Reducing Round Complexity of Byzantine Broadcast

Linda Chen, Jun Wan

Abstract

Byzantine Broadcast is an important topic in distributed systems and improving its round complexity has long been a focused challenge. Under honest majority, the state of the art for Byzantine Broadcast is 10 rounds for a static adversary and 16 rounds for an adaptive adversary. In this paper, we present a Byzantine Broadcast protocol with expected 8 rounds under a static adversary and expected 10 rounds under an adaptive adversary. We also generalize our idea to the dishonest majority setting and achieve an improvement over existing protocols.

1 Introduction

Byzantine Agreement and Byzantine Broadcast are fundamental problems in distributed computing [3, 5, 17] and are important in creating fault-tolerant distributed systems, which allow systems to continue operating reliably and correctly even when some of its components fail. This problem has various important applications, including in cryptocurrency and blockchain protocols [1, 9, 15] or state-machine replication [19]. An important aspect of Byzantine Agreement and Broadcast protocols, which we focus on in this paper, is its round complexity, because by improving the round complexity, users in the system will reach agreement faster, which improves the speed and efficiency when applied in real world systems.

In the Byzantine Agreement problem, n users are each given an input bit and they must come to an agreement and output the same value, even in the presence of up to f corrupt users. Throughout this paper, we will use n to denote the total number of users in the system and f to denote the maximum number of corrupt users. A variation of the Byzantine Agreement problem is Byzantine Broadcast, where a single selected leader sends a bit b to all n users, and the honest users must agree on this bit. If the leader is honest, then all honest users must output the leader's proposed bit.

In this paper, we focus mainly on Byzantine Broadcast. However, we will also show that it is possible to convert our honest majority protocol from Byzantine Broadcast to Agreement by increasing the expected round complexity by 2 rounds. Note that Byzantine Agreement is only possible under honest majority, where $f < n/2$. Therefore, when we consider the problem under the dishonest majority scenario, we will only discuss Byzantine Broadcast.

There have been many existing works related to Byzantine Broadcast. Dolev and Strong [8] have shown a deterministic protocol that achieves $f + 1$ round complexity assuming digital signatures, which is also proved to be the lower bound for any deterministic protocol. However, this $f + 1$ lower bound does not apply in randomized protocols. Under honest majority, many works [2, 10, 11, 16] have shown protocols that achieve expected constant round complexity, including a paper by Abraham et al. [2] that had expected 10 round complexity for a static adversary, and 16 round complexity for an adaptive adversary, assuming digital signatures and public-key infrastructure, which to our knowledge is the best existing result thus far. Under dishonest majority, many protocols [6, 12, 13], have achieved sublinear round complexity. In particular, recently, a work by Wan et al. [20] has shown a protocol with expected $\Theta((\frac{n}{n-f})^2)$ round complexity, assuming public-key infrastructure and an idealized signature scheme. This means that when $f = (1 - \epsilon)n$ where $\epsilon \in (0, 1)$ is a small constant, this protocol achieves Byzantine Broadcast in expected constant rounds.

However, the optimal round complexity for Byzantine Broadcast is still unknown. Garay et al. [13] proved a $\Theta(n/(n - f))$ lower bound for any randomized Byzantine Agreement / Broadcast protocol. This result is interesting, but there remains a gap between this lower bound and the round complexity of existing protocols. Therefore, further improving the round complexity is important in order to determine the optimal result.

In this paper, we improve the round complexity of honest majority Byzantine Broadcast to expected 8 rounds under a static adversary, as well as show how we can convert the protocol to Byzantine Agreement under an honest majority by adding two rounds. We also show how we can modify the Byzantine

Broadcast protocol to be resilient under an adaptive adversary in expected 10 rounds. Furthermore, we discuss how we can modify the work by Wan et al. [20] to reduce the round complexity of dishonest majority Byzantine Broadcast.

First, in section 2, we provide the problem definition and our high level intuition. In section 3, we introduce our protocol for honest majority Byzantine Broadcast under a static adversary, as well as describe how the protocol can be converted to an honest majority Byzantine Agreement protocol. In section 4, we explain how this protocol can be modified to be resilient against an adaptive adversary, and in section 5, we compare our protocol to a previous paper by Abraham et al. [2], explaining how our protocol is able to achieve a reduced round complexity in comparison. Finally, in section 6, we show a protocol which improves the round complexity for the dishonest majority setting.

2 Preliminaries

2.1 Model and Problem Definition

Assumptions: Our protocol assumes a synchronous setting, meaning that a message sent by an honest user in round r is guaranteed to be received by another honest user before round $r + 1$. We also require a random leader election oracle. This leader election oracle can be instantiated using various methods such as with common-coin protocols [4][18], or with pseudo-random functions including verifiable random functions [7][20]. We will also use digital signatures as well as a public-key infrastructure. These assumptions are the same as compared to the previous work [2].
Adversary: We consider two types of adversaries, namely static and strongly rushing adaptive adversary. A static adversary cannot corrupt users in the middle of the protocol. A strongly rushing adaptive adversary, on the other hand, can corrupt any user at any point of the protocol, as long as the total number of corrupted users is upper bounded by f. Furthermore, if the adversary corrupts a user u in round r, it can not only inject additional messages into round r, but also alter or erase u's round-r messages before they reach other users.
Problem definition: We mainly focus on the Byzantine Broadcast problem in this paper. In this problem, suppose there are n users in the system. One of the users is the designated sender whose identity is known to all other users. At the start of the protocol, the designated sender will receive a bit b. All users then interact to learn the sender's bit b. At the end of the protocol, each user i outputs a bit b_i. We say that the protocol is correct and achieves Byzantine Broadcast if it meets the following conditions:

- Consistency: for any honest users i and j, $b_i = b_j$.
- Validity: if the leader is honest, then for any honest user i, $b_i = b$.
- Liveness: all honest users will eventually terminate.

2.2 Previous work and high level intuition

On the high level, the dilemma for Byzantine Broadcast / Agreement lies in the indistinguishability between honest and dishonest users. We illustrate this dilemma using the following example. Suppose a user u does not respond to another user v's query, then v knows that u must be corrupt. However, v cannot prove u's maliciousness to another user w since w cannot distinguish between (1) u not responding to v's query and (2) v dropping u's response and framing u as corrupt. Most previous works simply ignore this dilemma and cleverly get around it using properties that hold regardless of the adversary's actions. For example, in Dolev and Strong's protocol [8], they used the fact that a set of $f + 1$ votes must contain at least one honest user's vote. Another commonly used trick when $f < n/3$ is to require a user to gather $2n/3$ votes on the same bit before committing. If an honest user receives $2n/3$ users' votes on the bit $b \in \{0, 1\}$, then at least $2n/3 - f > n/3$ honest users have voted on b; thus, no other honest users can observe $n - f$ votes on $1 - b$.

These previous works do not detect and punish corrupt users. This means that corrupt users can perform some malicious actions, be detected, yet still participate in the rest of the protocol. This is why most previous works have to upper bound the number of corrupt users by $n/2$ or $n/3$ and their protocols do not work for dishonest majority.

Recap of Wan et al.'s Byzantine Broadcast Protocol: Wan et al. [20] provide an interesting observation on the previously described dilemma. Recall that in the previous example, a user w receives v's accusation on u for not responding, yet w cannot distinguish whether u is corrupt or honest.

However, w can still learn that at least one of u and v is corrupt, assuming that honest users always follow the protocol and never accuse each other. While w cannot be convinced of u or v's maliciousness just by one accusation, if enough accusations are gathered, for example, if $f+1$ users have accused u as corrupt, then w can be convinced that u must be a corrupt user. Further, w can use the $f+1$ accusations as proof to convince other honest users that u is corrupt. In order to avoid being proved as corrupt, the adversary should avoid easily detected malicious behavior and pretend to follow the protocol just like an honest user. This would restrict the power of the adversary and thus help honest users reach consensus.

Based on this observation, they construct a data structure called the trust graph. Each user maintains its own trust graph. In a user w's trust graph, the nodes represent the users and an edge (u, v) exists iff w thinks that u and v "trust" each other, i.e., w has not received any accusation between u and v. Initially, all honest users' trust graphs are complete graphs. But during the protocol, edges are removed as users detect malicious behavior from other users. Note that an accusation between two users u and v must be signed by either u or v so that the adversary cannot frame accusations between honest users.

Without getting into the details, the most important property of the trust graph structure is that *in any honest user's trust graph, the set of honest users must be fully connected.* Even though honest users might have different trust graphs, this property is always satisfied. This property also implies the following:

- If an edge (u, v) is removed from an honest user's trust graph, then at least one of u and v is corrupt.
- If a user's degree is less than $n-f$, then it has been accused by at least $f+1$ users. It is then proved as corrupt and should be removed from the trust graph. Therefore, any node in any honest user's trust graph must have degree at least $n-f$.

Using graph theory, it can be shown that a connected graph where every node has degree at least $n-f$ must have diameter less than $\Theta(n/(n-f))$. This result is crucial as the round complexity in [20] is tied with the diameter of the trust graph.

Using the trust graph data structure, they construct a primitive called "trustcast". The trustcast is a weaker primitive of consensus, similar to reliable broadcast or gradecast. Typically, these weaker primitives can only achieve one of consistency or liveness. The trustcast protocol differs from previous primitives because it is strongly tied with the trust graph structure. In the trustcast protocol, a sender s wants to send a message to all other users. At the end of the trustcast protocol, for any honest user u, either u receives a message from the sender or s is removed from u's trust graph. The latter implies that s has been proved by u as a corrupt user and can no longer participate in the rest of the protocol. Finally, this trustcast primitive is bootstrapped to full consensus using a random leader election oracle, even under dishonest majority.

We think that the trust graph data structure and the trustcast primitive in [20] are very interesting. However, their protocol focuses on dishonest majority and only cares about the asymptotic round complexity. We want to further improve the round complexity and adapt it to the honest majority setting. Inspired by the trust graph idea, we adopt a similar method for honest majority. We provide our technical roadmap in Section 2.3.

2.3 Technical highlight

Similar to the trust graph idea in [20], we want users to keep track of whether other users are corrupted or not. To do this, each user u maintains an $n \times n$ array $u.A$, initialized such that $u.A[v, w] = 1$ for all v, w. We define $u.A[v, w] = 1$ to represent that v and w trust each other in u's view. If v accuses w as corrupt, or the other way around, we will set $u.A[v, w]$ and $u.A[w, v]$ to 0. The array is symmetric, i.e., $u.A[v, w]$ is always the same as $u.A[w, v]$. We can think of them as two pointers pointing toward the same variable. If we change $u.A[v, w]$, then $u.A[w, v]$ is automatically changed as well.

This array resembles how the trust graph is stored in [20]. However, the way we maintain and use this array is much simpler. Initially, we set $u.A[v, w] = 1$ for any u, v and w, representing that all users trust each other at the start of the protocol. During the protocol, this array is updated in the following ways:

- If v is supposed to send a message yet u does not hear from v in this round, u knows that v is corrupt. It then sets $u.A[u, v] = 0$ and sends a Not-Trust message of v, which is a signature from u of the form
$$\text{Not-Trust}(u, v) = \mathsf{sig}_u(\text{“not trust”}, v),$$
to all other users in the next round.

  For any user w, upon receiving Not-Trust(u, v), it will set $w.A[u, v] = 0$ and relay the Not-Trust message in the next round.

- If u receives equivocating messages from v, u will set $u.A[v, w] = 0$ for all w. It will also relay the signed equivocating messages to all other users in the next round.

- We say that a user v is proved to be corrupt by an honest user u iff $u.A[v, w] = 0$ for all w.

Note that we set $u.A[v, w] = 0$ iff at least one of v and w is corrupt. Therefore, for any honest users u and v,
$$\sum_w u.A[v, w] \geq \#\text{ of honest users} = n - f. \tag{1}$$
Similarly, for any honest users u, v and w, $u.A[v, x] \cdot u.A[w, x] = 1$ as long as x is honest. Therefore, we have
$$\sum_x (u.A[v, x] \cdot u.A[w, x]) \geq n - f. \tag{2}$$
Based on Equation 1 and 2, an honest user u can constantly check its array $u.A$ to further detect corrupt users. For any honest user u,

- If $\sum_w u.A[v, w] < n - f$, then v must be a corrupt user. u can then set $u.A[v, w] = 0$ for all w.
- If $\sum_x u.A[v, x] \cdot u.A[w, x] < n - f$, then at least one of v, w must be corrupt. u can then set $u.A[v, w] = 0$.

This guarantees that Equation 1 and 2 hold for any v or for any v, w that has not been proved to be corrupt by u. This property will be used in our proof of the protocol's correctness.

We now explain the structure of and intuition behind the protocol itself. The basic goal of Byzantine Broadcast is that when the leader is honest, all honest users will agree on the leader's message and terminate. However, when the leader is dishonest, any malicious behavior by the leader will be detected by the honest users, so honest users will not commit on different messages. Specifically, we want the protocol to satisfy that

1. if the leader of an epoch i is honest, then all honest users terminate and output the leader's proposed bit in epoch i;
2. if two honest users terminate in the same epoch, then they must output the same bit; and
3. after an honest user terminates on bit $b \in \{0, 1\}$, leaders in future epochs cannot propose $1 - b$; therefore, other honest users cannot output $1 - b$ in future epochs.

The first property ensures that our protocol satisfies liveness and validity. The second and the third properties guarantee that our protocol satisfies consistency.

We construct a protocol which proceeds in multiple epochs, that satisfies these properties. In each epoch, a leader is selected who observes the results of previous epochs and proposes a bit for other users to agree on. The proposal is accompanied by proofs showing that the leader has indeed followed the protocol. Note that in the first epoch, the leader is simply the designated sender and should send its input bit. In the rounds following the leader's proposal, users exchange messages to determine if the leader was honest and if they should terminate. These messages are also used to construct the leader's proof for its proposal in future epochs.

Specifically, each epoch contains a Propose and Vote round, and 2 Commit rounds. The basic purpose of each round is:

1. **Propose:** The leader sends its proposal on a message $m \in \{0, 1\}$, accompanied by a proof for the proposal, to all users.

2. **Vote:** All users relay the proposal they received in a message that we call a "vote". This allows users to check if the leader sent equivocating proposals to different users, in which case they can prove that the leader was dishonest.

3. **1st round of Commit:** If a user did not detect equivocation from the leader, then they send commit messages containing *a commit evidence* $\mathcal{E}$. The commit evidences are sets of votes from other users and are used as proofs for a future leader's proposal. It makes sure that if an honest user has already committed, the leader cannot propose a different bit in future epochs.

4. **2nd round of Commit:** All users relay the commit messages they received in the first round of Commit to all other users. This ensures that if a dishonest user sent a commit evidence to only some honest users in the previous round, the message will still be relayed in this round so that all honest users receive all commit evidences.

Throughout the protocol, users also check if they have received at least $f+1$ commit messages on the leader's proposal, meaning at least one honest user has committed, at which point they can terminate.

Intuitively, these steps are necessary to ensure that the protocol is correct because a dishonest leader can behave maliciously either by (1) proposing equivocating messages to different users, or (2) by only sending proposals to some users, which may result in some users committing but not others. Through the vote round, we guarantee that users will detect the leader's equivocation in the first attack. Through the commit rounds, we guarantee that even when only some honest users commit as a result of the second attack, other honest users will be aware of this after receiving the commit messages, and not commit on a different message in future epochs. Namely, dishonest leaders will be unable to propose a different message in future epochs that would be accepted by the honest users. Throughout the protocol, users also keep track of any attacks and malicious behavior through their array A, allowing them to keep a record of which users may be dishonest, and therefore being more resilient to a dishonest leader's attacks.

We explain the protocol in more detail and provide more concrete proofs for the correctness of the protocol in section 3.

3 Honest Majority Protocol

In this section, we present our protocol under honest majority. We first describe some implicit operations that users always follow throughout the protocol. Then, we define the notations used in the protocol. We describe our protocol in Section 3.1, prove that it is correct in Section 3.2 and analyze its performance and other variations in Section 3.3 and 3.4.

Implicit assumptions / operations First, whenever an honest user u receives a new message, u relays it to all other users. This includes messages sent in the protocol, proof of equivocation and Not-Trust messages. This occurs in parallel with any new messages sent in the next round, so relaying does not add to the round complexity. Second, whenever a user sends a message m, it is always accompanied by a signature on the message m and the current round number. This prevents the adversary from reusing old signatures in previous rounds.

Throughout the protocol, each user u maintains their array $u.A$. In any round,

- if v is supposed to send a message yet u does not receive any, then u sets $u.A[u,v]=0$ and sends a Not-Trust message of v to all other users.
- if u receives a Not-Trust message between v and w, then u sets $u.A[v,w]=0$.
- if u detects equivocating signatures from v, then u sets $u.A[v,w]=0$ for all w.

After each round in the protocol, u performs the following array maintenance:

- For every v, if $\sum_w u.A[v,w] < n-f$, then u sets $u.A[v,w]=0$ for all w.
- For every v,w, if $\sum_x (u.A[v,x]\cdot u.A[w,x]) < n-f$, then u sets $u.A[v,w]=0$.

Recall that we say a user v is proved to be corrupt w.r.t. u iff $u.A[v,w]=0$ for all w.

Fact 3.1. *If v has not been proved to be corrupt w.r.t. u, then $\sum_w u.A[v,w] \geq n-f$ must hold. Otherwise, v would be detected by the array maintenance and thus proved to be corrupt.*

Commit evidence: We define some notations regarding commit evidence that will be used in the protocol. If a set $\mathcal{E}$ contains $f+1$ distinct users' votes on message m in epoch e, then we say that $\mathcal{E}$ is a commit evidence for (e, m). We call a commit evidence for (e, m) *fresher* than a commit evidence for (e', m) iff $e > e'$.

In our protocol, users send commit messages containing their commit evidence in the commit round. We say that a commit evidence is from user u if it is in the commit message from user u. A commit evidence from u is considered valid by another user v iff (1) the commit evidence contains at least $f+1$ votes on the same message and (2) u has not been proved to be corrupt w.r.t. user v. This means that if a user is already proved to be corrupt, then we will not consider its commit evidence in the protocol.

3.1 Protocol

We outline our Byzantine Broadcast protocol as follows:

Byzantine Broadcast under honest majority

For each epoch $e = 1, 2, \ldots$:

1. **Round 1** (propose):
   - **Send**: If $e = 1$, the designated sender sends its input bit to all users. If $e \geq 2$, the leader of this epoch L_e sends the freshest valid commit evidence [1] it has seen to all users. If L_e has not received any commit evidence in the previous epochs, it sends a random bit.
   - **Receive**: Every user $u \in [n]$, upon receiving L_e's proposal, verifies that the commit evidence in the proposal (if any) is as fresh as all other valid commit evidence u received. If u receives a random bit, check that they have not received any commit evidence. If not, meaning u received an invalid proposal, u ignores the proposal and pretends that it has not received from the leader.
2. **Round 2** (vote): For every user $u \in [n]$,
   - **Send**: if u received m from the leader, then send m to all users. If u did not receive from the leader, then send $\perp$ to all users.
   - **Receive**: For every user $v \in [n]$, if u receives a vote on $\perp$ from v, then set $u.A[v, L_e] = 0$.
3. **Round 3** (1st round of commit): For every user $u \in [n]$,
   - **Send**:
     - If $u.A[u, L_e] = 1$, i.e., u still trusts the leader, then u sends a message of the form $(\text{comm}, e, \mathcal{E})$ to all users, where the commit evidence $\mathcal{E}$ contains a set of votes for m from all users v such that $u.A[u, v] \cdot u.A[v, L_e] = 1$, meaning that u trusts v and v trusts the leader.
     - Else, send the message $(\text{comm}, e, \perp)$ to all users.
   - **Receive**: For every user $v \in [n]$: if v receives a non-$\perp$ commit message from u, v verifies whether the commit evidence is valid. If not, v ignores u's commit message.
4. **Round 4** (2nd round of commit): For every user $u \in [n]$,
   - **Send**: Relay any commit messages received in the previous round to all users.
   - **Receive**: If u did not receive any commit message from v in both rounds of commit, then for all w such that $u.A[w, v] = 1$, u sets $u.A[u, w] = 0$ and sends $\text{Not-Trust}(u, w)$ to all users in the next round. [2]

Terminate: If a user u has received $f+1$ commit messages for (e, m), it relays the $f+1$ commit messages as proof of termination to all other users. It then outputs m and terminates.

[1]Here, we require the commit evidence to be valid. So commit evidences from users proved to be corrupt won't be considered.

[2]This step is slightly counter intuitive. We will explain its reason in the proof of Lemma 3.2.

3.2 Proof of Correctness

In this section, we prove that our honest majority protocol satisfies the consistency, validity, and liveness conditions. First, we show that throughout the protocol, we never cause honest users to distrust each other, i.e., for any honest users u, v and w, $u.A[v, w] = 1$ always holds.

Lemma 3.2. *For any honest users u, v and w, $u.A[v, w] = 1$ holds throughout the protocol.*

Proof. We consider all the possible cases where u sets $u.A[v, w]$ to 0 and show that in all those cases, at least one of v and w is corrupt. In our protocol, a user u will update its array $u.A$ when (1) u fails to receive from another user, (2) u receives a new Not-Trust message or (3) u detects equivocating signatures. It is clear that these scenarios will not happen between honest users. Therefore, we only need to consider non-trivial cases such as when user u receives from v yet ignores the message because it is invalid. We analyze all possible cases according to the order they appear in the protocol.

Propose round: an honest user u sets $u.A[u, L_e] = 0$ (L_e is the leader of epoch e) if and only if

1. u does not receive any proposal from L_e or
2. u has a valid commit evidence $\mathcal{E}$ fresher than the commit evidence in L_e's proposal.

In the first case, clearly L_e is corrupt. We now focus on the second case. We will use proof of contradiction and assume that L_e is an honest user. Since $\mathcal{E}$ is a valid commit evidence, it must come from a user v who has not yet been proved corrupt w.r.t. u. By Fact 3.1, this implies that

$$\sum_w u.A[v, w] \geq n - f \geq f + 1.$$

So there must exist an honest user w such that $u.A[v, w] = 1$.

- if w received $\mathcal{E}$ from v in the first round of commit, w would relay $\mathcal{E}$ to u and L_e in the second round of commit. This violates our assumption that L_e has not received $\mathcal{E}$.
- if w received a commit message containing $\mathcal{E}' \neq \mathcal{E}$ from v in the first round of commit, then w would forward $\mathcal{E}'$ to u. u would then detect equivocation from v and prove v to be corrupt.
- if w did not receive from v in the first round of commit, w would send a Not-Trust message on v to all other users. This will cause u to set $u.A[v, w] = 0$, contradicting our assumption once again.

In all these cases, we reach a contradiction. Thus, L_e must be corrupt.

Vote round: An honest user u sets $u.A[v, L_e] = 0$ if v votes on $\perp$. v voting on $\perp$ implies that v has not received a valid proposal from the leader L_e. Therefore, at least one of v and L_e must be corrupt.

Commit round 1: An honest user u sets $u.A[u, v] = 0$ if (1) u does not receive v's commit message or (2) v's commit message contains invalid commit evidence. We focus on the second case. If v's commit evidence is not valid, then either v is already proved to be corrupt or the commit evidence does not contain $f + 1$ votes on the same message. In either possibility, v must be a corrupt user.

Commit round 2: An honest user u sets $u.A[u, w] = 0$ if there exists another user v such that (1) u does not receive v's commit message and (2) $u.A[v, w] = 1$. Again, we use proof of contradiction and assume that w is an honest user.

- If w received a valid commit message from v in the first round of commit, w would forward it to u, contradicting condition (1);
- If w did not receive a valid commit message from v in the first round of commit, w would send a Not-Trust message to u and thus $u.A[v, w]$ should equal 0, contradicting condition (2).

In both cases, we have reached a contradiction. Thus, w must be corrupt.

We have analyzed all the possible cases in the protocol and showed that an honest user will never set $u.A[v, w]$ to 0 for any honest users v and w. Given this, it is also clear that the array maintenance after each round will not alter $u.A[v, w]$ for honest v, w as well. Therefore, for any honest users u, v and w, $u.A[v, w] = 1$ always holds. This ends our proof. □

In fact, Lemma 3.2 is the hardest part in our proof. We now prove that the protocol is correct using similar proofs to [2]. First, we show that it satisfies consistency, which means that all honest users must output the same bit. In other words, if an honest user commits, then no other user can commit on a different message.

Lemma 3.3. *Suppose an honest user u generates a commit evidence on m^* in epoch e. If a commit evidence for (e, m) exists, then $m = m^*$.*

Proof. The commit evidence for (e, m) contains votes from at least $f + 1$ users for m, at least one of the votes must come from an honest user v. According to the protocol, v would have received the leader's proposal for m and forwarded it to all other users, including u. If $m \neq m^*$, u would have detected the leader's equivocation and would not have committed. Therefore, $m = m^*$. □

Lemma 3.4. *If at the start of epoch e, (1) every honest user has a commit evidence for (e', m), and (2) all conflicting commit evidence are less fresh, then the above two conditions will hold at the end of epoch e.*

Proof. We will prove this by contradiction. Suppose a user is able to acquire a fresher commit evidence for $m' \neq m$. Then, it must receive a vote for m' from at least one honest user u in epoch e. However, at the start of epoch e, u has a commit evidence for (e', m), so for u to vote for m', the leader must show commit evidence fresher than (e', m), which contradicts condition (2). □

We can then use induction to show that these conditions will continue to hold in all later epochs. Therefore, if an honest user commits on m in epoch e, all honest users will receive its commit evidence which is the freshest in epoch e. Then, in all future epochs $e' > e$, no honest user can see a commit evidence for $m' \neq m$. This leads to our proof for consistency.

Theorem 3.5 (Consistency)**.** *If two honest users output m and m' respectively, then $m = m'$.*

Proof. Suppose honest user u outputs m in epoch e. After the commit phase of epoch e, every honest user receives a valid commit evidence for (m, e). Furthermore, due to Lemma 3.3, there cannot be a commit evidence for $(m*, e)$ in epoch e for any $m* \neq m$. Thus, the two conditions in Lemma 3.4 hold at the end of epoch e. Therefore, in all future epochs, no commit evidence for a value other than m can be formed. All honest users can only commit and output on m in future epochs. □

Next, we show that the protocol satisfies validity, which means that if the leader is honest, then all honest users must output the leader's proposed bit, and liveness, meaning that everyone will terminate.

Theorem 3.6 (Validity and Liveness)**.** *If the leader L_e in epoch e is honest, then every honest user will terminate in that epoch.*

Proof. In Lemma 3.2, we have shown that for any honest users u, v and w, $u.A[v, w] = 1$ holds throughout the protocol. Since the leader L_e is honest, this implies that for any honest user u, $u.A[u, L_e] = 1$.

In the propose round, L_e's proposal will be accepted by all honest users. Otherwise, if an honest user u ignores the leader's proposal, u would set $u.A[u, L_e]$ to 0, contradicting Lemma 3.2.

In the vote round, all honest users will vote on L_e's proposal and receive each other's votes. These votes then comprise a commit evidence. In the first round of commit, all honest users would receive each other's commit messages. Each honest user would receive at least $n - f \geq f + 1$ commit messages on the leader's proposal. This matches the termination condition and all honest users will terminate on the same message. □

3.3 Round and Communication Complexity

The expected round complexity of our honest majority protocol is 8 rounds. This is because each epoch takes 4 rounds, and by Theorem 3.6, if the leader L_e is honest, then all users will reach the terminate condition in the first round of commit and terminate one round later.

With random leader selection, it takes $n/(n - f) \leq 2$ epochs in expectation to get an honest leader. Therefore, the entire protocol will terminate in expected 8 rounds.

The communication complexity is $\tilde{O}(n^4)$. This is because in each round, $O(n)$ users are sending messages to $O(n)$ other users. Since users must relay any new messages it receives, the total message size is $O(n^2)$ because there are at most $O(n^2)$ Not-Trust messages that need to be relayed. This results in a final global communication complexity of $\tilde{O}(n^4)$. Our result is far from the current optimal which is $\tilde{O}(n^2)$. Reducing the communication complexity is a direction for future work.

3.4 Conversion from Byzantine Broadcast to Agreement

The protocol described in this paper only solves the Byzantine Broadcast problem, but under honest majority, being able to solve Byzantine Agreement, where all users start with an input bit, is also important. Converting from Byzantine Agreement to Broadcast can be achieved by adding a round at the beginning of the protocol where the leader sends its input bit to all users.

However, the conversion from Byzantine Broadcast to Agreement for our protocol is not as trivial and requires adding 2 “pre-rounds” to the beginning of our protocol, resulting in a Byzantine Agreement protocol with expected 10 round complexity, assuming honest majority. This round complexity is the same as the state of the art in [2].

In the first round that we add, all users send their input bits to all other users. In the second round, all users send votes on every bit they received, which contains their signature, to all other users. Now, we can proceed with our Byzantine Broadcast protocol, with the leader proposing the bit b that it received the majority of signatures on. When users receive this proposal, they will check that they did not receive $1-b$ as a valid input bit from more than $n/2$ users.

Intuitively, we can think of a set of $f+1$ votes on any bit b as an initial commit evidence. If all honest users have the same input bit b, they would hold a commit evidence for b. If the leader proposes $1-b$, the leader will be proved as corrupt and the proposal will not be accepted. We describe what the users do in each of the pre-rounds more specifically below.

1. **Pre-round 1**: Every user $u \in [n]$ sends its input bit $b \in \{0,1\}$ to all users.
2. **Pre-round 2**: Every user $u \in [n]$ performs the following:
   - **Send**: Relays the input bits received from other users in the previous round.
   - **Receive**: If u did not receive the input bit from v, then for all w such that $u.A[w,v]=1$, u sets $u.A[u,w]=0$ and sends Not-Trust(u,w) to all users in the next round.

Lemma 3.7. *For any honest user u, at the end of the Pre-round 2, for any user v, either (1) u has received v's input bit or (2) v has been proved to be corrupt w.r.t. u.*

Proof. If u has not received v's input bit, then by the protocol, u would set $u.A[u,w]=0$ for all w such that $u.A[w,v]=1$. Therefore, $u.A[u,w]+u.A[w,v] \le 1$ for all w. By summing this up for all w, we have

$$\sum_w u.A[u,w] + \sum_w u.A[w,v] = \sum_w (u.A[u,w]+u.A[w,v]) \le n. \tag{3}$$

By Lemma 3.2, $u.A[u,w]=1$ for all honest w; thus, $\sum_w u.A[u,w] \ge n-f$. Combining this with Equation 3, we have

$$\sum_w u.A[w,v] \le n - \sum_w u.A[u,w] \le f.$$

This implies that v must be a corrupt user. By the array maintenance mechanism, u will set $u.A[w,v]$ to 0 for all w. Therefore, v is proved to be corrupt w.r.t. u. □

Now, we can invoke our Byzantine Broadcast protocol from section 3, where in epoch $e=1$, the propose round is executed as follows:

- **Propose**: Suppose the leader of the first epoch is L,
  - **Send**: Let S be the set of users not proved to be corrupted w.r.t. L. By Lemma 3.7, L has received the input bit from every user in S. L proposes the majority among the input bits in S and uses the input bits (with signatures) in S as proof.
  - **Receive**: For any user u, upon receiving the leader's proposal, verify that the proof contains the input bits for every v not proved to be corrupt w.r.t. u.

The intuition here is that if L is honest, then it will relay all the Not-Trust messages to other honest users. So if a user is proved to be corrupt w.r.t. L, then it will be proved to be corrupt w.r.t. any other honest user in the next round. Therefore, if the leader is honest, all honest users will accept the leader's proposal. After this propose round, the protocol can be run identically as in the Byzantine Broadcast protocol.

We can also show this satisfies the additional validity condition that if all honest users start with the same input bit b, then they all commit on b. This is because all honest users would send b in the first pre-round and vote for other honest users sending b in the second pre-round, so at least half of the initial bits would support b, and an honest leader would propose b.

4 Adaptive Adversary

Our protocol can be modified to be resilient against a strongly rushing adaptive adversary. This means the adversary can adaptively decide which users to corrupt each round as long as the number of corrupt users is upper bounded by f, and it can decide which users to corrupt in round r after observing the messages sent in round r. After corrupting a user in round r, it can erase, alter, or inject additional messages in the same round. This creates a problem for our original protocol because it means that, after learning the leader's identity in the Propose phase of each epoch, the adaptive adversary can repeatedly corrupt the leader in each epoch and generate equivocating proposals, preventing users from committing. In a paper by Abraham et al. [2], they describe modifications that allow their protocol to be resilient against a strongly rushing adaptive adversary. We apply the same ideas in our protocol.

Overall, we want to ensure that the adversary cannot forge proposals from the leader even if the adversary corrupts immediately after the leader election. To do this, we make two main changes to the protocol. First, we postpone the leader election until the Propose round is over. In the Propose round, every user acts as a potential leader and sends a proposal to all users. After all the proposals have been delivered, the leader is elected via a common-coin protocol [4]. The leader election can be performed in parallel with the Vote round, where users continue to vote on all proposals, acting as if all of them could be a potential leader. After the leader is revealed, we proceed to the Commit rounds where only votes on the proposal from the elected leader are considered. All other votes and proposals are simply ignored.

Second, we must ensure that the adversary still cannot forge proposals if it corrupts the leader after the Propose phase. We do this by adding another round to allow users to "prepare" their proposals. We say that a proposal is *prepared* iff it is signed by at least $f+1$ distinct users, at least one of which must be an honest user.

- In the first round of Propose, users send their proposals to all users.
- In the second round of Propose, users sign all the proposals they have received and relay them to all other users.

After the leader's identity is revealed, a user u votes on a message m iff (1) u received a valid proposal on m from the leader in the first round and (2) u received at least $f+1$ distinct signatures on the proposal in the second round. Note that

- If the leader is honest during Propose, then all honest users would receive the leader's proposal and relay it. Therefore, all honest users would receive a prepared proposal.
- For any user u and leader L, if $u.A[u, L] = 1$ at the end of Propose, then u would receive a prepared proposal. This is because our array maintenance guarantees that

$$u.A[u, L] = 1 \implies \sum_{v} u.A[u, v] \cdot u.A[v, L] \geq f+1,$$

  for any v such that $u.A[u, v] = 1$ and $u.A[v, L] = 1$, v would receive the leader's proposal in the first round and relay it to u in the second round.

Therefore, the proof of correctness in Section 3.2 still applies.

We outline a summary of the steps for each epoch of the adaptive adversary protocol below.

- **Round 1** (Propose): Every user $u \in [n]$ sends its proposal to all other users. The proposal is determined in the same way as the static adversary protocol.
- **Round 2** (Propose): Every user $u \in [n]$ signs and relays any proposal it received to all other users.
- **Elect**: Every user $u \in [n]$ participates in a threshold coin-tossing scheme from [4] to elect a leader L_e. The elect round occurs in parallel with Vote. Therefore, it does not contribute extra round complexity.
- **Round 3** (Vote): A user votes on every message m that it has received a prepared proposal on.
- **Round 4** (Commit): If a user u has received votes on the prepared proposal from the elected leader from every user v such that $u.A[u, v] \cdot u.A[v, L_e] = 1$, send a commit message and evidence to all users.
- **Round 5** (Commit): Same as the static adversary protocol.

Terminate: Same as the static adversary protocol.

Overall, our round complexity under a strongly-rushing adaptive adversary is now 5 rounds per epoch, and expected 10 rounds in total, and proofs of correctness are still the same as the static adversary case.

5 Comparison to [2]'s Byzantine Agreement Protocol

Our protocol improves the round complexity of Byzantine Broadcast under the honest majority setting, in comparison to the state of the art results provided in a previous paper by Abraham et al. [2] which showed protocols to achieve both Byzantine Agreement and Broadcast in expected 10 rounds for the static adversary and 16 rounds for the adaptive adversary. In this paper, we mostly focus on the Byzantine Broadcast case. We compare this protocol with ours and explain how our protocol is able to achieve Byzantine Broadcast with fewer rounds.

Under a static adversary, the protocol in [2] has epochs each consisting of 4 rounds: status, propose, commit, and notify. In the propose rounds for both protocols, the leader sends its proposal. Their commit round corresponds to our vote round, and their notify and status rounds correspond to our two commit rounds. While our protocol shares similar structures with theirs, the detailed way in which each round is executed is very different. By maintaining an array that records the trust relationship for each user, we restrict the power of the adversary and thus reduce the round complexity.

Specifically, we are able to reduce the expected round complexity from 10 to 8 for our protocol because we do not need the extra "pre-round" or a terminate round, as used in their protocol. Their protocol requires a "pre-round" for Byzantine Broadcast where the leader first broadcasts its input bit to all users. This is because their protocol begins with the status round, where all users must start with a current accepted value to send to the leader, and the leader uses this to construct its proposal. We can avoid this by having our protocol begin with the propose round, and in epoch 1 the leader simply proposes its input bit. Furthermore, when the leader is honest, users will reach termination in only three rounds. This spares us another round.

This advantage further reduces the round complexity under the adaptive adversary setting. In [2], their adaptive adversary protocol requires 16 rounds due to the addition of two rounds to prepare the proposal and a separate elect round for each epoch. We only need one round to prepare a proposal, and we are able to run the elect round in parallel with the vote round, resulting in a total of 10 rounds for the adaptive adversary.

However, the communication complexity in [2] is $\tilde{O}(n^2)$, which is better than ours.

6 Dishonest Majority Protocol

In this section, we show how to adapt our protocol for the dishonest majority scenario, where $f > n/2$. Our protocol achieves an improved round complexity in comparison to the current best result obtained by the Byzantine Broadcast protocol from [20].

6.1 Recap of [20]'s dishonest majority Byzantine Broadcast protocol

As described in section 2.2, in Wan et al.'s protocol [20], each user w maintains a trust graph, where the nodes represent users and an edge (u, v) represents that u and v trust each other in w's view. If a user u thinks that v is dishonest, then they send a Distrust message to other users, which is similar to the Not-Trust messages we used in our honest majority protocol. In [20], Wan et al. also proved that all honest users will remain direct neighbors with each other in their trust graphs throughout the protocol. Also as mentioned in section 2.2, every node in the trust graph must have degree at least $n - f$, so the diameter of this trust graph, which we will call d for the rest of the paper, is upper bounded by $\Theta(n/(n - f))$.

Then, users in the trust graphs exchange messages with each other using the "TrustCast" protocol, which required d rounds. To summarize how the TrustCast protocol works, suppose a sender s wants to send a signed message m to everyone. Then, for every round $1 \leq r \leq d$, each user u checks if they have received a message signed by s, using a verification function Vf to check the validity of the message. If u does not receive a valid message, then u knows that any of their direct neighbors that are distance less than r away from s in u's trust graph must be dishonest. Otherwise, they would have received s's message within $r - 1$ rounds and sent it to u in the r^{th} round. This process guarantees that by the end of the TrustCast protocol, any honest node will either receive a valid message signed by s or remove s from their trust graphs, which is proved in Theorem 4.1 of [20].

The dishonest protocol in [20] also consists of epochs with the same Propose, Vote, and Commit phases, and the purpose of each phase is similar to the honest majority protocol. However, in each of these phases, rather than simply sending a message to all users, each user uses the TrustCast protocol to send their messages, meaning that each phase requires d rounds. They then use the verification functions $\mathrm{Vf}_{\mathrm{prop}}$, $\mathrm{Vf}_{\mathrm{vote}}$, and $\mathrm{Vf}_{\mathrm{comm}}$ in order to check that the messages they receive are valid in the propose, vote, and commit phase, respectively. Essentially, using these verification functions, the protocol guarantees the following for any honest node u in epoch e:

- At the end of propose, either the leader L_e is removed from u's trust graph, or u has received a valid proposal from the leader.
- At the end of vote, either the leader L_e is removed from u's trust graph, or u has received votes on the leader's proposal from every user in u's trust graph.
- At the end of commit, either the leader L_e is removed from u's trust graph, or u has received a valid commit evidence on the leader's proposal from every user in u's trust graph.

6.2 Intuition for dishonest majority protocol

We improve the round complexity of Wan et al.'s protocol [20] by reducing the number of rounds for the propose and vote phase from d to $d-1$, where d denotes the diameter of the trust graph, and is defined to be $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$ as shown in [20]. Thus, each epoch will require $3d-2$ rounds, an improvement by two rounds from the original $3d$ rounds. We do this by only running the TrustCast protocol for $d-1$ rather than d rounds in the propose and vote phases. This means that when a sender s trustcasts a messages m, for every round $1 \leq r \leq d-1$, every user u checks if they have received s's message, and if not, removes their edges with any neighbors that are less than distance r from s in their trust graphs. Therefore, by the end of this modified TrustCast protocol, any honest node u will either receive s's message, remove s from their trust graphs, or there will be a distance d between u and s in u's trust graph. In other words, the difference compared to Wan et al.'s protocol [20] is that a honest node u will not receive messages from s in the propose and vote phases, even if s remains in u's trust graph, if s is distance d away from u in u's trust graph.

To provide intuition for why the protocol still works despite users not receiving some messages as a result of this lower round complexity, we show that it is not necessary to exchange messages with all users, but rather it is enough if at least one honest user receives valid messages. Specifically, in the Propose phase, rather than the leader sending their proposal to every user, it is enough to just make sure that at least one honest user receives the proposal. In the Vote phase, rather than receiving votes from everyone in their trust graphs, we ensure that every honest user will receive a vote from at least one honest user who also received the leader's proposal. In Appendix A.2, we prove that these conditions are guaranteed to be met even with the reduced round complexity, and that as long as this is true, the protocol will still satisfy the necessary consistency, validity, and liveness properties. To account for the fact that users will no longer receive messages from users that are a distance d away from them in their trust graphs during the propose and vote phases, we modify various aspects of the protocol such as the requirements for the commit evidence and verification functions. Again, in Appendix A.2, we explain these changes in more detail.

Note that we are focusing on the static adversary case for the dishonest majority protocol, but it has been shown in [20] that the protocol can be modified to be resilient under a weakly adaptive adversary, which means that if the adversary corrupts a user in round r, they can inject messages into round r, but they cannot erase messages already sent in round r. This type of adversary has been defined previously in [14].

7 Conclusion

The honest majority protocol described in this paper achieves Byzantine Broadcast in expected 8 rounds. This is an improvement from previous honest majority results. Although our protocol can also be converted to Byzantine Agreement, doing so required adding 2 rounds for total 10 rounds, which is equivalent to existing results. This paper also explained a protocol under a strongly rushing adaptive adversary that achieves Byzantine Broadcast in expected 10 rounds, which is an improvement over previous results.

Finally, we showed a method to reduce the round complexity of Byzantine Broadcast under the dishonest majority setting by 2 rounds per epoch. Future work includes improving the communication complexity and exploring the optimal round complexity for Byzantine Broadcast.

A Dishonest Majority Protocol

A.1 Protocol

Throughout the protocol, we use the notation $\text{TrustCast}^{\text{Vf},s}$ to mean that a user s is sending a message using the TrustCast protocol, and all users u should use the verification function Vf to verify the messages they receive during this protocol. Originally, the TrustCast protocol runs for d rounds, which is the diameter of the trust graphs. However, for our protocol, in the propose and vote phases, when we call the TrustCast protocol we are only running it for $d-1$ rounds. The final protocol for dishonest majority Byzantine Broadcast is described below:

Byzantine Broadcast under dishonest majority

For each epoch $e = 1, 2, \ldots$:

1. **Propose** ($d-1$ rounds)
   - **Send:** The leader of this epoch L_e calls the $\text{TrustCast}^{\text{Vf}_{\text{prop}},L_e}$ protocol for $d-1$ rounds to TrustCast a proposal of the form $(\text{prop}, e, (b, \mathcal{E}))$, where
     - if $e = 1$, then the proposed bit b is the leader's input bit and the commit evidence $\mathcal{E}$ is $\perp$.
     - if L_e has not received any commit evidence, then b is a random bit and $\mathcal{E}$ is $\perp$.
     - if L_e has seen a non-$\perp$ commit evidence, then $\mathcal{E}$ is the freshest commit evidence L_e has seen and b is the bit that the commit evidence was vouching for.
   - **Receive:** For every user $u \in [n]$, if u receives L_e's proposal during the $\text{TrustCast}^{\text{Vf}_{\text{prop}},L_e}$ protocol, u verifies the leader's proposal using the Vf_{prop} function, which is defined as follows: $u.\text{Vf}_{\text{prop}}(\text{prop}, e, (b, \mathcal{E})) = \texttt{true}$ in round r iff the following holds:
     (a) either $\mathcal{E} = \perp$, or $\mathcal{E}$ is a valid commit evidence supporting the proposed bit b; and
     (b) either $e = 1$, or $\mathcal{E}$ is at least as fresh as any commit evidence TrustCasted by u in *all* previous epochs — note that we treat $\perp$ as a valid commit evidence for epoch 0.

2. **Vote** ($d-1$ rounds): For every user $u \in [n]$,
   - **Send:** call $\text{TrustCast}^{\text{Vf}_{\text{vote}},u}$ for $d-1$ rounds to TrustCast a vote of the form (vote, e, b'), where
     - if u received L_e's proposal, then $b' := b$ where $b \in \{0, 1\}$ is L_e's proposed bit
     - if u did not receive L_e's proposal, then $b' := \perp$
   - **Receive:** For every user $v \in [n]$, if v receives u's vote during the $\text{TrustCast}^{\text{Vf}_{\text{vote}},u}$ protocol, v verifies u's vote using the Vf_{vote} function, which is defined as follows: $v.\text{Vf}_{\text{vote}}(\text{vote}, e, b') = \texttt{true}$ in round r iff either
     (a) L_e is removed from G_v^r,
     (b) L_e is distance d away from u in G_v^r,
     (c) or b' matches L_e's proposal (in v's view).

3. **Commit** (d rounds): For every user $u \in [n]$,
   - **Send:** call $\text{TrustCast}^{\text{Vf}_{\text{comm}},u}$ to TrustCast a commit message of the form $(\text{comm}, e, \mathcal{E})$, where
     - if all users v such that $d(v, L_e, G_u) < d$ and $d(v, u, G_u) < d$, voted for the same bit $b \in \{0, 1\}$, then $\mathcal{E}$ contains a graph G, which represents u's trust graph at the end of the Vote phase, and a signed vote message of the form $(\text{vote}, e, \_)$ from all users v such that $d(v, L_e, G) < d$ and $d(v, u, G) < d$. Furthermore, u **outputs** the bit b.
     - Else, $\mathcal{E} := \perp$.

- **Receive:** For every user $v \in [n]$, if v receives a commit message from u's during the $\text{TrustCast}^{\text{Vf}_{\text{comm}},u}$ protocol, v verifies u's commit message using the Vf_{comm} function, which is defined as follows:
  $v.\text{Vf}_{\text{comm}}(\text{comm}, e, \mathcal{E}) = \texttt{true}$ in round r if either L_e is removed from G_v^r, or $\mathcal{E}$ is a valid commit evidence for (e, b), where b is L_e's proposed bit.

Terminate: If a user u receives commit evidences for (e, b) from everyone still in their trust graphs, then relay the commit messages and terminate.

A.2 Reducing Round Numbers from d to $d - 1$

A.2.1 Propose Phase

In the following sections, we explain in more detail the aspects of the protocol that are changed in comparison to the protocol in [20] in order to accommodate for the reduced round complexity, and how our protocol still works with these changes.

First, if we reduce the round number of the Propose phase from d to $d - 1$, then for any user u such that the leader is removed from u's trust graph or that u is distance d away from the leader in u's trust graph by the end of the Propose phase, u will not receive the leader's proposal. As a result, u will not be able to TrustCast the leader's proposal in the Vote phase. However, u may still be honest and their vote should still be verified. Therefore, we need to modify the verification function Vf_{vote} so that votes from these users will be valid. The original Vf_{vote} function is defined such that in round r, $v.\text{Vf}_{\text{vote}}(\text{vote}, e, b') = \text{true}$ iff (1) either the leader is removed from G_v^r, or (2) b' agrees with the leader's proposed bit (in v's view).

We modify Vf_{vote} so that a user v verifies user u's vote iff either (1) u has not received from the leader, or (2) u's vote agrees with the leader's proposed bit. A user u would not receive from the leader if, at the end of the Propose phase, u and L_e were more than distance $d - 1$ away from each other in u's trust graph (note that if L_e has been removed from u's trust graph, we can essentially consider L_e to be more than distance d away from u). Another user v can verify if u did not receive a proposal from the leader by checking if u and L_e are more than distance $d-1$ away from each other in v's trust graph at the time that v receives u's vote, i.e. $d(u, L_e, G_v) > d - 1$. This works because of the monotonicity invariant from [20], which guarantees that an honest user's trust graph in round t is a subset of any honest user's trust graph in any round $r < t$.

This means that G_v at the time that v receives u's vote is a subgraph of G_u at the end of Propose, so $d(u, L_e, G_v) \geq d(u, L_e, G_u) > d-1$. Specifically, if v receives (vote, e, b') from u, then the Vf_{vote} function is defined below. Again, note that when we say $d(u, L_e, G_v) > d$, this is equivalent to saying that L_e is removed from G_v because the maximum diameter of the trust graph is maintained to be d.

$$v.\text{Vf}_{\text{vote}}(\text{vote}, e, b') = \begin{cases} \text{true} & \text{if } d(u, L_e, G_v) \geq d \\ \text{true} & \text{if } b' \text{ agrees with } L_e\text{'s proposed bit} \\ \text{false} & \text{otherwise} \end{cases}$$

We have now modified the protocol so that votes from users that may still be honest but did not receive from the leader will still be verified. However, we must also ensure that if the leader remains in an honest user u's trust graph, u should receive a valid vote on the leader's proposal from at least one honest user, meaning the leader's proposal reaches at least one honest user in $d - 1$ rounds. We show this using the fact that in the trustcast protocol, if a user u has not received a message from a sender s in round r, then s must be at distance at least $r + 1$ from u or removed from u's trust graph. By setting the round number r to $d - 1$, we show that if in round $d - 1$, the leader remains in a user u's trust graph, then any v such that $d(v, L_e, G_u) < d$ will receive the leader's proposal. Furthermore, it has been shown in [20] that $|\{v \mid d(v, L_e, G_u) = d\}| \geq n - f$ is impossible due to the properties of the trust graph. Therefore, there is at least one honest user v where $d(v, L_e, G_u) < d$ and will receive the leader's proposal, so u will receive the vote of at least one honest user v.

We also must change the requirements for a commit evidence from a user u to contain a graph G which represents u's trust graph at the end of the Vote phase, and a set of votes from all users v such that $d(v, L_e, G) < d$. The Vf_{comm} function then checks that this commit evidence is valid iff it contains votes from every v such that $d(v, L_e, G) < d$. If a user did not commit, the Vf_{comm} function checks that the leader is removed from their trust graphs.

A.2.2 Vote Phase

If we additionally reduce the number of rounds in the Vote phase from d to $d-1$, as explained previously, using the TrustCast protocol, users will no longer receive votes from other users that are distance d away from them in their trust graphs. Instead, users will receive votes from all users that are within a distance of $d-1$ away from them. In other words, for any user u, if the leader remains in u's trust graph, then u must receive a vote on the leader's proposal from any v such that $d(v, L_e, G_u) < d$, meaning v received the leader's proposal, and $d(u, v, G_u) < d$, meaning u receives v's vote. Therefore, we again modify the commit evidence $\mathcal{E}$ from a user u to contain a graph G which represents u's trust graph at the end of the Vote phase, and a set of votes from all users v such that $d(v, L_e, G) < d$ and $d(v, u, G) < d$.

The $\mathrm{Vf}_{\mathrm{comm}}$ function remains the same: if the commit condition, which is that the leader remains in their trust graphs, is met, then it verifies the commit evidence is valid. If a user did not commit, it verifies that the leader is removed from their trust graphs. However, the difference is that the requirements for a commit evidence to be valid are modified. A commit evidence from u is considered valid by another user w iff it contains votes on the same message from every user v such that $d(v, L_e, G) < d$ and $d(v, u, G) < d$. Also, note that by the monotonicity invariant from [20] defined earlier, w's trust graph should be a subset of G.

Next we show that our protocol will still work with $d-1$ rounds for the Vote phase, because if the leader remains in a user's trust graph, then they will receive votes on the leader's message from at least one honest user. In the following lemma, we prove that any honest user u will receive a vote on the leader's proposal from at least one honest user, meaning there is at least one honest user that is less than distance d away from both the leader and u.

Lemma A.1. *For any honest user u, if the leader remains in u's trust graph, then there is at least one honest user v that satisfies $d(v, L, G_u) < d$ and $d(u, v, G_u) < d$.*

Proof. We can prove this by contradiction. Suppose that for all honest users, either $d(v, L, G_u) = d$ or $d(v, u, G_u) = d$. Denote S_L to be the set of honest users such that $d(v, L, G_u) = d$, and S_u to be the set of honest users such that $d(v, u, G_u) = d$ yet $d(v, L, G_u) \neq d$.

We are assuming that all $n-f$ honest users must be contained in either S_L or S_u. However, $|S_L|$ and $|S_u|$ alone must each be less than $n-f$ by the properties of the trust graph described in [20], so neither set can be empty.

Since S_L and S_u contain only honest users, they must be directly connected. Also, because any user in S_L is distance d away from L, any user in S_u can be distance $d-1$, d, or $d+1$ away from L. However, distance d is impossible due to the definition of S_u, and $d+1$ is impossible due to properties of the trust graphs, as the trust graph can have maximum diameter of d. Therefore, any user in S_u must be distance $d-1$ from L.

Suppose the shortest path between L and a node in S_u is $L_0 - L_1 - L_2 - \cdots - L_{d-1}$ where $L_0 = L$ and $L_{d-1} \in S_u$. Since all nodes in S_L are connected to L_{d-1}, for any node $w \in S_L$, $L_0 - L_1 - L_2 - \cdots - L_{d-1} - w$ must be a path in u's trust graph. Further, by definition of S_L, this should be one of the shortest paths between L and w. Therefore, w.l.o.g., we can assume that the shortest path between L and any node in S_L is $L_0 - L_1 - L_2 - \cdots - L_d$ where $L_0 = L$, $L_{d-1} \in S_u$ and $L_d \in S_L$.

Denote $S(L, i)$ to be the set of all intersections between $N(L_i)$ and $N(L_{i+1})$, where $N(v)$ is the set of v's neighbors. By the properties of the trust graph, we know that $S(L, i) \geq n - f$ and for any i, $S(L, i)$ and $S(L, i+2)$ must be disjoint. Then, suppose $\mathrm{SET} = S(L, 0) \cup S(L, 1) \cup \cdots \cup S(L, d-1)$. Since $L_{d-1} \in S(L, d-2)$, the distance between L_{d-1} and any node in $\mathrm{SET} \setminus L$ is at most $d-2$.

Since $d(u, L_{d-1}, G_u) = d$, u is not in SET, so now we consider the shortest path between u and L_{d-1}, $u_0 - u_1 - \cdots - u_{d-1} - L_{d-1}$. We similarly define $S(u, i)$ as the set of intersections between $N(u_i)$ and $N(u_{i+1})$. Since $d(u_1, L_{d-1}) = d - 1$, u_1 is not in SET either, meaning $S(u, 0)$ is not in SET. Since $|S(u, 0)| = n - f$, if $|\mathrm{SET}| > f$ then we reach a contradiction as there would have to be more than n nodes.

We know that $|\mathrm{SET}| > f$ using a similar method to the one used in Claim 3.1 of [20]. We use the symbol $h = n - f$ to denote the number of honest users.

- If n is not divisible by h, then suppose $n = k \cdot h + l$ where k is the quotient of n divided by h and l is the remainder. By definition, $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1 = 2k$.

- If n is perfectly divisible by h, i.e., $n = k \cdot h$, then $d = 2k - 1$.

In both cases,

$$\begin{aligned}|\mathrm{SET}| &= |S(L,0)\cup S(L,1)\cup\cdots\cup S(L,d-1)|\\ &\geq |S(L,0)\cup S(L,2)\cup\cdots\cup S(L,2k-2)|\\ &= |S(L,0)|+|S(L,2)|+\cdots+|S(L,2k-2)|\\ &\geq h\cdot k > n-h.\end{aligned}$$

We have now shown that if there is no honest user v such that $d(v,L,G_u) < d$ and $d(u,v,G_u) < d$, then we reach a contradiction as there would have to be more than n users.

□

Through Lemma A.1, we have proved that we can reduce the round number of both the Propose and Vote phases each from d to $d-1$, and, if the leader remains in their trust graphs, every honest user will still receive a vote on the leader's proposal from at least one honest user. Overall, the improved round complexity is now $3d-2$ rounds per epoch, compared to $3d$ rounds per epoch in [20], where the protocol requires expected $n/(n-f)$ epochs to terminate.

A.3 Monotonicity and Validity at Origin

In this section, we prove that our modified verification functions continue to satisfy the monotonicity and validity at origin conditions described in [20]. These conditions are necessary to ensure that all honest nodes remain direct neighbors in each others' trust graphs throughout the protocol.

- *Monotonicity Condition from [20]:* Suppose $r < t$ and u,v are honest users. If $u.\mathrm{Vf}(m) = \mathrm{true}$ in round r, then it must be that $v.\mathrm{Vf}(m) = \mathrm{true}$ in round t.
- *Validity at Origin from [20]:* If the leader s is honest, it must be that $s.\mathrm{Vf}(m) = \mathrm{true}$ in round 0.

Our proofs that the modified Vf functions will still satisfy these conditions are similar to the proofs in [20], with a few changes.

A.3.1 Monotonicity Condition of Verification Functions

Lemma A.2. *Vf_{vote} satisfies the monotonicity condition.*

Proof. Suppose u,v are honest nodes, and $r < t$. If (vote,e,b') passes $\mathrm{Vf}_{\mathrm{vote}}$ w.r.t. node u in round r, then in round r, either $L_e \notin G_u^r$, $d(u,L_e,G_u^r) = d$, or $d(u,L_e,G_u^r) < d$ and u heard L_e propose the same bit $b \in \{0,1\}$.

In the first case, we know that in round t, $L_e \notin G_v^t$ by the monotonicity invariant and so in round t, $(vote,e,b')$ must pass the verification function $\mathrm{Vf}_{\mathrm{vote}}$ w.r.t. the node v.

If the second case happens, then in round t, $d(u,L_e,G_v^r) = d$, or $d(u,L_e,G_v^r) > d$ and L_e is removed from v's trust graph, by the trust graph monotonicity lemma, and thus in round t, (vote,e,b') must pass the verification function $\mathrm{Vf}_{\mathrm{vote}}$ w.r.t. the node v.

In the third case, if in round t, $L_e \notin G_v^t$ then $\mathrm{Vf}_{\mathrm{vote}}$ would pass w.r.t. v in round t. Next, we discuss the case when $L_e \in G_v^t$. In this case, v must have heard L_e TrustCast the same proposal since otherwise u would have sent to v the leader's equivocating proposal, and v would have removed L_e from its trust graph. Therefore, in round t, (vote,e,b') must pass the verification function $\mathrm{Vf}_{\mathrm{vote}}$ w.r.t. the node v. □

The proofs for $\mathrm{Vf}_{\mathrm{prop}}$ and $\mathrm{Vf}_{\mathrm{comm}}$ are still the same as in [20].

A.3.2 Validity at Origin

The validity at origin proofs for $\mathrm{Vf}_{\mathrm{prop}}$ and $\mathrm{Vf}_{\mathrm{vote}}$ remain the same as in the original paper [20]. Therefore, we focus on validity at origin for $\mathrm{Vf}_{\mathrm{comm}}$.

Fact A.3. *Suppose that in some epoch e by the end of the Vote phase, L_e remains in an honest node u's trust graph. Then, for any node v that satisfies $d(v,L_e,G_u) < d$ and $d(v,u,G_u) < d$ by the end of the Vote phase, v must have TrustCasted to u a vote message of the form $(vote,e,b)$ where b denotes the bit proposed by L_e.*

Proof. By Lemma A.1, we know that there is at least one honest user v that satisfies $d(v, L_e, G_u) < d$ and $d(v, u, G_u) < d$. Furthermore, by the property of the TrustCast protocol, for any v that satisfies $d(v, L_e, G_u) < d$ and $d(v, u, G_u) < d$ by the end of the Vote phase, v must have TrustCasted to u a vote (vote, e, b). The fact follows because u would check Vf_{vote} on every vote it receives, and Vf_{vote} makes sure that the vote is only accepted if the vote agrees with all other votes it receives. □

The validity at origin proof for the Vf_{comm} function then follows directly.

A.4 Proof of dishonest majority protocol's correctness

In order to prove the correctness of our dishonest majority protocol, it must satisfy the same consistency, validity, and liveness conditions that we proved in the honest majority protocol. We use similar proofs as the ones for the honest majority protocol.

Lemma A.4. *If two honest users u and v see commit evidences for b^* and b, respectively, in epoch e, it must be that $b = b^*$.*

Proof. By Lemma A.1, if the leader remains in their trust graphs, any honest user u will receive a vote on the leader's proposal from at least one other honest user, so u's commit evidence for (e, b^*) must contain a vote from at least one honest user w. As shown in [20], all honest users remain direct neighbors with each other in their trust graphs, and we showed in section A.3 that this holds true even with our modified Vf functions. Since v and w are both honest, they would be directly connected and v would have also received w's vote on b^* in epoch e. If $b \neq b^*$, then v would have detected the leader's equivocation and would not have outputted b. Therefore, we must have $b = b^*$. □

Lemma A.5. *If an honest user u outputs b in epoch e, then no honest user v can output a different bit $b' \neq b$ in any future epoch $e' > e$.*

Proof. After u obtains a commit evidence for (e, b), u will trustcast its commit evidence and all honest users will receive the commit evidence by the end of epoch e. By Lemma A.4, honest users can not output different bits in the same epoch, so there can not be a commit evidence for (e, b'). Therefore, at the start of epoch $e+1$, every honest user's freshest commit evidence is for b. During epoch $e+1$, every honest user would reject the leader's L_{e+1}'s proposal if it is not for the same bit b, because otherwise it would not be verified by Vf_{prop}. Therefore, no honest user will vote for a different bit $b' \neq b$, and so no honest user will be able to generate a fresher commit evidence for b' in epoch $e+1$. We can then user induction to show that this will continue to be true in all future epochs $e' > e$, and no honest user will ever see a commit evidence for a message $b' \neq b$. □

Theorem A.6 (Consistency)**.** *If two honest users output b and b' respectively, it must be that $b = b'$.*

Proof. Suppose honest user u is the first to output a bit and outputs b in epoch e. Due to Lemma A.4, honest users will not see a commit evidence for (b^*, e) in epoch e for any $b^* \neq b$. Furthermore, by Lemma A.5, no honest user will be able to form a commit evidence for a message $b' \neq b$ in all future epochs. Therefore, all honest users can only commit and output on b in future epochs. □

Theorem A.7 (Validity and Liveness)**.** *If the leader L_e in epoch e is honest, then every honest user will terminate on the leader's proposed bit in that epoch.*

Proof. Since the leader remains in all honest user u's trust graph, we can use Fact A.3 to show that u will receive votes on the leader's proposal from every node v that satisfies $d(v, L_e, G_u) < d$ and $d(v, u, G_u) < d$ by the end of the vote phase, so all honest users u will output the leader's proposal. In the commit phase, since we are using the TrustCast protocol for d rounds, we can apply Theorem 4.1 of [20] to show that u will receive commit evidences on the leader's proposal from everyone still in their trust graphs by the end of the commit phase. Therefore, all honest users u will terminate by the end of epoch e. □

Acknowledgement

I would like to thank the MIT PRIMES program for the research opportunity and my mentor, Jun Wan, for guiding me throughout the research process.

References

[1] Ittai Abraham et al. "Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus". In: *CoRR* abs/1612.02916 (2016).

[2] Ittai Abraham et al. "Synchronous Byzantine Agreement with Optimal Resilience, Expected O(n^2) Communication, and Expected O(1) Rounds". In: *Financial Cryptography and Data Security (FC)*. 2019.

[3] Atul Adya et al. "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment". In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 1–14.

[4] Christian Cachin, Klaus Kursawe, and Victor Shoup. "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography". In: *Journal of Cryptology* 18.3 (2005), pp. 219–246.

[5] Miguel Castro and Barbara Liskov. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[6] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. "Sublinear-Round Byzantine Agreement under Corrupt Majority". In: *PKC*. 2020.

[7] Jing Chen and Silvio Micali. *ALGORAND: The Efficient and Democratic Ledger*. 2016. URL: `https://arxiv.org/abs/1607.01341`.

[8] Danny Dolev and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement". In: *Siam Journal on Computing - SIAMCOMP* 12.4 (1983), pp. 656–666.

[9] Ryan Farell. "An analysis of the cryptocurrency industry". In: (2015).

[10] Paul Feldman and Silvio Micali. "Byzantine agreement in constant expected time". In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (1985), pp. 267–276.

[11] Pesech Feldman and Silvio Micali. "An optimal probabilistic protocol for synchronous Byzantine agreement". In: *SIAM Journal of Computing*. 1997.

[12] Matthias Fitzi and Jesper Buus Nielsen. "On the number of synchronous rounds sufficient for authenticated Byzantine agreement". In: *International Symposium on Distributed Computing*. Springer. 2009, pp. 449–463.

[13] Juan A Garay et al. "Round complexity of authenticated broadcast with a dishonest majority". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE. 2007, pp. 658–668.

[14] Juan A. Garay et al. "Adaptively Secure Broadcast, Revisited". In: (2011). URL: `https://ssltest.cs.umd.edu/~jkatz/papers/asb-final.pdf`.

[15] Yossi Gilad et al. "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM. 2017, pp. 51–68.

[16] Jonathan Katz and Chiu-Yuen Koo. "On expected constant-round protocols for Byzantine agreement". In: *Annual International Cryptology Conference*. Springer. 2006, pp. 445–462.

[17] Leslie Lamport. "The part-time parliament". In: *ACM Transactions on Computer Systems (TOCS)* 16.2 (1998), pp. 133–169.

[18] Julian Loss and Tal Moran. "Combining asynchronous and synchronous Byzantine agreement: The best of both worlds". In: (2018).

[19] Fred B. Schneider. "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial". In: *ACM Computing Surveys* 22.4 (1990).

[20] Jun Wan et al. "Expected Constant Round Byzantine Broadcast under Dishonest Majority". In: (2020). URL: `https://eprint.iacr.org/2020/590.pdf`.