{\Longrightarrow}$") means that the right-hand side will become true no later than Δ units of time after the left-hand side, where Δ is the timeout.

B.1 Proving goodness

Lemma B.4. *If $x \in X^s_c$ for $c \in \mathcal{C}$, then eventually $x \in \bar{X}^{sr}_{cp_r}$.*

Proof. With public-key cryptography:

$$x \in X^s_c \overset{ev}{\Longrightarrow} \mathsf{Sign}^s_c \underbrace{\Longrightarrow}_{\text{Cor. B.2}} \langle x, X^s_c \rangle_\sigma \in \hat{X}^s_c \overset{d}{\Longrightarrow} \langle x, X^s_c \rangle_\sigma \in \hat{X}^s_{cp_r} \overset{ev}{\Longrightarrow} \mathsf{Collect}^{sr}_c \underbrace{\Longrightarrow}_{\text{Cor. B.2}} x \in \bar{X}^{sr}_{cp_r}.$$

Without public-key cryptography (for any $y \neq x$):

$$x \in X^s_c \overset{d}{\Longrightarrow} x \in X^s_{c\mathcal{C}} \overset{d}{\Longrightarrow} x \in X^s_{c\mathcal{C}p_r} \Longrightarrow X^s_{c\mathcal{A}p_r}(y) \leq X^s_{c\mathcal{F}p_r}(y) \leq f \leq 2f,$$

so action $\mathsf{Collect}^{sr}_c$ can only be applied to x. Therefore,

$$x \in X^s_{c\mathcal{C}p_r} \Longrightarrow X^s_{c\mathcal{A}p_r}(x) \geq X^s_{c\mathcal{C}p_r}(x) \geq n - f > 2f \overset{ev}{\Longrightarrow} \mathsf{Collect}^{sr}_c \underbrace{\Longrightarrow}_{\text{Lemma B.1}} x \in \bar{X}^{sr}_{cp_r}.$$

□

Lemma B.5. *If p_r is correct and $x \in \bar{X}^{sr}_{ap_r}$, then eventually $\mathsf{Confirmed}^r_{\mathcal{A}}(x \in X^s_a)$.*

Proof. With public-key cryptography:

$$x \in \bar{X}^{sr}_{ap_r} \Longrightarrow \langle x, X^s_a \rangle_\sigma \in \hat{X}^s_{ap_r} \overset{d}{\Longrightarrow} \langle x, X^s_a \rangle_\sigma \in \hat{X}^s_{ap_r\mathcal{A}} \Longrightarrow \mathsf{Confirmed}^r_{\mathcal{A}}(x \in X^s_a).$$

Without public-key cryptography:

$$x \in \bar{X}^{sr}_{ap_r} \Longrightarrow X^s_{a\mathcal{A}p_r}(x) > 2f \Longrightarrow X^s_{a\mathcal{C}p_r}(x) > f \Longrightarrow X^s_{a\mathcal{C}}(x) > f \overset{d}{\Longrightarrow}$$
$$X^s_{a\mathcal{C}\mathcal{A}}(x) > f \Longrightarrow \mathsf{Confirmed}^r_{\mathcal{A}}(x \in X^s_a).$$ □

Lemma B.6. *If $\bar{X}^{\mathcal{S}}_{\mathcal{A}} \subseteq \bar{X}'^{\mathcal{S}}_{\mathcal{A}}$, then $\mathsf{Good}[\bar{X}^{\mathcal{S}}_{\mathcal{A}}](w) \Longrightarrow \mathsf{Good}[\bar{X}'^{\mathcal{S}}_{\mathcal{A}}](w)$.*

Proof. First, note that for any w the assumption implies

$$\bar{X}^{\mathcal{S}}_{\mathcal{A}}(w) \le \bar{X}'^{\mathcal{S}}_{\mathcal{A}}(w), \qquad \bar{X}^{\mathcal{S}}_{\mathcal{A}}(\hat{w}) \le \bar{X}'^{\mathcal{S}}_{\mathcal{A}}(\hat{w}).$$

Hence,

$$\mathsf{Poss}^s[\bar{X}^s_{\mathcal{A}}] \supseteq \mathsf{Poss}^s[\bar{X}'^s_{\mathcal{A}}], \qquad \mathsf{Acc}^s[\bar{X}^s_{\mathcal{A}}] \subseteq \mathsf{Acc}^s[\bar{X}'^s_{\mathcal{A}}].$$

The assertion follows from the definition of Good. □

Lemma B.7. *If $\bar{X}_{\mathcal{C}} \neq \emptyset$, then $|\mathsf{Poss}| \le 1$ and $\mathsf{Poss} \subseteq \mathsf{Acc}$.*

Proof. $\bar{S}_{\mathcal{C}} \neq \emptyset$, so $\bar{S}_{\mathcal{A}}(\circ) = \bar{S}_{\mathcal{C}}(\circ) + \bar{S}_{\mathcal{F}}(\circ) = \bar{S}_{\mathcal{F}}(\circ)$. For any $w \in \mathsf{PS}$:

$$S_{\mathcal{A}}(w) \ge S_{\mathcal{C}}(w) \ge \bar{S}_{\mathcal{C}}(w) = \bar{S}_{\mathcal{A}}(w) - \bar{S}_{\mathcal{F}}(w) \ge \bar{S}_{\mathcal{A}}(w) - \bar{S}_{\mathcal{F}}(\bullet) \ge \bar{S}_{\mathcal{A}}(\hat{w}) - \bar{S}_{\mathcal{A}}(\circ) - \bar{S}_{\mathcal{F}}(\bullet) =$$
$$= \bar{S}_{\mathcal{A}}(\hat{w}) - \bar{S}_{\mathcal{F}}(\circ) - \bar{S}_{\mathcal{F}}(\bullet) = \bar{S}_{\mathcal{A}}(\hat{w}) - |\mathcal{F}| \ge \bar{S}_{\mathcal{A}}(\hat{w}) - f > 0,$$

so there is $a \in \mathcal{A}$ such that $w \in S_a$. Hence,

$$\bar{W}_{\mathcal{C}}(\hat{w}) \ge W_{\mathcal{C}}(w) \ge W_{\mathcal{C}a}(w) \ge W_{\mathcal{A}a}(w) - |\mathcal{F}| > \tfrac{n+f}{2} - |\mathcal{F}|.$$

The same is true for any $w \in \mathsf{PW}$ because

$$\bar{W}_{\mathcal{C}}(\hat{w}) = \bar{W}_{\mathcal{A}}(\hat{w}) - \bar{W}_{\mathcal{F}}(\hat{w}) \ge \bar{W}_{\mathcal{A}}(\hat{w}) - |\mathcal{F}| > \tfrac{n+f}{2} - |\mathcal{F}|.$$

Now, $\bar{W}_{\mathcal{C}} \neq \emptyset \implies \bar{W}_{\mathcal{C}}(\circ) = 0$, so for any $w \in \mathsf{Poss} = \mathsf{PW} \cup \mathsf{PS}$:

$$\bar{W}_{\mathcal{C}}(w) = \bar{W}_{\mathcal{C}}(w) + \bar{W}_{\mathcal{C}}(\circ) = \bar{W}_{\mathcal{C}}(\hat{w}) > \tfrac{n+f}{2} - |\mathcal{F}| \underbrace{\ge}_{\text{see Lemma A.12}} \tfrac{1}{2}|\mathcal{C}|.$$

Lemma A.1 applied to $\bar{W}_{\mathcal{C}}$ gives $|\mathsf{Poss}| \le 1$. Moreover, $\mathsf{Poss} \subseteq \mathsf{Acc}$ because

$$w \in \mathsf{Poss} \implies \bar{W}_{\mathcal{A}}(w) \ge \bar{W}_{\mathcal{C}}(w) > \tfrac{n+f}{2} - |\mathcal{F}| \ge \tfrac{n-f}{2} > f \implies w \in \mathsf{Acc}.$$ □

Lemma B.8. *If $X^{\mathcal{S}}_{\mathcal{C}} \neq \emptyset$, then there is w for which $\mathsf{Good}[X^{\mathcal{S}}_{\mathcal{C}}]$ holds.*

Proof. Lemma B.7 implies $|\mathsf{Poss}^{\mathcal{S}}| \le 1$ and $\mathsf{Poss}^{\mathcal{S}} \subseteq \mathsf{Acc}^{\mathcal{S}}$. If all $\mathsf{Poss}^{\mathcal{S}} = \emptyset$, then $w = \mathit{input}^r$ satisfies Condition (3). Otherwise, Condition (4) is met by the single element w of Poss^s, where $s < r$ is the highest possible with $\mathsf{Poss}^s \neq \emptyset$. □

B.2 Single round with a correct proposer

Let r be a round which has a correct proposer p_r and is never frozen. Assuming that all round $s < r$ have been frozen, we will prove that a decision will eventually be reached at round r.

Lemma B.9. *If every correct acceptor executes Freeze^s for all $s < r$, then eventually p_r will execute $\mathsf{Propose}^r$.*

Proof.

$$\mathsf{Freeze}^{\mathcal{S}}_{\mathcal{C}} \implies X^{\mathcal{S}}_{\mathcal{C}} \neq \emptyset \underbrace{\overset{d}{\implies}}_{\text{Lemma B.4}} \bar{X}^{\mathcal{S}r}_{\mathcal{C}p_r} \neq \emptyset \underbrace{\implies}_{\text{Lemma B.8}} \exists\, w : \mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{C}p_r}](w) \overset{ev}{\implies} \mathsf{Propose}^r.$$

□

Lemma B.10. *If $w \in W^r_a$, then p_r has proposed w.*

Proof.

$$w \in W^r_a \implies w \in P^r_a \underbrace{\implies}_{p_r \text{ is correct}} w \in P^r.$$

□

Lemma B.11. *If p_r proposes w, then eventually $w \in W^r_b$ for all acceptors b.*

Proof. First, for any x we have

$$x \in \bar{X}^{sr}_{ap_r} \underbrace{\overset{d}{\implies}}_{\text{Lemma B.5}} \left\{ \begin{array}{c} x \in \bar{X}^{sr}_{ap_rb} \\ \mathsf{Confirmed}^r_b(x \in X^s_a) \end{array} \right\} \overset{ev}{\implies} \mathsf{Collect}^{sr}_{ab} \underbrace{\implies}_{\text{Cor. B.2}} x \in \bar{X}^{sr}_{ab}.$$

Therefore,

$$w \in P^r \implies \left\{ \begin{array}{c} \overset{d}{\implies} \quad w \in P^r_b \\ \mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}p_r}](w) \underbrace{\overset{d}{\implies}}_{\text{Lemma B.6}} \mathsf{Good}[\bar{X}^{\mathcal{S}r}_{\mathcal{A}b}](w) \end{array} \right\} \overset{ev}{\implies} \mathsf{Weak}^r_b \underbrace{\implies}_{\text{Cor. B.2}} w \in W^r_b.$$

□

Lemma B.12. *If p_r proposes w, then every acceptor will decide on w in three communication steps.*

Proof. Lemma B.2 for W, S, and D implies

$$\mathsf{Propose}^r \underbrace{\overset{d}{\implies}}_{\text{Lemma B.11}} w \in W^r_{\mathcal{A}} \overset{d}{\implies}$$

$$w \in W^r_{\mathcal{C}\mathcal{A}} \implies W^r_{\mathcal{A}\mathcal{A}}(w) \geq |\mathcal{C}| \geq n - f > \tfrac{n+f}{2} \overset{ev}{\implies} \mathsf{Strong}^r_{\mathcal{A}} \implies w \in S^r_{\mathcal{A}} \overset{d}{\implies}$$

$$w \in S^r_{\mathcal{C}\mathcal{A}} \implies S^r_{\mathcal{A}\mathcal{A}}(w) \geq |\mathcal{C}| \geq n - f > 2f \overset{ev}{\implies} \mathsf{Decide}^r_{\mathcal{A}} \implies w \in D^r_{\mathcal{A}}.$$ □

Lemma B.13. *If p_r proposes w and more than $\frac{n+3f}{2} < n$ acceptors are correct, then every acceptor will decide on w in* two *communication steps.*

Proof. Lemma B.2 for W and D implies

$$\mathsf{Propose}^r \underbrace{\overset{d}{\implies}}_{\text{Lemma B.11}} w \in W^r_{\mathcal{A}} \overset{d}{\implies} w \in W^r_{\mathcal{C}\mathcal{A}} \implies W^r_{\mathcal{A}\mathcal{A}}(w) \geq |\mathcal{C}| > \tfrac{n+3f}{2} \overset{ev}{\implies} \mathsf{Decide}^r_{\mathcal{A}}.$$

□

B.3 Whole algorithm

In this section, we assume that the network is eventually timely and the timeout $\Delta > 5d$. We will prove that there is a round r at which all acceptors decide. For the sake of contradiction assume that such a round does not exist. This is equivalent to saying that $D^r_{\mathcal{C}}(\bullet) \leq f$ for all rounds r, because otherwise (for some w)

$$D^r_{\mathcal{C}}(\bullet) > f \underbrace{\implies}_{\text{Lemma A.14}} D^r_{\mathcal{C}}(w) > f \overset{d}{\implies} D^r_{\mathcal{A}\mathcal{A}}(w) > f \underbrace{\overset{ev}{\implies}}_{\text{Cor. B.2}} \mathsf{Decide}^r_{\mathcal{A}} \implies w \in D^r_{\mathcal{A}},$$

which contradicts the assumption that not all acceptors decide at round r.

Definition B.14. *Let $\mathcal{U}^r \subseteq \mathcal{C}$ be the set of all* correct *acceptors c that never decide at round r, that is, for which D^r_c always equals $\emptyset$.*

We have

$$|\mathcal{U}^r| = D^r_{\mathcal{C}}(\circ) = |\mathcal{C}| - D^r_{\mathcal{C}}(\bullet) \geq n - f - f > f.$$

Definition B.15. *Round r has* started *if for every $s < r$, more than f correct acceptors have executed Freeze^s. We denote the moment when r starts by an artificial action Start^r.*

Lemma B.16. *For every started round r, all acceptors a will eventually execute Freeze^s_a for all $s < r$.*

Proof.

$$\mathsf{Start}^r \equiv \mathsf{Freeze}^s_{\mathcal{C}}(\bullet) > f \overset{d}{\implies} \mathsf{Freeze}^s_{\mathcal{A}\mathcal{A}}(\bullet) \geq \mathsf{Freeze}^s_{\mathcal{C}\mathcal{A}}(\bullet) > f \overset{ev}{\implies} \mathsf{Freeze}^s_{\mathcal{A}}. \qquad \square$$

Lemma B.17. *Every round r will eventually start.*

Proof. By contradiction. Let r be the lowest round for which Start^r is never executed. For all $t < s < r$, we have

$$\mathsf{Start}^s \underbrace{\overset{d}{\implies}}_{\text{Lemma B.16}} \mathsf{Freeze}^t_{\mathcal{A}} \overset{d}{\implies} \mathsf{Freeze}^t_{\mathcal{C}\mathcal{A}} \implies \mathsf{Freeze}^t_{\mathcal{A}\mathcal{A}}(\bullet) \geq |\mathcal{C}| \geq n - f > 2f \overset{ev}{\implies}$$
$$\mathsf{Clock}^s_{\mathcal{A}} \implies \mathsf{Clock}^s_{\mathcal{U}^s} \overset{\Delta}{\implies} \mathsf{Freeze}^s_{\mathcal{U}^s} \implies \mathsf{Freeze}^s_{\mathcal{C}}(\bullet) > f \implies \mathsf{Start}^r. \qquad \square$$

Lemma B.18. *Action Clock^r_a cannot be executed before Start^r.*

Proof. For any $s < r$:

$$\mathsf{Clock}^r_a \implies \mathsf{Freeze}^s_{\mathcal{A}a}(\bullet) > 2f \implies \mathsf{Freeze}^s_{\mathcal{C}}(\bullet) > f \equiv \mathsf{Start}^r. \qquad \square$$

Lemma B.19. *Round r cannot start earlier than Δ after all rounds $s < r$ started.*

Proof. For any $s < r$, assume round s started at time t^s. Lemma B.18 states that for no acceptor a can Clock^s_a be executed before time t^s. Thus, no correct acceptor a executes Freeze^r_a before $t^s + \Delta$, which proves the assertion. $\square$

Theorem B.20 (Termination). *There is a round r, in which all correct acceptors decide.*

Proof. Lemmas B.17 and B.19 imply that there is r_0 such that the network stabilizes before round r_0 starts. Let $r \geq r_0$ be a round with a correct proposer. Consider a modified algorithm, in which round r cannot be frozen. Thus, for every $s < r$,

$$\mathsf{Start}^r \underbrace{\overset{d}{\Longrightarrow}}_{\text{Lemma B.16}} \mathsf{Freeze}^s_{\mathcal{A}} \underbrace{\overset{d}{\Longrightarrow}}_{\text{Lemma B.9}} \mathsf{Propose}^r \underbrace{\overset{d}{\Longrightarrow} \overset{d}{\Longrightarrow} \overset{d}{\Longrightarrow}}_{\text{Lemma B.12}} \mathsf{Decide}^r_{\mathcal{A}}.$$

Assuming that round r starts at time t, the above series of implications shows that the the modified algorithm decides by $t + 5d$. Lemma B.18 implies that round r will not be frozen before $t + \Delta > t + 5d$. The assertion holds because modified algorithm is indistinguishable from the original one until then. □