On the Amortized Communication Complexity of Byzantine Broadcast

ATSUKI MOMOSE[*], LING REN[†], ELAINE SHI[‡], JUN WAN[§], and ZHUOLUN XIANG[¶]

Designing an efficient solution for Byzantine broadcast is an important problem for many distributed computing and cryptographic tasks. There have been many attempts to achieve sub-quadratic communication complexity in several directions, both in theory and practice, all with pros and cons. This paper initiates the study of another attempt: improving the *amortized* communication complexity of multi-shot Byzantine broadcast. Namely, we try to improve the average cost when we have sequential multiple broadcast instances. We present a protocol that achieves optimal amortized linear complexity under an honest majority. Our core technique is to efficiently form a network for disseminating the sender's message by keeping track of dishonest behaviors over multiple instances. We also generalize the technique for the dishonest majority to achieve amortized quadratic communication complexity.

1 INTRODUCTION

Byzantine broadcast (BB) is one of the fundamental problems in distributed systems. In the problem of Byzantine broadcast, there are n nodes, one of which is a designated sender. The sender is given a message m and wants to share it with the rest of the nodes. Despite up to f nodes being corrupt, all honest nodes agree on the sender's message.

An important metric of a Byzantine broadcast protocol is its communication complexity, which measures the number of bits sent by non-corrupt parties during the protocol execution. Dolev and Reischuk [9] showed any deterministic Byzantine broadcast protocol costs at least $\Omega(f^2)$ messages in the worst case, which was matched by Berman et al. [3] for $f < n/3$ and then by Momose and Ren [22] for $f < n/2$. However, for large-scale distributed systems, $\Theta(n^2)$ communication complexity may still be expensive. Many attempts have been made to circumvent the impossibility results, mainly in three directions – (1) Randomized solutions [2, 4, 7, 18], which improves the communication cost to (worst-cast or expected) sub-quadratic. But this approach is inherently vulnerable to *strongly adaptive adversaries* [2]. (2) Optimistic solutions [13, 19, 21, 26, 28, 32] that have sub-quadratic communication under optimistic executions such as synchrony and no failures. But they still incur quadratic costs in the worst case. (3) Extension protocols [11, 12, 24] can achieve the optimal communication cost $O(nL)$ for an input of sufficient size L. Thus they can also reduce the cost of multiple parallel broadcasts (or atomic broadcasts [8]) through batching. But they do not allow sequential and causal broadcast invocations (i.e., a decision in an instance may affect the input of the next instance), which is required in many cryptographic protocols assuming *broadcast channel* [14, 25, 31]. We refer the reader to Section 2 for more discussions and comparisons.

Despite all the efforts in the above three directions, another natural attempt through amortization across multiple (sequential) instances, is somehow overlooked in the literature. In this paper, we initialize the formal study of the amortized communication complexity of multi-shot Byzantine broadcast. Multi-shot BB consists of sequential broadcasts with clear boundaries (i.e., one instance ends before the next instance starts) performed by possibly different senders. The amortized communication cost of a multi-shot Byzantine broadcast protocol is measured as the average cost of each broadcast instance if the protocol runs sufficiently long. Due to the ever-growing nature of targeted applications such as blockchains, reducing the amortized cost of the multi-shot Byzantine

[*]atsuki.momose@gmail.com, University of Illinois at Urbana-Champaign
[†]renling@illinois.edu, University of Illinois at Urbana-Champaign
[‡]runting@gmail.com, CMU
[§]Lead author. junwan@mit.edu, Massachusetts Institute of Technology.
[¶]xiangzhuolun@gmail.com, Aptos

broadcast can significantly improve the performance of a long-running system. More formally, suppose the total communication complexity in bits of the multi-shot protocol is $C(L, n, f)$ after L sequential instances of Byzantine broadcast, our goal is to design multi-shot Byzantine broadcast protocols that can minimize

$$\lim_{L\to\infty} \frac{C(L, n, f)}{L}.$$

Interestingly, we show that we can design amortized linear multi-shot Byzantine broadcast protocols under synchrony with the honest majority and strongly adaptive adversaries, circumventing the $\Omega(f^2)$ lower bound. We also extend our technique to the dishonest majority case to achieve quadratic amortized cost. Our results are the following (summarized in Table 1).

- Assuming a threshold signature scheme, we show how to achieve $O(\kappa n)$ (κ is a security parameter) amortized communication cost for synchronous multi-shot Byzantine broadcast under $f \le (1/2 - \varepsilon)n$ for any positive constant ε (Section 4).
- Assuming a digital signature scheme, we show how to achieve $O(\kappa n^2)$ amortized communication cost for synchronous multi-shot Byzantine broadcast under $f < n$ (Section 5).

Protocol	Network Model	Fault Tolerance	Total Cost (L decisions)	Amortized Cost	Cryptographic Primitives
Berman et al. [3]	synchrony	$f < n/3$	$O(n^2L)$	$O(n^2)$	None
Momose-Ren [22]	synchrony	$f < n/2$	$O(\kappa n^2L)$	$O(\kappa n^2)$	threshold sig.
Momose-Ren [22]	synchrony	$f \le (1/2-\epsilon)n$	$O(\kappa n^2L)$	$O(\kappa n^2)$	signature
This work	synchrony	$f \le (1/2-\epsilon)n$	$O(\kappa nL + \kappa n^3)$	$O(\kappa n)$	threshold sig.
Dolev-Strong [10]	synchrony	$f < n$	$O((\kappa n^2 + n^3)L)$	$O(\kappa n^2 + n^3)$	multi-sig *
This work	synchrony	$f < n$	$O(\kappa n^2L + \kappa n^4)$	$O(\kappa n^2)$	signature

Table 1. Comparison with existing solutions for multi-shot BB with constant-sized inputs under strongly adaptive adversaries. *The original Dolev-Strong broadcast uses signature and has cost $O(\kappa n^3)$.

1.1 Technical challenge

At a high level, Byzantine broadcast is commonly implemented with repeated invocations of *consistent broadcast* [6], which guarantees nodes' outputs are consistent, i.e., nobody outputs different values. There are known solutions to achieve consistent broadcast with linear communication [22, 32], so achieving safety with a linear cost is not hard. However, note that consistent broadcast does not provide *totality.* In other words, it is allowed that some nodes output but others do not when the sender is dishonest. Therefore, to achieve liveness, we have to disseminate the sender's message to everybody. The common step to handle this issue is each node forwards the sender's message once received. But this always costs quadratic messages. The key technical contribution of this work is to adaptively form an efficient data dissemination network between honest nodes that eventually converges with bounded cost, which is amortized over multiple instances.

2 RELATED WORK

Communication bound of Byzantine broadcast. Dolev and Reischuk [9] showed any deterministic Byzantine broadcast protocol requires at least $\Omega(f^2)$ messages in the worst case even against static adversaries. This was later extended by Abraham et al. [2], who showed that even the expected communication of randomized protocols is subject to the quadratic bound under *strongly adaptive adversaries* who can corrupt a node and retract the messages sent by that node in the same round.

The Dolev-Strong protocol [10] costs quadratic messages but its communication complexity in bits (for a constant size input) is $\Omega(\kappa n^2 + n^3)$ (assuming multi-signatures). These lower bounds on communication were first matched by Berman et al. [3] who showed a synchronous protocol with $O(n^2)$ communication and $f < n/3$. Recently, Momose and Ren [22] extended it to protocols with $O(\kappa n^2)$ communication with 1) $f < n/2$ assuming threshold signatures, or 2) $f \leq (1/2 - \varepsilon)n$ for any positive constant ε assuming digital signatures.

Circumventing the quadratic bound. There have been significant efforts to circumvent the quadratic lower bound in several directions, all with pros and cons. Below, we highlight three main directions to compare with our approach.

One common direction is to achieve (worst-case or expected) sub-quadratic communication through randomization. There are several solutions for both (partial) synchrony [2, 15, 18] and asynchrony [4, 7]. Roughly speaking, these solutions usually sample a random committee of a small size to perform consensus within the committee and then inform the rest of the nodes. However, due to the aforementioned impossibility result [2], this approach is inherently vulnerable to strongly adaptive adversaries, who can perform *after-the-fact* message removal, i.e., rushingly corrupt nodes (e.g., nodes selected in the committee) and erase messages already sent by the nodes before reaching the recipients.

Another approach is to have an optimistic execution path that costs sub-quadratic communication under good conditions such as synchrony and no faults [13, 19, 21, 26, 28]. But they always have another expensive path that costs quadratic communication in the worst case. We note here HotStuff [32] can also be categorized into this type of solution. More specifically, HotStuff can also solve a synchronous multi-shot BB with amortized linear communication in failure-free cases. However, we have to prepare a fallback path for dishonest senders (not fully specified in [32]) due to the aforementioned message dissemination problem (Section 1.1). In Appendix A, we explain why/how HotStuff fails to achieve liveness without a fallback path.

Finally, extension protocols achieve optimal $O(Ln)$ communication if input size L is sufficiently large [11, 12, 24]. This is another orthogonal approach to reduce the amortized communication cost of multiple parallel instances of broadcast. Namely, we can batch the inputs of multiple parallel instances together and solve them once. However, in order to achieve optimal cost, the extension protocol needs to wait for inputs of sufficient size (e.g. $L = \Omega(\kappa n)$ for honest majority) and then batch the inputs to start the protocol, which can introduce additional latency. Moreover, such a solution does not support sequential causal inputs of broadcasts, which is important in certain applications [31].

Byzantine atomic broadcast. With synchrony, multi-shot Byzantine broadcast can directly solve *Byzantine atomic broadcast* [8, 27] that commits values at increasing slots (not vice versa, as an atomic broadcast does not have clear boundaries). State-of-the-art synchronous Byzantine atomic broadcast protocols [1, 17] cost quadratic communication per decision. Our protocol also solves Byzantine atomic broadcast with linear communication complexity.

Trust graph in Byzantine broadcast/agreement. There have been several works that use *trust graphs.* Liang and Vaidya [20] use a trust graph to design an extension protocol for the Byzantine agreement with $f < n/3$. The amortized cost of their protocol is $\Omega(n^4)$. Recently, Wan et al. [30] improved the expected round complexity of single-shot Byzantine broadcast using a trust graph. Their key observation is that, if properly maintained, the diameter of the trust graph at any honest node can be constant. Although orthogonal in goal, their protocol inspired us to use the trust graph for keeping track of misbehaviors across different instances.

3 PRELIMINARY

Model and assumptions. We consider a system of n nodes (numbered 1 to n) in the lock-step synchronous model; all nodes have synchronous clocks that start at the same time from round $t = 0$ and increase at the same speed, and any message sent by an honest node in round t will be delivered to the recipient by the beginning of round $t + 1$. We assume f out of n nodes are corrupt (Byzantine) and behave arbitrarily. We assume *adaptive corruption*, i.e., corruption happens anytime during the execution. We also assume the adversary is *strongly adaptive* [2] who can perform *after-the-fact* message removal, i.e., an adversary can decide the newly corrupted nodes in round t after seeing the messages sent by nodes in round t and erase the messages. Any node that remains non-faulty during the entire execution is referred to as *honest.*

We assume a digital signature scheme with a public-key infrastructure (PKI). A message m signed by node i is denoted $\langle m\rangle_i$. Our protocol in Section 4 assumes a threshold signature scheme [5]. In a (t, n)-threshold signature scheme, each node i can generate a signature share $\langle m\rangle_i$ on a message m. A set of t distinct signature shares $\{\langle m\rangle_{j_1}, .., \langle m\rangle_{j_t}\}$ on the same message m can be combined into a full signature $\mathrm{thsig}(m)$, which has the same length as a single signature share $\langle m\rangle_*$. An adversary cannot generate the full signature $\mathrm{thsig}(m)$ from less than t signature shares. The threshold signature scheme can be set up either through a trusted dealer, or distributed key generation [16]. Our protocol uses the threshold $t = n - f$.

Problem definition. We are interested in the problem of *multi-shot Byzantine broadcast*, which consists of a sequence of single-shot *Byzantine broadcasts* [10]. We first review the definition of the single-shot BB.

Definition 1 (Byzantine Broadcast (BB)). *A Byzantine broadcast protocol for a set of n nodes with a designated sender with input value v invoking* $\mathrm{bc}(v)$, *must satisfy the following properties.*

- *Consistency. If two honest nodes commit values v and v' respectively, then $v = v'$.*
- *Termination. All honest nodes commit and terminate.*
- *Validity. If the designated sender is honest and invokes* $\mathrm{bc}(v)$, *then all honest nodes commit v and terminate.*

We now define *multi-shot Byzantine braodcast* below, which repeatedly invokes single-shot BB.

Definition 2 (Multi-shot Byzantine Broadcast). *A multi-shot Byzantine broadcast protocol for a set of n nodes with a (possibly different) designated sender S_i for each slot $i > 0$ with input value v invoking* $\mathrm{bc}_k(v)$, *must satisfy the following properties.*

- *Consistency. If two honest nodes commit values v and v' respectively at the same slot, then $v = v'$.*
- *Termination. All honest nodes eventually commit a value at any slot.*
- *Validity. If the designated sender of slot i is honest and invokes* $\mathrm{bc}_i(v)$, *then all honest nodes commit v at slot i.*
- *Sequentiality. For any slot i, the sender S_i is allowed to invoke* bc_i *after* bc_j *is committed at all honest nodes for all slots $j < i$.*

Note that the multi-shot BB has clear boundaries between each slot due to the *sequentiality.* In contrast to atomic broadcast [8], it supports causal inputs. Namely, the sender of each slot can decide its input depending on the previous decisions. As mentioned, using extension protocols with batching cannot solve the problem with linear cost, since $O(n)$ many slots' senders must input in parallel.

Metrics. First of all, *communication complexity* is measured as the bit amount sent by honest nodes. In this paper, we are interested in the *amortized communication complexity* of multi-shot Byzantine broadcast, defined as follows.

Definition 3 (Amortized Communication Complexity). *Let $C(L, n, f)$ be the communication complexity of a multi-shot Byzantine broadcast protocol for n nodes with f faults to commit L slots. The amortized communication complexity of the protocol is defined to be* $\lim_{L\to\infty} \frac{C(L,n,f)}{L}$.

For example, a multi-shot BB protocol that naively runs multiple instances of single-shot BB of cost C_{BB}, will have an amortized communication cost of C_{BB}. Since the current state-of-the-art BB protocol for the honest majority is the Momose-Ren Byzantine broadcast of cost $O(\kappa n^2)$, such a naive approach only gives us an amortized $O(\kappa n^2)$ protocol, which has an $O(n)$ gap from the optimal.

Expander graph. Our protocol in Section 4 uses an expander (inspired by [22]), which is a graph with sparse edges but overall good connectivity. More formally, an (n, α, β)-expander ($0 < \alpha < \beta < 1$) is a graph of n vertices s.t. for any set S of αn vertices, the number of neighbors of S is more than βn. It is well-known that for any n and any constants α, β such that $0 < \alpha < \beta < 1$, an expander with constant degree exists [22].

4 AMORTIZED LINEAR COMMUNICATION UNDER HONEST MAJORITY

This section shows how to achieve $O(\kappa n)$ amortized communication complexity with $f \leq (1/2 - \varepsilon)$ for any positive constant ε.

Our protocol is described in Algorithm 4. At a high level, our multi-shot BB protocol consists of multiple slots, where each slot implements a single-shot BB. Each slot progresses through many *epochs*, with each epoch having a unique *leader*. A leader proposes a value, other nodes vote for it, and nodes commit it after collecting enough votes. Before explaining the detail, we first define some notions and notations used in our protocol.

Definition and notations. For each slot $k \geq 1$, we have $f + 2$ epochs $0 \leq i \leq f + 1$ each takes 11 rounds. So epoch i of slot k starts in round $t = 11((k - 1)(f + 2) + i)$. The leader L_0 of the first epoch $i = 0$ is the sender S_k, and the leader L_i of epoch $1 \leq i \leq f + 1$ is node i.

To reduce message size, we use a threshold signature scheme to combine a set of votes into a *certificate*. A certificate for a value m in epoch i of slot k, denoted $C_{k,i}(m)$, is $\mathsf{thsig}(\mathsf{vote}, k, i, m)$ of an $(n - f, n)$-threshold signature scheme, i.e., aggregated votes from a quorum of $n - f$ nodes. For a technical reason, we also consider $\perp$ as a certificate for any slot k and value m. We define *freshness* of certificates of the *same slot* by epoch: the higher the epoch, the fresher the certificate (e.g., $C_{k,1}(m)$ is fresher than $C_{k,0}$). Also, any certificate is fresher than $\perp$.

We say a leader L_i *equivocates*, if there are two different proposals in the same epoch and slot, i.e., $\langle \mathsf{prop}, k, i, m, C\rangle_{L_i}$ and $\langle \mathsf{prop}, k, i, m', C\rangle_{L_i}$ for $m \neq m'$.

Common path. We now explain the protocol in more detail. The first 7 rounds of each epoch perform the propose-then-vote operation. The leader proposes a value m (round Propose), and other nodes vote for it twice. They first vote for the value m (round Vote), then vote for the certificate $C_{k,i}(m)$ (round Propagate-2). Nodes commit m after receiving a *commit-proof* $\mathsf{thsig}(C_{k,i}(m))$. To make the cost linear, we use two known techniques.

First, we use the leader as a "message hub" [32]. Namely, nodes send signed votes (threshold signature shares) only to the leader, and the leader, after collecting a quorum of $n - f$ votes, aggregates them into a κ-size certificate (or a commit-proof) and sends it to nodes. This allows nodes to make progress with linear costs under an honest leader.

Second, we use an expander graph to prevent the formation of certificates on two different values (which would lead to disagreement) [22]. More specifically, we use an $(n, 2\varepsilon, 1-2\varepsilon)$-expander with each vertex representing each node. Each node, after receiving a proposal from the leader, sends it to its neighbors in the expander (round Propagate-1) before voting. If a certificate $C_{i,k}(m)$ exists, $n - f \geq f + 2\varepsilon n$ nodes, out of which at least $2\varepsilon n$ honest nodes, must have sent the proposal of m to their neighbors. The expansion property implies more than $(1-2\varepsilon)n \geq 2f$, out of which at least $f+1$ honest nodes would receive the proposal. They would never vote for $m' \neq m$, so $C_{i,k}(m')$ cannot exist.

Achieving liveness with dishonest leaders. So far, we have explained how to commit safely with linear communication when the leader is honest. But if the leader is dishonest, a commit-proof may not be formed. In that case, to ensure liveness, nodes accuse the leader by sending accuse messages to all nodes (round Query-1). If the leader is completely silent, at least $n - f$ nodes accuse the leader which forms a *corrupt-proof* of the leader. The corrupt proof will be forwarded to everybody (after aggregation), and honest nodes will simply ignore this leader ever after. But the situation will be more complex if the dishonest leader sends messages selectively. Some nodes may receive the commit-proof, but some may not. Since not everybody accuses the leader, we do not have a corrupt-proof; but not everybody receives a commit-proof, either. The last four rounds (rounds 8–11) resolve this issue. Roughly, we try to disseminate the commit-proof in two steps. First, the node u missing the commit-proof queries one node v selected deterministically (round Query-1), who responds with the commit-proof (round Repond-1). If v does not help, then node u accuses v and queries all nodes, some of whom have a commit-proof (round Query-2, Respond-2). The high-level idea is, though quadratic communication may be incurred in this latter query-all step, the number of such occurrences is bounded. An honest node u selects its helper from nodes that it has not accused (Query-1). So each honest node will eventually find an honest helper, after which they can receive a commit-proof from the helper alone. A dishonest node u may try to keep invoking the query-all step. But honest nodes respond only when u accuses a new node (Respond-2). Thus, the dishonest node u will eventually run out of new nodes to accuse, after which honest nodes will no longer respond to u.

Algorithm 4: Linear communication multi-shot BB

Each epoch $0 \leq i \leq f+1$ of slot $k \geq 1$ takes 11 rounds. Let L_i be the leader of epoch i, and G_ε be an $(n, 2\varepsilon, 1-2\varepsilon)$-expander that is known to all nodes. Each node u runs the following steps if it has neither 1) committed in slot k, nor 2) received the corrupt-proof $\text{thsig}(\text{accuse}, L_i)$.

```
// Leader proposes a value
    (1) Collect: Send the freshest slot-$k$ certificate to $L_i$.
    (2) Propose: If $u = L_i$, multicast $\langle \text{prop}, k, i, m, C \rangle_{L_i}$ where:
        (a) If $u$ has received a slot-$k$ certificates ($\neq \perp$), then $C$ is the freshest $C_{k,j}(m)$ among
            those ever received.
        (b) Otherwise, $C = \perp$, and $m \leftarrow \text{bc}_k$ (if $i = 0$) or an arbitrary value (if $i > 0$).

// Vote for a leader's value
    (3) Propagate-1: If $u$ receives a proposal $\langle \text{prop}, k, i, m, C_{k,j}(m) \rangle_{L_i}$ s.t. $C_{k,j}(m)$ is a certificate
        as fresh as what $u$ sent to $L_i$ during Collect, then send the proposal to its neighbors in
        the expander $G_\varepsilon$.
    (4) Vote: If $u$ has detected equivocation of $L_i$, $u$ multicast $\langle \text{accuse}, L_i \rangle_u$ (if not yet sent).
        Else if $u$ has sent $L_i$'s proposal on $m$ in Propagate-1, send $\langle \text{vote}, k, i, m \rangle_u$ to $L_i$
```

// Vote for a certificate

(5) **Certificate**: If $u = L_i$ and it receives $n - f$ $\langle \text{vote}, k, i, m\rangle_*$, aggregate them to generate $C_{k,i}(m) \leftarrow \mathsf{thsig}(\text{vote}, k, i, m)$ and multicast it.

(6) **Propagate-2**: If u receives $C_{k,i}(m)$, send it to its neighbors in G_ε, and send $\langle C_{k,i}(m)\rangle_u$ to the leader L_i.

(7) **Commit**: If $u = L_i$ and it receives $n - f$ $\langle C_{k,i}(m)\rangle_*$, aggregate them to generate a *commit-proof* $\mathsf{thsig}(C_{k,i}(m))$ and multicast it.

// Disseminate a commit-proof

(8) **Query-1**: If u has not received any commit-proof of epoch i, multicast $\langle \text{accuse}, L_i\rangle_u$ (if not yet sent), and send $\langle \text{query}_1, k, i\rangle_u$ to the smallest node v s.t. 1) u has not accused v and 2) v has not accused L_i.

(9) **Respond-1**: If u has received $\langle \text{query}_1, k, i\rangle_v$ and it has a commit-proof $\mathsf{thsig}(C_{k,i}(m))$, send $\mathsf{thsig}(C_{k,i}(m))$ to v if 1) v has accused L_i and 2) u is the smallest node v has not accused.

(10) **Query-2**: If u sent a query_1 message to node v in **Query-1** and has not received a commit-proof from v, then multicast $\langle \text{accuse}, v\rangle_u$ and $\langle \text{query}_2, k, i\rangle_u$.

(11) **Respond-2**: If u has received $\langle \text{accuse}, w\rangle_v$ and $\langle \text{query}_2, k, i\rangle_v$ and it has a commit-proof $\mathsf{thsig}(C_{k,i}(m))$, send $\mathsf{thsig}(C_{k,i}(m))$ to v, if this is the first time u receives $\langle \text{accuse}, w\rangle_v$.

At any point of the protocol:

($\star$) Upon receiving a commit-proof $\mathsf{thsig}(C_{k,j}(m))$, commit m at slot k.

($\star$) Upon receiving $\langle \text{accuse}, v\rangle_w$, forward it to the accused node v.

($\star$) Upon receiving $n - f$ $\langle \text{accuse}, v\rangle_*$ for any v, aggregate them into a corrupt-proof $\mathsf{thsig}(\text{accuse}, v)$ and multicast it.

($\star$) Upon receiving a commit-proof $\mathsf{thsig}(C_{k,j}(m))$ for any j, if u has received $n - f$ $\langle \text{accuse}, L_j\rangle_*$, then multicast the commit-proof.

4.1 Proof of correctness

We first show that the protocol satisfies *consistency* using the following two lemmas. Lemma 1 implies that nodes cannot commit different messages within the same epoch. Lemma 2 implies that nodes cannot commit differently even across different epochs.

Lemma 1. *If certificates $C_{k,i}(m)$ and $C_{k,i}(m')$ both exist, then $m = m'$.*

Proof. Suppose $C_{k,i}(m)$ exists, then at least $n - f \geq f + 2\varepsilon n$ must have sent $\langle \text{vote}, k, i, m\rangle_*$, out of which at least $2\varepsilon n$ honest nodes must have forwarded the leader's proposal to the neighbors in the expander graph (in round Propagate-1). Due to the expansion property, more than $(1 - 2\varepsilon)n \geq 2f$ nodes, out of which at least $f + 1$ honest nodes would receive the proposal, who would never vote for $m' \neq m$. Thus, $C_{k,i}(m')$ cannot exist. □

Lemma 2. *If there exists a commit-proof $\mathsf{thsig}(C_{k,i}(m))$, then for all epochs $j > i$, there cannot exist a certificate $C_{k,j}(m')$ on $m' \neq m$.*

Proof. The commit-proof requires a quorum of $n - f$ nodes' signatures on $C_{k,i}(m)$. Therefore, at least $n - 2f \geq 2\varepsilon n$ honest nodes must have received $C_{k,i}(m)$ in epoch i of slot k. After they propagate it in round Propagate-2, more than $(1 - 2\varepsilon)n \geq 2f$ nodes (out of which at least $f + 1$ are honest) must have received $C_{k,i}(m)$. The $f + 1$ honest nodes will send $C_{k,i}(m)$ to the next leader L_{i+1} during Leader setup, so they will never vote for a proposal in the epoch $i + 1$ unless it contains an epoch i certificate. As a certificate $C_{k,i}(m')$ cannot exist for $m' \neq m$ (by Lemma 1), they will never vote for m' in epoch $i+1$ of slot k, which implies $C_{k,i+1}(m')$ cannot exists. Inductively, for any $j > i$, a certificate $C_{k,j}(m')$ cannot exist. □

With the above lemmas, we can now show that our protocol satisfies *consistency.*

Theorem 1. *The protocol satisfies consistency. If any two honest nodes u and v commit m_u and m_v respectively at the slot k, it must be that $m_u = m_v$*

Proof. An honest node outputs if it has observed a commit-proof. Suppose u observes a commit-proof on m_u of epoch i and v observes a commit-proof on m_v of epoch j. We can assume w.l.o.g. that $i \leq j$.

- If $i = j$, by Lemma 1, there cannot exist commit-proof on different messages in epoch i. Therefore, $m_u = m_v$.
- If $i < j$, by Lemma 2, if a commit-proof on m_u exists in epoch i, then any future commit-proof must also be on m_u. Therefore, $m_u = m_v$.

This completes our consistency proof. □

Finally, to prove the remaining properties, we prove the following lemma that shows an honest leader's epoch is always successful.

Lemma 3. *If the leader L_i is honest, all honest nodes commit by the end of epoch i in each slot.*

Proof. Consider slot $k = 1$. In epochs before i, since L_i is not the leader, the only case where L_i gets accused by an honest node u is when u has sent query_1 to L_i but it has not sent back a commit-proof to u during round Respond-1. However, if L_i does not have a commit-proof to send back to u, then L_i must have also accused the epoch's leader, and u would not have sent query_1 to L_i. So, honest nodes do not accuse L_i before epoch i. Now, in epoch i, since $n - f$ accusations cannot exist for L_i, all honest nodes (of at least $n - f$) vote for the leader's proposal (the leader's proposal is always accepted by honest nodes since it contains the freshest certificate among those honest nodes sent during the round Collect, forming a certificate and then a commit-proof. So honest nodes commit and do not accuse L_i. Therefore, in the next slot (and inductively in all later slots), all honest nodes commit by the end of epoch i without being accused by honest nodes. □

The lemma above trivially implies *validity* as the first leader L_0 is the sender. Also, since we have at most f dishonest leaders, we will have an honest leader by epoch $f+1$. So, by the end of each slot, all honest nodes commit a value for the slot. This implies our protocol satisfies *termination* and *sequentiality.*

4.2 Communication complexity

Let us first consider the cost of *expensive slots*: a slot with more than one epoch with non-zero communication. In an expensive slot, the leaders of all epochs except the last one must be accused by all honest nodes, forming the corrupt-proofs (i.e., $n - f$ accusations) for these leaders; Otherwise, at least an honest node who does not accuse the leader must have received a commit-proof before round Query-1 which is forwarded to all honest nodes (in either round Respond-1 or 2) and all honest nodes would stop sending messages in all later epochs. So, the number of epochs across all expensive slots is $O(n)$, and the total cost across all expensive slots is $O(\kappa n^3)$ as each epoch costs at most $O(\kappa n^2)$ communication.

Next, for non-expensive slots (i.e., with only one epoch), we consider *expensive epochs*: an epoch with super-linear communication. Obviously, the first 7 rounds cost $O(\kappa n)$ per epoch. The cost of forwarding a commit-proof will be super-linear communication only when a corrupt-proof is formed; such epochs exist at most f times, hence totally costs $O(\kappa n^3)$ across all expensive epochs. Finally, the cost of round 8–11 is analyzed below:

(1) Query-1: Each honest node sends only one message; per-epoch cost is linear.
(2) Respond-1: Each honest node can receive a response from one node, which costs linear per epoch. Suppose in an epoch, multiple honest nodes R respond to a malicious node v. In this case, v has accused all nodes in R except the highest one, which are forwarded to the accused nodes, who will stop responding in all later epochs/slots. Therefore, for each malicious node v, there are at most n epochs s.t. multiple honest nodes respond to v; totally costs $O(\kappa n^3)$ across all expensive epochs.
(3) Query-2: Each honest node sends query_2 when it accuses a new node (i.e., the helper), which can happen at most f times; totally costs $O(\kappa n^3)$ across all expensive epochs.
(4) Respond-2: An honest node u responds to a node v only if v accuses a new node, which can happen at most n times; totally costs $O(\kappa n^3)$ across all expensive epochs.

To sum up, the total cost is $O(\kappa nL + \kappa n^3)$ for L slots.

5 AMORTIZED QUADRATIC COMMUNICATION UNDER DISHONEST MAJORITY

In this section, we generalize the idea used in Section 4 to show how to achieve amortized $O(\kappa n^2)$ communication complexity for the dishonest majority case, i.e., $f < n$.

Overview. The state-of-the-art for the dishonest majority case in terms of communication complexity is the Dolev-Strong protocol, which costs cubic communication ($O(\kappa n^2 + n^3)$ with multi-signature or $O(\kappa n^3)$ with signature). Our first natural idea is, instead of directly agreeing on the sender's value, we use the Dolev-Strong protocol to agree on the sender's dishonesty when the sender misbehaves. Every time we call the Dolev-Strong protocol, at least one node will be proved corrupt and removed from the protocol. This way, the Dolev-Strong protocol is called at most $f + 1$ times across all instances.

Now, to agree on a dishonest sender, we have to provably detect the sender's misbehavior. For the honest majority case, more than f accusations from honest nodes work as a corrupt-proof. But it does not work directly for the dishonest majority case. We instead utilize the *TrustCast* protocol introduced by Wan et al. [30]. Looking ahead, our technique used in Section 4 can also be explained generally in the same context. Before describing our protocol, we briefly review the TrustCast protocol and what they provide below.

5.1 TrustCast

We describe a simplified TrustCast protocol [30] in Algorithm 5.1 (as our protocol does not require the constant-size diameter property of the trust graph as in [30]). It allows a designated sender to multicast a message while allowing other nodes to provably detect the sender's misbehavior when they do not receive the message from the sender. Nodes detect a dishonest sender by carefully observing trust relationships between nodes. More specifically, each node locally maintains a *trust graph*, a graph of n vertices with edges representing the trust relationships between nodes. The edges are updated based on nodes' accusations, and a dishonest sender is detected when it is removed from the graph, i.e., losing trust by everybody.

Algorithm 5.1: TrustCast(G_u, T, k).

Inputs: G_u is an undirected graph of numbered n vertices, T and k are integers representing the length of the protocol and the slot number, respectively.

In round 0, the sender S who has a message m multicasts $\langle \text{prop}, m, k \rangle_S$. Each node u runs all of the following steps in each round $1 \le t \le T$.

- If u receives $\langle \text{prop}, m, k \rangle_S$ for the first time, multicast it.
- If u receives $\langle \text{accuse}, v \rangle_w$ for any two vertices $v, w \in G_u$, remove the edge (v, w) from G_u, and multicast $\langle \text{accuse}, v \rangle_w$ (if not yet sent).
- If u has not received any $\langle \text{prop}, *, k \rangle_S$, then for any vertex $v \in G_u$ s.t. the distance between v and S is less than t, multi-casts $\langle \text{accuse}, v \rangle_u$ (if not yet sent).
- Remove all vertices in G_u unconnected with vertex u (i.e., nodes with no direct/indirect path from u).
- If u receives $\langle \text{prop}, m, k \rangle_S$ and $\langle \text{prop}, m', k \rangle_S$ for $m \neq m'$, then multicast them, and remove S from G_u.

Suppose G_u is a complete graph (i.e., every pair of vertices has an edge), and $T \ge n$. The protocol provides the following guarantees.

(1) *Transferability.* For any honest nodes u, v and any round t, G_u in round $t + 1$ is a subgraph of G_v in round t.
(2) *Termination.* By the beginning of round n, any honest node u either receives a sender's message or removes the sender from G_u.
(3) *Integrity.* For any honest nodes u, v, honest nodes never removes the edge between u and v from their trust graphs.

First of all, transferability is obvious since honest nodes forward all accusations received. Second, if any honest node u has not received any $\langle \text{prop}, *, k \rangle_S$ at the beginning of round t, u accuses any node with a distance of less than t from S in round t. So u's distance from S at the beginning of round $t + 1$ should be at least $t + 1$. By induction, we can show that honest nodes never accuse each other (integrity). Finally, termination holds since the diameter of the trust graph is at most $n - 1$ [*](by definition). If an honest node does not receive the sender's message by the beginning of round n, then the node's distance from S should be at least n, which means S is unconnected.

Communication complexity: For L instances of TrustCast with each node maintaining the trust graph across instances, the total communication cost is $O(\kappa n^2 L + \kappa n^4)$. In each instance, an honest node multicasts the sender's messages at most twice which costs $O(\kappa n^2)$. The cost of maintaining the trust graph is bounded by $O(\kappa n^4)$ across all instances, since for each edge in the trust graph, honest nodes multicast the accuse message at most once.

Our technique in Section 4. Interestingly, the technique we used for Algorithm 4 can be explained generally in the context of the TrustCast operation above. Recall that in Algorithm 4, a node u accuses its helper node v if v does not respond with a commit-proof. The rationale was since v has not accused the leader, it must have received a commit-proof from the leader. This is exactly what a node does in round $t = 2$ in TrustCast. Since a node v who has a distance $1 < t$ from the sender must have received the message in round 1, if u has not received the sender's message (which means v has not forwarded the sender's message), u knows v is malicious and accuses v. Essentially the TrustCast does this operation repeatedly and inductively.

[*]The diameter of the trust graph is actually more tightly bounded [30], but we use a loose bound since it is sufficient for our result and makes the protocol simpler.

5.2 Our protocol

Our protocol is described in Algorithm 5.2. It consists of two phases: 1) the TrustCast protocol to receive the sender's message or create a corrupt-proof when the sender is silent, and 2) the Dolev-Strong style protocol to agree on whether the sender is dishonest, which helps decide whether the sender's message (if received) is committed. Note that each node uses the same trust graph across all slots. So communication to maintain the trust graph and detect malicious senders is bounded and amortized over all slots. Likewise, each $\langle \text{corrupt}, v\rangle_w$ message in the Dolev-Strong phase is shared among all slots and is sent/forwarded only once. So communication in the Dolev-Strong phase is also amortized over all slots.

Algorithm 5.2: Quadratic communication multi-shot BB

Let G_u be an undirected complete graph of numbered n vertices. Each slot $k \geq 1$ takes $T = n + f + 3$ rounds. Each node u runs the following steps.

TrustCast. In round 0, start invoking TrustCast(G_u, n, k) with S_k works as the sender S in the TrustCast. Let m be a value received from S_k (through $\langle \text{prop}, m, k\rangle_{S_k}$) by the beginning of round n.

Dolev-Strong. In each round $n + 1 \leq t \leq n + f + 2$ run the following. Let $\tau = t - (n + 1)$, i.e., rounds after starting this phase.

- $\tau = 0$: if S_k is not in G_u, then multicast $\langle \text{corrupt}, S_k\rangle_u$ (if not sent before).
- $1 \leq \tau \leq f + 1$: If u has received $\langle \text{corrupt}, S_k\rangle_*$ signed by at least τ distinct nodes and S_k is not in G_u, then multicast them (those not sent before) and $\langle \text{corrupt}, S_k\rangle_u$ (if not sent before).

Finally, in round $t = n + f + 2$, if u has not sent $\langle \text{corrupt}, S_k\rangle_u$, then commit m for slot k. Otherwise, commit $\perp$. Note that each $\langle \text{corrupt}, v\rangle_w$ message is shared among all slots and sent/forwarded only once.

5.3 Proof of correctness

We prove the correctness of Algorithm 5.2. Termination and sequentiality are obvious. Below, we say a node u *votes for the corruption of* S_k if u send $\langle \text{corrupt}, S_k\rangle_u$.

Lemma 4 (Validity). *If the sender S_k is honest, then all honest nodes commit the sender's message at slot k.*

Proof. Due to the *integrity* of TrustCast, an honest sender never gets removed from any honest node's trust graph. So honest nodes never vote for the corruption of an honest sender. Thus, all honest nodes always commit the honest sender's message. □

Lemma 5 (Consistency). *If two honest users u and v commit m_u and m_v respectively at slot k, then $m_u = m_v$.*

Proof. We first show that if an honest node u votes for the corruption of S_k, then any honest node v also votes for it.

Suppose u sends $\langle \text{corrupt}, S_k\rangle_u$ before the last round (i.e., $\tau < f + 1$), then v receives it by the beginning of round $\tau + 1$. The node u must have removed S_k from G_u in round τ. Due to the *transferability* of TrustCast, v removes S_k from G_v in round $\tau + 1$. Also, u must have forwarded the corrupt messages from at least τ nodes, so v receives the corrupt messages from $\tau + 1$ distinct nodes by round $\tau + 1$. Thus, u also votes for the corruption of the S_k.

Suppose u sends $\langle \text{corrupt}, S_k \rangle_u$ at the last round (i.e., $\tau = f + 1$). Then u must have received the corrupt messages from $f + 1$ distinct nodes, at least one of them must be from an honest node, who must have sent it before the last round. The analysis above implies v votes for the corruption of S_k.

Therefore, if an honest node commits $\perp$, then all honest nodes also commit $\perp$.

Suppose an honest node u commits $m \neq \perp$, then honest nodes do not commit $m' \neq m$; otherwise, these two different prop messages are forwarded to all honest nodes by the beginning of round $n + 1$ (when the Dolev-Strong phase starts), which would lead to all honest nodes voting for the sender's corruption and u would not commit m. Since none of the honest nodes could have voted for the sender's corruption, they must have received m by the *termination* of TrustCast, hence they all commit m. Therefore, all honest nodes commit the same value at the same slot. □

5.4 Communication complexity.

We analyze the communication complexity of Algorithm 5.2.

- In the TrustCast protocol, as analyzed in Section 5.1, an honest node multicasts the sender's messages at most twice. This adds up to $O(\kappa n^2)$ communication complexity per instance. Since we use the same trust graph for all slots, the cost of maintaining the trust graph is $O(\kappa n^4)$ across all instances.
- In the proof of Lemma 5, we showed that if any honest node sent a corrupt message during Dolev-Strong, then all honest nodes would remove the sender from their trust graphs. Therefore, there can be at most f instances where any honest node sends messages in the Dolev-Strong protocol. So the communication complexity for the Dolev-Strong protocol is upper bounded by $O(\kappa n^3 \cdot f)$ across all instances.

To sum up, the total cost is $O(\kappa n^2 L + \kappa n^4)$ for L slots.

6 CONCLUSION AND OPEN QUESTIONS

This paper studied amortized communication complexity of multi-shot Byzantine broadcasts and presented protocols with linear complexity for the honest majority and quadratic complexity for the dishonest majority using a novel data dissemination and node accusation technique. We finalize the paper with two open questions. First, our protocol for the dishonest majority is not known to be optimal. It is an interesting question whether quadratic communication is necessary or linear complexity is possible under a dishonest majority. Another question is whether we can also achieve the same complexity under partial synchrony or asynchrony. Since nodes can accuse honest senders in an asynchronous network, we need a mechanism to refresh the trust information (e.g., the history of accusations) maintained, which is an interesting technical challenge.

REFERENCES

[1] I. Abraham, D. Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118, 2020.

[2] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

[3] Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science*, pages 313–321. Springer, 1992.

[4] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *Theory of Cryptography Conference*, pages 353–380. Springer, 2020.

[5] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2003.

[6] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[7] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. *arXiv preprint arXiv:2002.06545*, 2020.

[8] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158–179, 1995.

[9] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[10] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[11] Chaya Ganesh and Arpita Patra. Broadcast extensions with optimal communication and round complexity. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 371–380. ACM, 2016.

[12] Chaya Ganesh and Arpita Patra. Optimal extension protocols for byzantine broadcast and agreement. *Distributed Computing*, pages 1–19, 2020.

[13] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. *arXiv preprint arXiv:2106.10362*, 2021.

[14] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[15] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[16] Jens Groth. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive*, 2021.

[17] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[18] Valerie King and Jared Saia. Breaking the o (n 2) bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):1–24, 2011.

[19] Klaus Kursawe and Victor Shoup. Optimistic asynchronous atomic broadcast. In *International Colloquium on Automata, Languages, and Programming*, pages 204–215. Springer, 2005.

[20] Guanfeng Liang and Nitin Vaidya. Error-free multi-valued consensus with byzantine failures. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 11–20, 2011.

[21] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as pipelined bft. *arXiv preprint arXiv:2103.09425*, 2021.

[22] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[23] Oded Naor and Idit Keidar. Expected linear round synchronization: The missing link for linear byzantine smr. In *34th International Symposium on Distributed Computing*, 2020.

[24] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[25] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

[26] HariGovind V Ramasamy and Christian Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In *International Conference On Principles Of Distributed Systems*, pages 88–102. Springer, 2005.

[27] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[28] Alexander Spiegelman. In search for an optimal authenticated byzantine agreement. In *35th International Symposium on Distributed Computing*, 2021.

[29] The DiemBFT Team. State machine replication in the diem blockchain, 2021. https://developers.diem.com/docs/technical-papers/state-machine-replication-paper/.

[30] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. In *Theory of Cryptography Conference*, pages 381–411. Springer, 2020.

[31] Avi Wigderson, MB Or, and S Goldwasser. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, pages 1–10, 1988.

[32] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

A COMMUNICATION COMPLEXITY OF HOTSTUFF

HotStuff [32] is a partially synchronous atomic broadcast protocol with $f < n/3$. To briefly review, it progresses through repeated *views* (epochs), with each view having a unique leader. In each view, a leader proposes a value along with a certificate for the previous decision. Upon receiving a leader's proposal, nodes send votes (threshold signature shares) for it to the leader, and the leader collects $n - f$ votes to generate a certificate and send it to all nodes. After three rounds of voting, the leader sends a commit-proof (an aggregated $n - f$ round-3 votes) to all nodes, and everybody commits the value. We can also think of HotStuff as a synchronous multi-shot Byzantine broadcast in *failure-free* cases. More specifically, if we consider the leader of each view v as a sender S_v of slot v, all honest nodes commit a single value at every slot v at the end of view v.

However, we point out HotStuff must have a fallback path (not fully specified in [32]) to prepare for dishonest senders/leaders. Otherwise, we can have a permanent liveness failure due to the message dissemination problem mentioned in Section 1.1. More specifically, a dishonest leader may not send messages to at most f honest nodes. The leader can still create a certificate and a commit-proof by talking to the rest of the nodes. The f honest nodes left behind cannot commit values at the slot. Existing works address the above issue by either introducing a deterministic round synchronization with quadratic cost [29], or a randomized round synchronization with expected linear cost [23]. This work exactly addresses this liveness issue with a deterministic solution of amortized linear communication under synchrony.