A Leader-free Byzantine Consensus Algorithm [*]

Fatemeh Borran and André Schiper

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract. The paper considers the consensus problem in a partially synchronous system with Byzantine faults. It turns out that, in the partially synchronous system, all deterministic algorithms that solve consensus with Byzantine faults are leader-based. This is not the case of benign faults, which raises the following fundamental question: is it possible to design a deterministic Byzantine consensus algorithm for a partially synchronous system that is not leader-based? The paper gives a positive answer to this question, and presents a leader-free algorithm that is resilient-optimal and signature-free.

1 Introduction

In a distributed system of n processes, where each process has an initial value, Byzantine consensus is the problem of agreeing on a common value, even though some of the processes may fail in arbitrary, even malicious, ways. Consensus is related to the implementation of state machine replication, atomic broadcast, etc. It was first identified by Pease, Shostak and Lamport [1], formalized as the *interactive consistency* problem and solved in a synchronous system. An algorithm achieves interactive consistency if it allows the nonfaulty processes to come to a consistent view of the initial values of all the processes, including the faulty ones. Once interactive consistency has been achieved, the nonfaulty processes can reach consensus by applying a deterministic averaging or filtering function on the values of their view. It is shown in [1] that in a synchronous system $3t + 1$ processes are needed to solve the Byzantine consensus problem without signatures, where t is the maximum number of Byzantine processes.

Later, Fischer, Lynch and Peterson [2] proved that in an asynchronous system, no deterministic asynchronous consensus protocol can tolerate even a single crash failure. The problem can however be solved using randomization even with Byzantine faults, with at least $5t + 1$ processes, as shown by BenOr [3] and Rabin [4]. Later, Bracha [5] increased the resiliency of the randomized algorithm to $3t + 1$ using a "reliable broadcast" primitive.

In 1988, Dwork, Lynch and Stockmeyer [6], considered an asynchronous system that eventually becomes synchronous (called *partially synchronous system*). The consensus algorithms proposed in [6], for benign and for Byzantine faults, achieve safety in all executions, while guaranteeing liveness only if there exists

[*] Research funded by the Swiss National Science Foundation under grant number 200021-111701.

a period of synchrony. Recently, several papers have considered the partially synchronous system model for Byzantine consensus [7–10].

However, [10] points out a potential weakness of these Byzantine consensus algorithms, namely that they can suffer from "performance failure". According to [10], a performance failure occurs when messages are sent slowly by a Byzantine leader, but without triggering protocol timeouts, and the paper points out that the PBFT leader-based algorithm [7] is vulnerable to such an attack. Similar arguments are mentioned in [11] and in [12], where Lamport suggests the use of a virtual leader.

Interestingly, all deterministic Byzantine algorithms for non-synchronous systems are leader-based, e.g., [6–9]. Even the protocol in [10] is leader-based. However, the authors of [10] managed to make the leader-based protocol less vulnerable to performance failure attacks than PBFT [7] through a complicated mechanism that enables non-leader processes to (i) aggressively monitor the leader's performance, and (ii) compute a threshold level of acceptable performance. Note that randomized consensus algorithms such as [3, 4] are not leader-based. This raises the following fundamental question: is it possible to design a deterministic Byzantine consensus algorithm for a partially synchronous system that is not leader-based? With such an algorithm, performance failure of Byzantine processes might be harmless.

One may imagine that leader-free (non-leader-based) algorithms for benign faults might be extended for Byzantine faults. A leader-free algorithm typically consists of a sequence of rounds, where in each round all processes send messages to all, and a correct process updates its value based on the values received. It is not difficult to design an algorithm based on this all-to-all communication pattern that does not violate the validity and agreement properties of consensus, even with Byzantine faults. However, termination requires that in some round r all correct processes receive exactly the same set of messages (from correct *and* from faulty processes). Let us denote this property for round r by $\mathit{uniform}(r)$. Indeed, if $\mathit{uniform}(r)$ holds and each correct process applies a deterministic function to the received values, the configuration becomes univalent. Can we ensure the existence of a round r in which $\mathit{uniform}(r)$ holds?

For benign faults, it is easy to guarantee that during the synchronous period of the partially synchronous system, in every round r, all correct processes receive messages from the same set of processes. This is not the case for Byzantine faults. In round r, a Byzantine process could send a message to some correct process, and no message to some other correct process. If this happens, $\mathit{uniform}(r)$ does not hold. Therefore one may think that with Byzantine faults the leader is needed to ensure termination, and conclude that no deterministic leader-free Byzantine consensus algorithm could exist in a partially synchronous system. In this paper, we show that this intuition is wrong.

Our new idea is the following. We started from the observation that leader-free consensus algorithms exist for the synchronous system, both for benign faults (e.g., the *FloodSet* algorithm [13]) and for Byzantine faults (e.g., the algorithm based on interactive consistency [1]). However, these algorithms violate agreement if executed during the asynchronous period of a partially synchronous system. Therefore we tried to combine these algorithms with a second algorithm that never violates agreement in an asynchronous system. This methodology turned out to be successful, and the resulting leader-free Byzantine consensus algorithm, is presented here. The algorithm requires $3t + 1$ processes and does not rely on digital signatures.

The following approaches differ from ours. Several papers have considered some primitives to transform authenticated algorithms into non-authenticated algorithms, e.g., [5, 14, 6]. These papers either consider synchronous systems, or asynchronous systems with randomized algorithms. Note that although the "broadcast" primitive in [6] is leader-free, the consensus algorithm itself is based on a rotating coordinator, hence, it is not leader-free. Some existing papers have proposed different approaches to transform protocols resilient to crash faults into protocols resilient to Byzantine faults, e.g., [15–17]. Although the first two papers propose leader-free algorithms, the first one considers synchronous systems and the second one considers approximate agreement problem in asynchronous systems. The last paper considers asynchronous systems with failure detectors and the algorithm is leader-based.

The rest of the paper is structured as follows. We define the consensus problem and our system model in Section 2. Our methodology to derive a leader-free consensus algorithm for Byzantine faults is presented in Section 3.[1] Future work is discussed in Section 5. Finally, we conclude the paper in Section 6.

2 Definitions and System Model

2.1 Byzantine Consensus

We consider a set Π of n processes, among which at most t can be Byzantine faulty. Nonfaulty processes are called correct processes. Each process has an initial value. We formally define consensus by the following properties:

- *Strong validity:* If all correct processes have the same initial value, this is the only possible decision value.
- *Agreement:* No two correct processes decide differently.
- *Termination:* All correct processes eventually decide.

With Byzantine faults, a weaker validity property is sometimes defined. We consider here the strong validity property.

2.2 System Model

We consider a partially synchronous system as defined in [6] in which processes communicate through message passing. As in [6], we consider an abstraction on top of the system model, namely a round model, defined next. Using this abstraction, rather than the raw system model, improves the clarity of the algorithms and simplifies the proofs.

[1] A simpler algorithm that uses digital signatures is proposed in Section 4.

There are two fault models considered with Byzantine processes: "authenticated Byzantine" faults, and "Byzantine" faults [6]. In both models a faulty process behaves arbitrarily, but in the authenticated Byzantine model messages can be signed by the sender, and it is assumed that the signature cannot be forged by any other process. No signatures are used with Byzantine faults, but the receiver of a message knows the identity of the sender.

2.3 Basic Round Model

In the round model, processing is divided into rounds of message exchange. Each round r consists of a *sending step* denoted by S_p^r (sending step of p for round r), and of a *state transition step* denoted by T_p^r. In a sending step, each process sends a message to all. A subset of the messages sent is received at the beginning of the state transition step: messages can get lost, and a message sent in round r can only be received in round r. We denote by σ_p^r the message sent by p in round r, and by $\boldsymbol{\mu}_p^r$ the messages received by process p in round r ($\boldsymbol{\mu}_p^r$ is a vector of size n, where $\boldsymbol{\mu}_p^r[q]$ is the message received from q). Based on $\boldsymbol{\mu}_p^r$, process p updates its state in the state transition step.

Let *GSR* (*Global Stabilization Round*) be the smallest round, such that for all rounds $r \geq \mathit{GSR}$, the message sent in round r by a correct process q to a correct process p is received by p in round r. This is formally expressed by the following predicate (where $\mathcal{C}$ denotes the set of correct processes):

$$\forall r \geq \mathit{GSR} : \mathcal{P}_{good}(r), \text{ where } \mathcal{P}_{good}(r) \equiv \forall p, q \in \mathcal{C} : \ \boldsymbol{\mu}_p^r[q] = \sigma_q^r.$$

An algorithm that ensures — in a partially synchronous system — the existence of *GSR* such that $\forall r \geq \mathit{GSR} : \mathcal{P}_{good}(r)$, is given in [6]. Note that "$\forall r \geq \mathit{GSR} : \mathcal{P}_{good}(r)$" is sufficient for the termination of our algorithms, but not necessary. If the system is synchronous, the following stronger property can be ensured: $\forall r : \mathcal{P}_{good}(r)$.

3 Byzantine Faults: From Synchrony to Partial Synchrony

In this section we explain our methodology to design a leader-free consensus algorithm that tolerates Byzantine faults without signatures. We start with a leader-free consensus algorithm for Byzantine faults in a synchronous system model, and then extend it to a leader-free consensus algorithm in a partially synchronous system.

3.1 Leader-free Consensus Algorithm for a Synchronous System

One of the first consensus algorithms that tolerates Byzantine faults in synchronous systems was proposed by Pease, Shostak and Lamport [1]. It is based on an algorithm that solves the *interactive consistency* problem, which consists for each correct process p to compute a vector of values, with an element for each of the n processes, such that

Algorithm 1 *EIGByz* with $n > 3t$ (code of process p)

```
1:  Initialization:
2:    $W_p := \{\langle \lambda, v_p \rangle\}$                /* $v_p$ is the initial value of process $p$; $val_p(\lambda) = v_p$ */

3:  Round $r$ :                                               /* $1 \le r \le t+1$ */
4:    $S_p^r$:
5:      send $\{\langle \alpha, v \rangle \in W_p \ : \ |\alpha| = r-1 \wedge p \notin \alpha \wedge v \neq \bot\}$ to all processes
6:    $T_p^r$:
7:      for all $\{q \mid \langle \alpha, v \rangle \in W_p \wedge |\alpha| = r-1 \wedge q \in \Pi \wedge q \notin \alpha\}$ do
8:        if $\langle \beta, v \rangle$ is received from process $q$ then
9:          $W_p := W_p \cup \{\langle \beta q, v \rangle\}$                     /* $val_p(\beta q) = v$ */
10:       else
11:         $W_p := W_p \cup \{\langle \beta q, \bot \rangle\}$                    /* $val_p(\beta q) = \bot$ */
12:     if $r = t+1$ then
13:       for all $\langle \alpha, v \rangle \in W_p$ from $|\alpha| = t$ to $|\alpha| = 1$ do
14:         $W_p := W_p \setminus \langle \alpha, v \rangle$                       /* replace $val_p(\alpha)$ ... */
15:         if $\exists v'$ s.t. $|\langle \alpha q, v' \rangle \in W_p| \ge n - |\alpha| - t$ then
16:           $W_p := W_p \cup \langle \alpha, v' \rangle$                     /* ... with $newval_p(\alpha)$ */
17:         else
18:           $W_p := W_p \cup \langle \alpha, \bot \rangle$                    /* ... with $newval_p(\alpha)$ */
19:       for all $q \in \Pi$ do                                       /* level 1 of the tree */
20:         $\boldsymbol{M}_p[q] := v$ s.t. $\langle q, v \rangle \in W_p$
```

- The correct processes compute exactly the same vector;
- The element of the vector corresponding to a given correct process is the initial value of that process.

The algorithm presented in [1] is not leader-based, does not require signatures, tolerates $t < n/3$ Byzantine faults, and consists of $t+1$ rounds of exchange of messages. We briefly recall the principle of this algorithm (see Algorithm 1).

The information maintained by each process during the algorithm can be represented as a tree (called *Exponential Information Gathering (EIG)* tree in [13, 18]), in which each path from the root to a leaf contains $t+2$ nodes. Thus the height of the tree is $t+1$. The nodes are labeled with sequences of processes' identities in the following manner. The root is labeled with the empty sequence λ ($|\lambda| = 0$). Let i be an internal node in the tree with label $\alpha = p_1 p_2 \dots p_r$; for every $q \in \Pi$ such that $q \notin \alpha$, node i has one child labeled αq. Node i with label α will be simply called "node α".

Intuitively, $val_p(p_1 p_2 \dots p_r)$ (which denotes the value of node $p_1 p_2 \dots p_r$ in p's tree) represents the value v that p_r told p at round r that p_{r-1} told p_r at round $r-1$ that ... that p_1 told p_2 at round 1 that p_1's initial value is v. Each correct process p maintains the tree using a set W_p of pairs $\langle$*node label*, *node value*$\rangle$. At the beginning of round r, each process p sends the $(r-1)$th level of its tree to all processes (line 5). When p receives a message from q in format $\langle p_1 p_2 ... p_r, v \rangle$, it adds $\langle p_1 p_2 ... p_r q, v \rangle$ to its set W_p (line 9). If p fails to receive a message it expects from process q, p simply adds $\langle p_1 p_2 \dots p_r q, \bot \rangle$ to its set W_p (line 11).

Information gathering as described above continues for $t+1$ rounds, until the entire tree has been filled in. At this point the second stage of local computation starts. Every process p applies to each subtree a recursive data reduction function to compute a new value (lines 13 to 18). The value of the reduction function on p's subtree rooted at a node labeled α is denoted $newval_p(\alpha)$. The reduction function is defined for a node α as follows.

- If α is a leaf, its value does not change ($newval(\alpha) = val(\alpha)$);
- Otherwise, if there exists v such that $n - |\alpha| - t$ children have value v, then $newval(\alpha) = v$, else $newval(\alpha) = \bot$ (lines 16 and 18).

The reason for a quorum of size $n - |\alpha| - t$ can be explained as follows.[2] Each correct process, at the end of round $t + 1$, has constructed a tree with $t + 2$ levels. Any node in level $0 < k < t+1$ has $n-k$ children and a label α such that $|\alpha| = k$. If α is a label with only correct processes, then all its children except t (i.e., $n - k - t$ children) have the same value.

At the end of round $t + 1$, every correct process p constructs a vector $\boldsymbol{M}_p$ of size n (corresponding to the level 1 of its tree), where $\boldsymbol{M}_p[q]$ is the new value of process q (line 20). *EIGByz* ensures that:

- The correct processes compute exactly the same vector, i.e., $\forall p, q \in \mathcal{C} : \boldsymbol{M}_p = \boldsymbol{M}_q$, and
- The element of the vector corresponding to a given correct process q is the initial value of that process, i.e., $\forall p, q \in \mathcal{C} : \boldsymbol{M}_p[q] = v_q$.

Therefore, a correct process can decide by applying a deterministic function on its vector $\boldsymbol{M}_p$. The *EIGByz* algorithm ensures the following property:

$$(\forall r,\, 1 \leq r \leq\, t{+}1 : \mathcal{P}_{good}(r)) \Rightarrow \forall p, q \in \mathcal{C} : (\boldsymbol{M}_p = \boldsymbol{M}_q) \wedge (|\boldsymbol{M}_p| \geq |\mathcal{C}|) \qquad (1)$$

where $|\boldsymbol{M}_p|$ denotes the number of non-$\bot$ elements in vector $\boldsymbol{M}_p$, and $|\mathcal{C}|$ denotes number of correct processes. The premise holds if the system is synchronous.

3.2 Extending *EIGByz* for a Partially Synchronous Model

If Algorithm 1 is executed in a partially synchronous system, it does not ensure $\forall p, q \in \mathcal{C} : (\boldsymbol{M}_p = \boldsymbol{M}_q) \wedge (|\boldsymbol{M}_p| \geq |\mathcal{C}|)$. Therefore, it cannot ensure the agreement property of Byzantine consensus. However, following two properties hold for Algorithm 1 in synchronous as well as in asynchronous periods:

$$\forall p, q \in \mathcal{C} : \boldsymbol{M}_p[q] \in \{v_q, \bot\} \qquad (2)$$

$$\forall q \in \Pi \setminus \mathcal{C}, \exists v \text{ s.t. } \forall p \in \mathcal{C} : \boldsymbol{M}_p[q] \in \{v, \bot\} \qquad (3)$$

where v_q is the initial value of process q. The proofs are in Appendix A.

To ensure agreement in a partially synchronous system, we need to combine Algorithm 1 with another algorithm. We show below two such algorithms: (i) a simple algorithm (Algorithm 2), which requires $n > 5t$, and (ii) a more complex algorithm with optimal resilience $n > 3t$ (Algorithm 3). In both cases, Algorithm 2 and Algorithm 3 ensure agreement, while Algorithm 1 ensures termination.

Consensus Algorithm with $n > 5t$. We start with a simple parameterized consensus algorithm (see Algorithm 2). Parametrization allows us to easily adjust the algorithm to ensure agreement for different fault models. The algorithm was first presented in [19] as *One Third Rule* algorithm ($T = E = 2n/3$) to tolerate

[2] Since $n > 3t$, this quorum can be replaced by $\frac{n+t}{2} - |\alpha|$ (see [1]).

$t < n/3$ benign faults. The parameterized version was given in [20] to tolerate "corrupted communication". Here, since we consider "Byzantine process faults" we need different values for the parameters. Note that in the context of Byzantine faults, Algorithm 2 alone does not ensure termination.

Algorithm 2 Byzantine algorithm with $n > 5t$ (code of process p)

```
 1: Initialization:
 2:    $x_p := v_p \in V$                         /* $v_p$ is the initial value of $p$ */
 3: Round $r = 2\phi - 1$:           /* round simulated by $t+1$ micro-rounds of Algorithm 1 */
 4:    $S_p^r$:
 5:       send $\langle x_p \rangle$ to all processes
 6:    $T_p^r$:
 7:       if number of non-$\perp$ elements in $\boldsymbol{\mu}_p^r > T$ then
 8:          $x_p$ := smallest most frequent non-$\perp$ element in $\boldsymbol{\mu}_p^r$
 9: Round $r = 2\phi$:
10:    $S_p^r$:
11:       send $\langle x_p \rangle$ to all processes
12:    $T_p^r$:
13:       if more than $E$ elements in $\boldsymbol{\mu}_p^r$ are equal to $v \neq \perp$ then
14:          DECIDE($v$)
```

The algorithm consists of a sequence of phases ϕ, where each phase has two rounds $2\phi - 1$ and 2ϕ. Round 2ϕ is a normal round; to ensure termination, round $2\phi - 1$ will have to be simulated by Algorithm 1. Each process p has a single variable x_p, and in every round p sends x_p to all processes. Parameter T (line 7) refers to a "threshold" for updating x_p, and parameter E (line 13) refers to "enough" same values to decide.[3]

With Byzantine faults, Algorithm 2 ensures agreement with $E \geq (n+t)/2$ and $T \geq 2n - 2E + 2t$. Strong validity requires $T \geq 2t$ and $E \geq t$. Termination, together with Algorithm 1, requires $n - t > T$ and $n - t > E$. Putting all together, for the case $E = T$, we get $T = E = 2(n+t)/3$ and $n > 5t$. The proofs of agreement and strong validity are in Appendix B. We discuss now termination. For termination, it is sufficient for Algorithm 2 to have one round $r = 2\phi - 1$ in which the following holds (where $|\boldsymbol{\mu}_p^r|$ denotes the number of non-$\perp$ elements in vector $\boldsymbol{\mu}_p^r$):

$$\forall p, q \in \mathcal{C} : (\boldsymbol{\mu}_p^r = \boldsymbol{\mu}_q^r) \wedge (|\boldsymbol{\mu}_p^r| > T) \tag{4}$$

and one round $r + 1 = 2\phi$ in which we have:

$$\forall p \in \mathcal{C} : |\boldsymbol{\mu}_p^{r+1}| > E. \tag{5}$$

If (4) holds, all correct processes set x_p to the some common value v_0 in round r (line 8), and if (5) holds all correct processes decide v_0 in round $r + 1$ (line 14).

By comparing (1) with (4) and (5), it is easy to see that Algorithm 1 ensures (4) and (5) if it is executed after *GSR*, and we have $|\mathcal{C}| > T$ and $|\mathcal{C}| > E$ (where $|\mathcal{C}| = n - t$). Therefore, the idea is to replace the send/receive of round

[3] The notation $\boldsymbol{\mu}_p^r$ is introduced in Section 2.3.

$2\phi - 1$ of Algorithm 2 by the $t+1$ micro-rounds of Algorithm 1. In other words, we simulate round $r = 2\phi - 1$ of Algorithm 2 using the $t+1$ micro-rounds of Algorithm 1:

- Each instance of Algorithm 1 is started with $W_p = \{\langle p, x_p\rangle\}$, where x_p is defined in Algorithm 2;
- At the end of these $t+1$ micro-rounds, the vector $\boldsymbol{M}_p$ computed by Algorithm 1 is the vector $\boldsymbol{\mu}_p$ of messages received by p in round r ($\boldsymbol{M}_p[q] = \boldsymbol{\mu}_p[q] = \perp$ means that p did not receive any message from q in round r).

Note that, the *One Third Rule* algorithm (Algorithm 2 with $T = E = 2n/3$) cannot be used with Byzantine faults because of the agreement problem. Using *EIGByz*, a Byzantine process cannot send different values to different processes in a single round, however, it can send different values to different processes in different rounds which violates agreement.

Consensus Algorithm with $n > 3t$. As Algorithm 2 requires $n > 5t$, its resilience is not optimal. Here we show a new algorithm, which uses mechanisms from several consensus algorithms, e.g., Ben-Or [3], and PBFT [7] with strong validity, and requires only $n > 3t$ (see Algorithm 3). Note that, as for Algorithm 2, Algorithm 3 ensures strong validity and agreement, but not termination. As for Algorithm 2, termination is ensured by simulating the first round of each phase ϕ of Algorithm 3 by $t+1$ micro-round of Algorithm 1.

Algorithm 3 consists of a sequence of phases ϕ, where each phase has three rounds ($3\phi - 2$, $3\phi - 1$, 3ϕ). Each process p has an estimate x_p, a vote value $vote_p$ (initially ?), a timestamp ts_p attached to $vote_p$ (initially 0), and a set $pre\text{-}vote_p$ of valid pairs $\langle vote, ts\rangle$ (initially $\emptyset$). The structure of the algorithm is as follows:

- If a correct process p receives the same estimate v in round $3\phi - 2$ from $n - t$ processes, then it accepts v as a valid vote and puts $\langle v, \phi\rangle$ in $pre\text{-}vote_p$ set. The pre-vote set is used later to detect an invalid vote.
- If a correct process p receives the same pre-vote $\langle v, \phi\rangle$ in round $3\phi - 1$ from $n - t$ processes, then it votes v (i.e., $vote_p = v$) and updates its timestamp to ϕ (i.e., $ts_p = \phi$).
- If a correct process p receives the same vote v with the same timestamp ϕ in round 3ϕ from $2t + 1$ processes, it decides v.

The algorithm guarantees that (i) two correct processes do not vote for different values in the same phase ϕ; and (ii) once $t+1$ correct processes have the same vote v and the same timestamp ϕ, no other value can be voted in the following phases. We discuss now agreement and termination. The full proofs are in Appendix C.

Agreement: A configuration is v-valent if (i) $\exists\phi$ such that at least $t+1$ correct processes p have $(vote_p, ts_p) = (v, ts)$ with $ts \geq \phi$, and (ii) the other correct processes q have $(vote_q, ts_q) = (v' \neq v, ts')$ with $ts' < \phi$.

Let ϕ_0 be the smallest round in which some correct process decides v (line 26). By line 25 at least $t+1$ correct processes p have $vote_p = v$, $ts_p = \phi_0$, and $x_p = v$ from line 20; the other correct processes q with $vote_q \neq v$ have $ts_q < \phi_0$ from line 19. Therefore the v-valent definition holds. We denote the former set by

Algorithm 3 Byzantine algorithm with $n > 3t$ (code of process p)

```
1:  Initialization:
2:     $x_p := v_p \in V$                                   /* $v_p$ is the initial value of $p$ */
3:     $pre\text{-}vote_p := \emptyset$
4:     $vote_p \in V \cup \{?\}$, initially ?
5:     $ts_p := 0$

6:  Round $r = 3\phi - 2$ :            /* round simulated by $t + 1$ micro-rounds of Algorithm 1 */
7:     $S_p^r$:
8:        send $\langle x_p, vote_p \rangle$ to all processes
9:     $T_p^r$:
10:       if at least $n - t$ elements in $\boldsymbol{\mu}_p^r$ are equal to $\langle -, ? \rangle$ then
11:          $x_p :=$ smallest most frequent element $\langle x, - \rangle$ in $\boldsymbol{\mu}_p^r$
12:          $pre\text{-}vote_p := pre\text{-}vote_p \cup \{\langle x_p, \phi \rangle\}$
13:       if at least $n - t$ elements in $\boldsymbol{\mu}_p^r$ are equal to $\langle v, - \rangle$ then
14:          $pre\text{-}vote_p := pre\text{-}vote_p \cup \{\langle v, \phi \rangle\}$

15: Round $r = 3\phi - 1$ :
16:    $S_p^r$:
17:       send $\langle v \mid \langle v, \phi \rangle \in pre\text{-}vote_p \rangle$ to all processes
18:    $T_p^r$:
19:       if at least $n - t$ elements in $\boldsymbol{\mu}_p^r$ are equal to $\langle v \rangle$ then
20:          $vote_p := v$; $ts_p := \phi$; $x_p := v$

21: Round $r = 3\phi$ :
22:    $S_p^r$:
23:       send $\langle vote_p, ts_p, pre\text{-}vote_p \rangle$ to all processes
24:    $T_p^r$:
25:       if at least $2t + 1$ elements in $\boldsymbol{\mu}_p^r$ are equal to $\langle v \neq ?, \phi, - \rangle$ then
26:          DECIDE($v$)
27:       if exists $\langle v \neq ?, ts, - \rangle$ in $\boldsymbol{\mu}_p^r$ s.t. $vote_p \neq v$ and $ts > ts_p$ then
28:          if exists $t + 1$ elements $\langle -, -, pre\text{-}vote \rangle$ in $\boldsymbol{\mu}_p^r$ s.t. $\langle v, ts' \rangle \in pre\text{-}vote$ and $ts' \geq ts$ then
29:             $vote_p := ?$; $ts_p := 0$; $x_p := v$
30:       if $vote_p \neq ?$ then $x_p := vote_p$
```

$\Pi_{=\phi_0}$, and the latter by $\Pi_{<\phi_0}$. Processes in $\Pi_{=\phi_0}$ keep $x_p = vote_p = v$ from phase ϕ_0 onward, and processes in $\Pi_{<\phi_0}$ can only update $vote_p$ to ? or v, as we explain now. This ensures agreement.

First, by lines 10 and 13, it is impossible for a correct process to have two different values with the same timestamp in its pre-vote set. By lines 27-30, in phase ϕ_0, processes in $\Pi_{<\phi_0}$ can only update $vote_p$ to ?; processes in $\Pi_{=\phi_0}$ do not update neither $vote_p$, nor x_p to some value $\neq v$. By lines 10-14, in phase $\phi_0 + 1$, correct processes can only update x_p to v and can only add $(v, \phi_0 + 1)$ to $pre\text{-}vote_p$. Therefore in round $3(\phi_0 + 1) - 1$, correct processes can only update $vote_p$ to v, i.e., only v can be decided in phase $\phi_0 + 1$. The same reasoning can be repeated for all phases after phase $\phi_0 + 1$.

Termination: We explain intuitively termination by considering the smallest phase ϕ such that $3\phi - 2 \geq GSR$. We distinguish two cases: (i) at the beginning of round $3\phi - 2$, all correct processes have $vote_p = ?$, and (ii) at the beginning of round $3\phi - 2$ at least one correct process has $vote_p \neq ?$.

Case (i): Consider round $3\phi - 2$. Since we are after GSR, Algorithm 1 ensures that all correct processes p receive the same set $\boldsymbol{\mu}_p^{3\phi-2}$ of messages with $|\boldsymbol{\mu}_p^{3\phi-2}| \geq |\mathcal{C}|$ (see formula (1)), i.e., all correct processes p set x_p to the same common value v (line 11), and add the pair $\langle v, \phi \rangle$ to $pre\text{-}vote_p$ (line 12). It follows that, in round

$3\phi - 1$, all correct processes p set $vote_p$ to v (line 20), and all correct processes decide v in round 3ϕ (line 26).

Case (ii): This case is more complex to expose. Consider round 3ϕ, and let q be a correct process with the highest timestamp ts_q and $vote_q = v \neq ?$ at the beginning of round 3ϕ. Line 19 ensures that for any other correct process q' with $ts_{q'} = ts_q$, we have $vote_q = vote_{q'}$. Since $3\phi > GSR$, all correct processes p with $vote_p \neq v$ execute lines 27-29. Therefore, at the end of round 3ϕ all correct processes p have $x_p = v$ and $vote_p \in \{v, ?\}$, i.e., all correct processes p start round $3\phi + 1 = 3(\phi + 1) - 2$ with $x_p = v$. If the condition of line 10 holds, then the most frequent pair received is $\langle v, - \rangle$, i.e., $\langle v, \phi + 1\rangle$ is added to $pre\text{-}vote_p$ (line 12). The condition of line 13 necessary holds at each correct process, i.e., $\langle v, \phi + 1\rangle$ is added to $pre\text{-}vote_p$ (line 14). Therefore, at the end of round $3\phi + 1$, all correct processes p only have $\langle y, \phi + 1\rangle$ with $y = v$ in $pre\text{-}vote_p$. It follows that, in round $3\phi + 2$, all correct processes p set $vote_p$ to v (line 20), and all correct processes decide v in round $3\phi + 3$ (line 26).

Note that in Algorithm 3, the set $pre\text{-}vote_p$ can be bounded, based on the following observation. For instance, if $\langle v, \phi\rangle \in pre\text{-}vote_p$ and p wants to add $\langle v, \phi'\rangle$ into its pre-vote with $\phi' > \phi$, then $\langle v, \phi\rangle$ becomes obsolete.

3.3 Summary

The following table summarizes our results. The second column shows the smallest number of processes needed for each algorithm. The third and forth columns give an upper bound on number of rounds needed for a single consensus in both best and worst cases. The best case is when the system is synchronous form the beginning, i.e., $GSR = 0$. Both algorithms require n^2 messages per round.

	# processes	# rounds (best case)	# rounds (worst case)
Algorithm 2	$5t + 1$	$t + 2$	$GSR + 2(t + 2) - 1$
Algorithm 3	$3t + 1$	$t + 3$	$GSR + 2(t + 3) - 1$

3.4 Optimizations

We describe two possible optimizations that can be applied to our leader-free Byzantine consensus algorithm.

Early termination: The "early termination" optimization can be applied to Algorithm 1 (*EIGByz*). Algorithm 1 always requires $t + 1$ rounds, even in executions in which no process is faulty. With early termination, the number of rounds can be reduced in such cases.

Let f denote the actual number of faulty processes in a given execution. Moses and Waarts in [21] present an early termination version of the exponential information gathering protocol for Byzantine consensus that requires $n > 4t$ and terminates in $min\{t+1, f+2\}$ rounds. The idea is the following. Consider some node α in p's tree. Process p may know that a quorum (i.e., $n - |\alpha| - t$) of correct children of node α store the same value. When this happens, process p can already determine the value of $newval_p(\alpha)$, and can stop at the end of the next round. The paper presents another early termination protocol with optimal resiliency ($n > 3t$) that terminates in $min\{t+1, f+3\}$ rounds. These two optimizations can be applied to Algorithm 1.

One round decision: The "one round decision" optimization is relevant to Algorithm 2. One round decision means that if all correct processes start with the same initial value, and the system is synchronous from the beginning, then correct processes decide in one single round. Algorithm 2 does not achieve one round decision, because the simulation of Algorithm 1 (*EIGByz*) appears in each phase, including phase 1. To achieve one round decision, we simply skip round 1, and start Algorithm 2 with round 2. If all correct processes start with the same initial value, and $GSR = 0$, then correct processes decide in one round.

The fact that our one round decision algorithm requires "only" $n > 5t$ is not in contradiction with the result in [22], which establishes the lower bound $n = 7t + 1$ for one-step decision. The reason is that we assume for fast decision a partially synchronous system with $GSR = 0$, i.e., the system is initially synchronous, while [22] considers a system that is initially asynchronous.

4 Authenticated Byzantine Faults

In this section we show that leader-free Byzantine consensus is even simpler if signatures are used (a fault model called authenticated Byzantine faults, see Section 2.2). In this model, a faulty process who cheats about its value can be detected by the correct processes. Therefore, the *EIGByz* algorithm is not needed here. It can be replaced by a leader-free synchronous algorithm that uses digital signatures (to simulate rounds of Algorithm 2).

We consider here a variant of the *FloodSet* algorithm [13] (see also [23]) called *Authenticated FloodSet*, see Algorithm 4. With similar arguments as in Section 3, the combination of Algorithm 2 (or Algorithm 3) and Algorithm 4 ensures strong validity and agreement. Termination holds from Algorithm 4 in a partially synchronous system (after *GSR*).

In Algorithm 4 we denote by $v : p$ the value v signed by process p, and by $v : p_1 : p_2 : ... : p_k$ the value v signed by k processes, initially by p_1, then $v : p_1$ signed by p_2, etc. In round r processes send values signed exactly by r distinct processes, and accept only values signed exactly by r distinct processes.

At line 5 of round r, process p sends $W_p : p$, which denotes the set obtained by having p signing all elements in set W_p not yet signed by itself. In round r, a process keeps only values received that are signed by r different processes (line 9 and 10). In round $t+1$, a correct process eliminates inconsistent values, i.e., two different initial values signed by the same process (line 13 and 14). At the end, every correct process constructs a vector $\boldsymbol{M}_p$ of size n, where $\boldsymbol{M}_p[q]$ is the initial value of process q (or $\perp$ if q is faulty).

Algorithm 4 *Authenticated FloodSet* with $n > t$ (code of process p)

```
Initialization:
  $W_p := \{v_p\}$                                   /* $v_p$ is the initial value of process $p$ */
Round $r$ :                                                                /* $1 \leq r \leq t+1$ */
  $S_p^r$:
    send $\langle W_p : p \rangle$ to all processes
  $T_p^r$:
    for all $q$ from which the set $W_q$ is received do
      for all $e \in W_q$ do
        if $e$ is signed by $r$ different processes then
          $W_p := W_p \cup \{e\}$
    if $r = t+1$ then
      for all $q \in \Pi$ do
        if $(v : q : ... \in W_p)$ and $(v' : q : ... \in W_p)$ and $v \neq v'$ then
          remove all elements $(- : q : ...)$ from $W_p$     /* eliminate inconsistent values of $q$ */
          if $\exists v$ such that $(v : q : ...) \in W_p$ then $M_p[q] := v$
          else $M_p[q] := \perp$
```

5 Discussion and Future Work

In a partially synchronous system the predicate $\mathcal{P}_{good}(r)$ can be ensured using the implementations given in [6]. Actually, [6] distinguishes two variant of partial synchrony: (a) one in which the communication bound Δ and the process bound Φ are *unknown*, and (b) one in which these bounds are known but hold only eventually.The implementation of the round model slightly differs depending on the partial synchrony variant that is considered. We consider here model (a), which is also the model considered in the leader-based Castro-Liskov PBFT protocol [7]. In this model a standard technique, used for example in PBFT, is to have exponentially growing timeouts. For example, in PBFT whenever the leader changes (i.e., the recovery protocol has to be executed), the timeout for the next leader is doubled. Taking this leader-based protocol as a case study, Amir et al. [10] pointed its vulnerability to performance degradation under an attack. Indeed in PBFT, f consecutive Byzantine leaders, say l_1, l_2, ..., l_f could do construct the following attack. The first leader l_1 is mute, the timeout expires, the recovery protocol is activated, and the algorithm switches to the next leader (rotating coordinator) while doubling the timeout. The same happens for leaders l_2 to l_{f-1} until l_f becomes leader. The last leader l_f sends its message as late as possible, but not too late to remain leader. If l_f remains leader forever, then the time required for any request (instance of consensus) is high.

Although PBFT does not assume a round-based model as we do in this paper, the performance failure attack is possible in the case of a leader-based protocol implemented in the round-based model, in the case the round-based model is constructed on top of a partially synchronous model of type (a). However, we believe that this is not the case for leader-free algorithms, i.e., performance failure attacks are not effective in this case. The intuition is that, once the timeout of a correct process becomes large enough to receive all messages from correct processes, Byzantine processes cannot introduce an attack that forces the correct process to double its timeout. Our future work is to validate this intuition analytically and/or experimentally, and to understand under which conditions leader-free algorithms outperform leader-based algorithms.

6 Conclusion

All previously known deterministic consensus algorithms for partially synchronous systems and Byzantine faults are leader-based. However, leader-based algorithms are vulnerable to performance degradation, which occurs when the Byzantine leader sends messages slowly, but without triggering timeouts. In the paper we have proposed a deterministic (no randomization), leader-free Byzantine consensus algorithm in a partially synchronous system. Our algorithm is resilient-optimal (it requires $3t+1$ processes) and signature-free (it doesn't rely on digital signatures). To the best of our knowledge this is the first Byzantine algorithm that satisfies all these characteristics. We have also presented optimizations for the Byzantine consensus algorithm, including one-round decision. Finally, a simpler leader-free consensus algorithm that uses digital signatures is proposed.

We have designed our algorithms using a new methodology. It consists of extending a synchronous consensus algorithm to a partially synchronous consensus algorithm using an asynchronous algorithm.The asynchronous protocol ensures safety (i.e., agreement and strong validity), while the synchronous algorithm provides liveness (i.e., termination) during periods of synchrony.

Acknowledgments: We would like to thank Martin Hutle, Nuno Santos and Olivier Rütti for their comments on an earlier version of the paper.

References

1. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2) (1980) 228–234
2. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. JACM **32**(2) (apr 1985) 374–382
3. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: PODC'83, NY, USA, ACM (1983) 27–30
4. Rabin, M.: Randomized Byzantine generals. In: Proc. Symposium on Foundations of Computer Science. (1983) 403–409
5. Bracha, G.: An asynchronous [(n - 1)/3]-resilient consensus protocol. In: PODC'84, New York, NY, USA, ACM (1984) 154–162
6. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. JACM **35**(2) (apr 1988) 288–323
7. Castro, M., Liskov, B.: Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS) **20**(4) (November 2002) 398–461
8. Martin, J.P., Alvisi, L.: Fast Byzantine consensus. IEEE Transactions on Dependable and Secure Computing **3**(3) (jul 2006) 202–215
9. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative byzantine fault tolerance. SIGOPS Oper. Syst. Rev. **41**(6) (2007) 45–58
10. Amir, Y., Coan, B., Kirsch, J., Lane, J.: Byzantine replication under attack. In: DSN'08. (2008) 197–206

11. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: NSDI'09, Berkeley, CA, USA, USENIX Association (2009) 153–168
12. Lamport, L.: State-Machine Reconfiguration: Past, Present, and the Cloudy Future. DISC Workshop on Theoretical Aspects of Dynamic Distributed Systems (September 2009)
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
14. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing **2**(2) (1987) 80–94
15. Neiger, G., Toueg, S.: Automatically increasing the fault-tolerance of distributed algorithms. J. Algorithms **11**(3) (1990) 374–419
16. Coan, B.A.: A compiler that increases the fault tolerance of asynchronous protocols. IEEE Trans. Comput. **37**(12) (1988) 1541–1553
17. Baldoni, R., Hélary, J.M., Raynal, M.: From crash fault-tolerance to arbitrary-fault tolerance: Towards a modular approach. In: DSN'00, Washington, DC, USA, IEEE Computer Society (2000) 273–282
18. Attiya, H., Welch, J.: Distributed Computing: fundamentals, simulations, and advanced topics. John Wiley & Sons (2004)
19. Charron-Bost, B., Schiper, A.: The Heard-Of model: computing in distributed systems with benign faults. Distributed Computing **22**(1) (2009) 49–71
20. Biely, M., Widder, J., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A.: Tolerating corrupted communication. In: PODC'07, NY, USA, ACM (2007) 244–253
21. Moses, Y., Waarts, O.: Coordinated traversal: (t + 1)-round byzantine agreement in polynomial time. In: FOCS. (1988) 246–255
22. Song, Y.J., van Renesse, R.: Bosco: One-step Byzantine asynchronous consensus. In: DISC. (2008) 438–450
23. Dolev, D., Strong, H.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4) (1983) 656–666

APPENDIX

A Proof of Formulas (2) and (3) – page 6

We first show two lemmas (adapted from [13] for a synchronous system) that hold in any execution of Algorithm 1, both in synchronous and asynchronous periods.

Lemma A.1. *Let q be a correct process, and p some other correct process such that $val_p(\alpha q) \neq \bot$. Then, after $t+1$ rounds, for all correct processes p' we have $val_{p'}(\alpha q) = val_p(\alpha q)$ or $val_{p'}(\alpha q) = \bot$.*

Proof. If $q \notin \{p, p'\}$, then the result follows from the fact that, since q is correct, it sends the same message to p and p' at round $|\alpha| + 1$. If the message sent by q to p' gets lost, then $val_{p'}(\alpha q) = \bot$. If $q \in \{p, p'\}$, then the result follows similarly from the convention by which each process relays values to itself. □

Lemma A.2. *Let q be a correct process, and p some other correct process such that $val_p(\alpha q) \neq \bot$. Then, after $t+1$ rounds we have $newval_p(\alpha q) = val_p(\alpha q)$ or $newval_p(\alpha q) = \bot$.*

Proof. By induction on the tree labels, working from the leaves up - that is, from those of length $t+1$ down to those of length 1.

Basis: Suppose, αq is a leaf, that is, $|\alpha q| = t+1$. Then Lemma A.1 implies that all correct processes p have the same $val_p(\alpha q)$ or $\bot$. Then also $newval_p(\alpha q) = val_p(\alpha q)$ or $newval_p(\alpha q) = \bot$ for every correct process p, by the definition of $newval$ for leaves.

Inductive step: Suppose $|\alpha q| = r$, $1 \leq r \leq t$. Then Lemma A.1 implies that for all correct processes p, have the same $val_p(\alpha q)$, call this v, or $\bot$. Therefore, every correct process q' send the same value v for αq to all processes at round $r+1$, so $val_p(\alpha q q') = v$ or $val_p(\alpha q q') = \bot$ for all correct p and q'. Then the inductive hypothesis implies that also $newval_p(\alpha q q') = v$ or $newval_p(\alpha q q') = \bot$ for all correct processes p and q'.

We now claim that $newval_p(\alpha q) = v$ or $newval_p(\alpha q) = \bot$. The number of children of αq is exactly $n - r$ which is $\geq n - t$ (i). At most t of the children have labels ending with faulty processes. Since $n > 3t$ we have $n - r - t > t$ (ii). From (i), (ii) and the definition of $newval$ we have $newval_p(\alpha q) = v$ or $newval_p(\alpha q) = \bot$. □

Based on these two lemmas we prove the following lemmas. Lemma A.3 proves (2), while Lemma A.4 proves (3).

Lemma A.3. *Let q be a correct process with initial value v_q, and p some other correct process. Then, after $t+1$ rounds, we have $\boldsymbol{M}_p[q] = v_q$ or $\boldsymbol{M}_p[q] = \bot$.*

Proof. Assume that q is a correct process with initial value v_q. We have to show that for any correct process p, $\boldsymbol{M}_p[q] \in \{v_q, \perp\}$ or $newval_p(q) \in \{v_q, \perp\}$. First note that from Lemma A.1 when $|\alpha| = 0$, and for some process p, $val_p(q) = v_q$, then for all correct process p', $val_{p'}(q) \in \{v_q, \perp\}$. Then from Lemma A.2 when $|\alpha| = 0$, for some correct process p, $val_p(q) = v_q$, then $newval_p(q) \in \{v_q, \perp\}$ or $\boldsymbol{M}_p[q] \in \{v_q, \perp\}$. □

Lemma A.4. *Let q be a faulty process. There exists v such that after $t + 1$ rounds, for all correct processes p, we have $\boldsymbol{M}_p[q] = v$ or $\boldsymbol{M}_p[q] = \perp$.*

Proof. Assume that q is a faulty process. And some correct process p has $\boldsymbol{M}_p[q] = v$ or $newval_p(q) = v \neq \perp$. We have to show that for all correct processes p', $newval_{p'}(q) \in \{v, \perp\}$. $newval_p(q) = v$ means that the node labeled q in the tree constructed by correct process p has at least $n - 1 - t$ children labeled qx with $newval_p(qx) = v$ because of the *newval* definition (i). However, since node q is a faulty process, among its children, only $t - 1$ of them have a label ending with faulty process (ii). We denote $Q = \{q' \mid newval_p(qq') = v \land q' \text{is correct}\}$. From (i) and (ii) we have $|Q| \geq n-1-t-t+1 = n-2t$. And $\forall q' \in Q : newval_p(qq') = v$. From Lemma A.2 we have $\forall q' \in Q : val_p(qq') = v$. From Lemma A.1, for any correct process p' we have $\forall q' \in Q : val_{p'}(qq') \in \{v, \perp\}$. Again from Lemma A.2 we have $\forall q' \in Q : newval_{p'}(qq') \in \{v, \perp\}$. This holds for at least $n - 2t$ of node q's children in tree constructed by p'. So at most $2t-1$ of node q's children might have $newval_{p'}(qq') = v' \notin \{v, \perp\}$. Since $n > 3t$, we have $n - 1 - t$ (the required quorum) $> 2t - 1$ which means that v' cannot be a *newval* and for all correct processes p' we have $\boldsymbol{M}_{p'}[q] \in \{v, \perp\}$. □

B Proof of Algorithm 2 – page 7

Lemma B.1. *Consider Algorithm 2 with Byzantine faults and $E \geq \frac{n+t}{2}$. If some correct process decides v in phase ϕ, then some other correct process can only decide v in phase ϕ.*

Proof. Assume that some correct process p decides v in round $r = 2\phi$. From condition at line 13, p has received more than E values v in round r, i.e., more than $\frac{n+t}{2} - t$ correct processes have sent v in round r. This means that at most $n - \frac{n+t}{2} + t = \frac{n+t}{2}$ processes could have sent a value $v' \neq v$ in round r. Since $E \geq \frac{n+t}{2}$, value v' cannot be decided in round r. □

Lemma B.2. *With Byzantine faults and $T \geq 2n - 2E + 2t$, if some correct process decides v in round $r = 2\phi$ of Algorithm 2, every correct process q that updates x_q in round $r' > r$, sets it to v.*

Proof. Assume that some correct process p decides v in round $r = 2\phi$. First we prove by induction on r that more than $E - t$ correct processes q have $x_q = v$ in round $r' \geq r$.

Base step ($r' = r$): Since p decides v in round r (line 14), from condition at line 13, p receives more than E values v in round r, i.e., more than $E - t$ correct processes q have $x_q = v$ in round r.

Induction step (from $r' = 2\phi'$ to $r'+1$): By induction hypothesis, more than $E-t$ correct processes q have $x_q = v$ in round $r' > r$. Therefore, at most $n-E+t$ processes q' might send $x_{q'} = v' \neq v$ in round $r'+1$. A correct process q updates x_q only in line 8, and if it receives messages from $k > T$ processes. From the assumption we have $T \geq 2n-2E+2t$ or $k > 2(n-E+t)$. Therefore, no correct process q updates x_q to v' in round $r'+1$. This implies that more than $E-t$ correct processes q have $x_q = v$ in round $r'+1$.

Let q' be some correct process that updates $x_{q'}$ in some round $r' = 2\phi'-1 > r$. Since more than $E-t$ correct processes q have $x_q = v$ in round r', and $T \geq 2n-2E+2t$, by same arguments as in induction step, q' sets $x_{q'}$ to v in round r'. □

Lemma B.3. *With Byzantine faults, $T \geq 2t$ and $E \geq t$, if all correct processes p have $x_p = v$ in round $r = 2\phi - 1$ of Algorithm 2, every correct process q that updates x_q in round $r' \geq r$, sets it to v.*

Proof. Assume that all correct processes have the same initial value v. Consider some correct process p so that the condition at line 7 holds. This means that p has received more than T non-$\bot$ messages. From $T \geq 2t$, p has received at least $2t+1$ non-$\bot$ messages. Among these messages at most t can have a value $v' \neq v$, and at least $t+1$ messages have v. Therefore, if p updates x_p at line 8, it sets x_p to v. □

Therefore we have following proposition.

Proposition B.1. *With Byzantine faults, $n > 5t$ and $T = E = 2(n+t)/3$, if round $2\phi-1$ of Algorithm 2 is simulated by the $t+1$ micro-rounds of Algorithm 1, then Algorithm 2 ensures strong validity, agreement and termination.*

Proof. Agreement follows directly from Lemmas B.1, B.2 and Lemmas A.3, A.4. Strong validity follows from Lemmas A.3, A.4 and B.3. For termination, if (4) holds, all correct processes set x_p to the some common value v_0 in round r (line 8), and if (5) holds all correct processes decide v_0 in round $r+1$. By comparing (1) with (4) and (5) it is easy to see that Algorithm 1 ensures (4) and (5) if $|\mathcal{C}| \geq n-t$, $n-t > T$, $n-t > E$, and Algorithm 1 is executed after *GSR*. □

C Proof of Algorithm 3 – page 9

Proof of agreement:

Lemma C.1. *Assume $n > 2t$. For all phases ϕ, and all correct processes p, there is at most one pair $\langle -, \phi \rangle$ in pre-vote$_p$.*

Proof. Consider round $3\phi - 2$. Assume that some correct process p adds $\langle v, \phi \rangle$ to *pre-vote$_p$* at line 14. By line 13, p received $n-t$ messages equal to $\langle v, - \rangle$. Assume for a contradiction that p has added $\langle v', \phi \rangle$, with $v' \neq v$, to *pre-vote$_p$* at line 12. By line 11, this is only possible if p has received $n-t$ messages $\langle v', ?\rangle$. In this case, p has received $(n-t)+(n-t)$ messages in round $3\phi-2$. However, if $n > 2t$, then $(n-t)+(n-t) > n$, a contradiction. □

We define $\mathcal{P}_{agree}(3\phi-1, v)$ as the following predicate: *$\exists ts$ such that at the end of round $3\phi-1$, (i) for at least $t+1$ correct processes p we have $x_p = vote_p = v$ and $ts_p \geq ts$, and (ii) for other correct processes q, if $\langle v', ts'\rangle \in pre\text{-}vote_q$ s.t. $v' \neq v$, then $ts' \leq ts$.*

Lemma C.2. *Assume $n > 2t$. If $\exists \phi, v$ such that $\mathcal{P}_{agree}(3\phi-1, v)$ holds, then for all $\phi' \geq \phi$, $\mathcal{P}_{agree}(3\phi'-1, v)$ also holds.*

Proof. The proof is by induction on ϕ.

Base step ($\phi' = \phi$): $\mathcal{P}_{agree}(3\phi-1, v)$ holds trivially from the assumption.

Induction step (from ϕ' to $\phi'+1$): By induction hypothesis, $\mathcal{P}_{agree}(3\phi'-1, v)$ holds. By the definition of $\mathcal{P}_{agree}(3\phi'-1, v)$, at the end of round $3\phi'-1$, (i) for at least $t+1$ correct processes p we have $x_p = vote_p = v$ and $ts_p \geq ts$, and (ii) for all other correct processes q, if $\langle v', ts'\rangle \in pre\text{-}vote_q$ s.t. $v' \neq v$, then $ts' \leq ts$. From (i) and (ii), no correct process p with $x_p = vote_p = v$ executes line 29 in round $3\phi'$.

Therefore $\mathcal{P}_{agree}(3\phi, v)$ holds, i.e., at least $t+1$ correct processes start round $3\phi'+1$ with $x_p = vote_p = v$. As a consequence, in round $3\phi'+1 = 3(\phi'+1)-2$, for correct processes, (i) the condition of line 10 cannot hold and (ii) the condition of line 13 can only hold for value v. It follows that no correct process p adds $\langle v', \phi'+1\rangle$ $(v' \neq v)$ to $pre\text{-}vote_p$, and $\mathcal{P}_{agree}(3\phi+1, v)$ holds.

In round $3\phi'+2 = 3(\phi'+1)-1$, since no correct process sends v' and $n-t > t$ (since $n > 2t$), no correct process sets $vote_p$ to $v' \neq v$. Therefore, $\mathcal{P}_{agree}(3\phi'+2, v)$ holds. □

Lemma C.3. *If some correct process p decides v in round 3ϕ, then $\mathcal{P}_{agree}(3\phi-1, v)$ holds.*

Proof. From line 25, at the end of round $3\phi-1$, at least $t+1$ correct processes q have $vote_q = v$, $ts_q = \phi$, and thus $x_q = v$ (from line 20). From lines 12 and 14, in round $3\phi-2$, no correct process adds $\langle -, x\rangle$, with $x > \phi$, to $pre\text{-}vote_p$. Therefore $\mathcal{P}_{agree}(3\phi-1, v)$ holds. □

Proposition C.1 (Agreement). *Assume $n > 2t$. If some correct process p decides v in phase ϕ, then no correct process decides $v' \neq v$ in phase $\phi' \geq \phi$.*

Proof. From Lemma C.3, if some correct process p decides v in phase ϕ, then $\mathcal{P}_{agree}(3\phi-1, v)$ holds. By Lemma C.2, $\mathcal{P}_{agree}(3\phi'-1, v)$ holds for all $\phi' \geq \phi$. This means that, for all $\phi' \geq \phi$, at the end of round $3\phi'-1$, for at least $t+1$ correct processes p we have $x_p = vote_p = v$. Therefore, at least $t+1$ correct processes p have $\langle v, -\rangle \in pre\text{-}vote_p$ in round $3\phi'-1$. From this and Lemma C.1, at most $n-t-1$ processes q may have $x_q = v'$ and $\langle v', -\rangle \in pre\text{-}vote_q$ in round $3\phi'-1$. This means that no correct process q sets $vote_q$ to v' in round $3\phi'-1$. Therefore, in round $3\phi'$ the condition of line 25 cannot hold for $v' \neq v$. □

Proof of termination:

Lemma C.4. *Assume $n > 3t$. In all rounds $r = 3\phi - 1$, if some correct process p sets $vote_p$ to $v \neq ?$, and some other correct process q sets $vote_q$ to $v' \neq ?$, then $v = v'$.*

Proof. Assume by contradiction that $v \neq v'$. By line 19, p receives $n-t$ messages v in round r and q receives $n-t$ messages v' in round r. From $n > 3t$, we have $(n-t)+(n-t) = 2n-2t > n+t$, or $(n-t)+(n-t) \geq n+t+1$. Therefore, $t+1$ processes have sent v to p and v' to q, i.e., one correct process has sent v to p and v' to q. A contradiction with Lemma C.1. □

Lemma C.5. *Assume $n > 3t$. Let ϕ be the smallest phase such that round $r = 3\phi$ is after GSR. Let q be a correct process with the highest timestamp ts_q and $vote_q = v \neq ?$ at the beginning of round r. Then at the end of round r all correct processes p have $x_p = v$ and $vote_p \in \{v, ?\}$.*

Proof. At the beginning of round r, for any correct process p three cases are possible: (i) $vote_p = v$, or (ii) $vote_p = v' \neq v$, or (iii) $vote_p = ?$.

In case (i), process p does not execute line 29 in round r, but executes line 30, and sets x_p to v.

In case (ii), from Lemma C.4, since $vote_p \neq vote_q$, $ts_p \neq ts_q$. By assumption ts_q is the highest timestamp, and so we have $ts_p < ts_q$. By line 19, at least $n-2t$ correct processes have $\langle v, ts_q \rangle$ in their pre-vote. If $n > 3t$, then $n-2t \geq t+1$. Since round r is executed after GSR, all messages sent in round r are received by all correct processes. Therefore, p executes line 29 in round r, and sets $vote_p$ to ?, x_p to v.

In case (iii), process p has $ts_p = 0 < ts_q$ and for the same reason as case (ii) executes line 29 in round r, and sets x_p to v, $vote_p$ to ?.

Therefore, at the end of round r, all correct processes p have $x_p = v$ and $vote_p \in \{v, ?\}$. □

Proposition C.2 (Termination). *Assume $n > 3t$. If round $3\phi - 2$ of Algorithm 3 is simulated by $t+1$ micro-rounds of Algorithm 1, then Algorithm 3 ensures termination.*

Proof. Consider round $r = 3\phi - 2 > GSR$ simulated by $t+1$ micro-rounds of Alogrithm 1. This implies that all correct processes receive the same set of messages in round r. Two cases are possible at the beginning of round r: (i) all correct processes p have $vote_p = ?$, or (ii) some correct process p has $vote_p \neq ?$.

In case (i), all correct processes p choose the same value v by line 11. and add $\langle v, \phi \rangle$ to $pre\text{-}vote_p$ in round r. By Lemma C.1 no other pair is added to $pre\text{-}vote_p$ in round r. In round $r+1 = 3\phi - 1$, all correct processes send v, receive at least $n-t$ messages v and set $vote_p = v$, $ts_p = \phi$. Finally in round $r+2 = 3\phi$, all correct processes send $\langle v, \phi, - \rangle$, receive at least $n-t$ messages $\langle v, \phi, - \rangle$ and decide v.

In case (ii), from Lemma C.5, all correct processes p have $x_p = v$ and $vote_p \in \{v, ?\}$ at the end of round $r+2 = 3\phi$. All correct processes start round $r+3 =$

$3\phi+1$ with $x_p = v$. In round $3\phi+1$, for any correct process p, if the condition of line 10 becomes true, x_p is updated to v because $n-2t > t$. And the condition of line 13 cannot be true for $\langle v' \neq v, - \rangle$ since $n-t > t$. Therefore, no correct process p adds $\langle v' \neq v, \phi+1 \rangle$ to *pre-vote*$_p$. By arguments similar to those of case (i), all correct processes decide v by round $r+5 = 3(\phi+1)$. □

Proof of strong validity: We define $\mathcal{P}_{val}(3\phi-1, v)$ as the following predicate: *at the end of round* $3\phi-1$, *(i) all correct processes* p *have* $x_p = v$, *vote*$_p \in \{v, ?\}$, *and (ii)* $\nexists v' \neq v$ *s.t.* $\langle v', - \rangle \in$ *pre-vote*$_p$.

Lemma C.6. *Assume* $n > 3t$. *If* $\exists \phi, v$ *such that* $\mathcal{P}_{val}(3\phi-1, v)$ *holds, then for all* $\phi' \geq \phi$, $\mathcal{P}_{val}(3\phi'-1, v)$ *also holds.*

Proof. The proof is by induction on ϕ.

Base step ($\phi' = \phi$)*:* $\mathcal{P}_{val}(3\phi-1, v)$ holds trivially from the assumption.

Induction step (from ϕ' *to* $\phi'+1$*):* By induction hypothesis, $\mathcal{P}_{val}(3\phi'-1, v)$ holds. By the definition of $\mathcal{P}_{val}(3\phi'-1, v)$, at the end of round $3\phi'-1$, all correct processes p have $x_p = v$, *vote*$_p \in \{v, ?\}$, and $\nexists v' \neq v$ s.t. $\langle v', - \rangle \in$ *pre-vote*$_p$. This means that in round $3\phi'$ no correct process executes line 29, and $\mathcal{P}_{val}(3\phi', v)$ holds. Therefore all correct processes p start round $3\phi'+1 = 3(\phi'+1)-2$ with $x_p = v$ and *vote*$_p \in \{v, ?\}$.

Assume that the condition of line 10 holds at some correct process q. In this case, q has received at least $n-2t$ messages from correct processes, and at most t messages from Byzantine processes. However, $n > 3t$ ensures $n-2t > t$, which means that q can only add v to *pre-vote*$_q$ in line 12. Assume that the condition of line 13 holds at some correct process q. From $n > 2t$, we have $n-t > t$. Since all correct processes send $\langle v, - \rangle$, the condition of line 13 can only hold for v, which means that q can only add v to *pre-vote*$_q$ in line 14. Therefore part (ii) of $\mathcal{P}_{val}(3\phi'+1, v)$ holds, and since *pre-vote* is not updated in round $3\phi'+2 = 3(\phi'+1)-1$, part (ii) of $\mathcal{P}_{val}(3\phi'+2, v)$ also holds.

Form this it follows that, in round $3\phi'+2 = 3(\phi'+1)-1$, all correct processes send only v. From $n > 2t$, we have $n-t > t$. Therefore, the condition of line 19 can only hold for v. It follows that part (i) of $\mathcal{P}_{val}(3\phi'+2, v)$ holds. □

Lemma C.7. *Assume* $n > 3t$. *If all correct processes* p *have the same initial value* v, *then* $\mathcal{P}_{val}(2, v)$ *holds.*

Proof. Since all correct processes have $x_p = v$ and *vote*$_p =?$ at the beginning of round 1, and $n-2t > t$ no correct process p adds $\langle v', 1 \rangle$ into *pre-vote*$_p$ by lines 12 and 14. In round 2, since no correct process sends v' and $n-t > t$, no correct process votes v'. Therefore, at the end of round 2 all correct processes p have $x_p = v$, *vote*$_p \in \{v, ?\}$, and $\nexists v' \neq v$ s.t. $\langle v', 1 \rangle \in$ *pre-vote*$_p$. This means that $\mathcal{P}_{val}(2, v)$ holds. □

Proposition C.3 (Strong validity). *Assume* $n > 3t$. *If all correct processes have the same initial value* v, *then no correct process decides* $v' \neq v$.

Proof. By Lemma C.7, $\mathcal{P}_{val}(2, v)$ holds. By Lemma C.6, $\mathcal{P}_{val}(3\phi - 1, v)$ holds for all $\phi \geq 1$. This means that, for all $\phi > 1$, at the end of round $3\phi - 1$, all correct processes p have $x_p = v$, and $vote_p \in \{v, ?\}$. Therefore, in round 3ϕ the condition of line 25 cannot hold for $v' \neq v$. □

Therefore we have following proposition.

Proposition C.4. *With Byzantine faults and $n > 3t$, if round $3\phi - 2$ of Algorithm 3 is simulated by the $t + 1$ micro-rounds of Algorithm 1, then Algorithm 3 ensures strong validity, agreement and termination.*

Proof. This follows from Propositions C.1, C.2, and C.3.