Pando: Extremely Scalable BFT Based on Committee Sampling

Xin Wang
Institute for Advanced Study
Tsinghua University
Beijing, China
wangxin87@tsinghua.edu.cn

Haochen Wang
Institute for Advanced Study
Tsinghua University
Beijing, China
whc20@mails.tsinghua.edu.cn

Haibin Zhang
Department of Information Technology
Yangtze Delta Region Institute of Tsinghua University
Zhejiang, China
bchainzhang@aliyun.com

Sisi Duan
Institute for Advanced Study
Tsinghua University
Beijing, China
duansisi@tsinghua.edu.cn

***Abstract*—Byzantine fault-tolerant (BFT) protocols are known to suffer from the scalability issue. Indeed, their performance degrades drastically as the number of replicas n grows. While a long line of work has attempted to achieve the scalability goal, these works can only scale to roughly a hundred replicas.**

In this paper, we develop BFT protocols from the so-called committee sampling approach that selects a small committee for consensus and conveys the results to all replicas. Such an approach, however, has been focused on the Byzantine agreement (BA) problem (considering replicas only) instead of the BFT problem (in the client-replica model); also, the approach is mainly of theoretical interest only, as concretely, it works for impractically large n.

We build an extremely efficient, scalable, and adaptively secure BFT protocol called Pando in partially synchronous environments based on the committee sampling approach. In particular, we devise novel BFT building blocks targeting scalability, including communication-efficient and computation-efficient consistent broadcast and atomic broadcast protocols.

Pando inherits some inherent issues of committee sampling-based protocols: Pando can only achieve near-optimal resilience (i.e., $f < (1/3 - \epsilon)n$, where f is the number of faulty replicas and ϵ is a small constant), and Pando attains safety and liveness only probabilistically. Interestingly, to make ϵ come close to 0 (near-optimal resilience), n needs to be sufficiently large but not impractically large, e.g., $n > 500$—just what we need for scalable BFT.

Our evaluation on Amazon EC2 shows that in contrast to existing protocols, Pando can easily scale to a thousand replicas in the WAN environment, achieving a throughput of 62.57 ktx/sec.

1. Introduction

Byzantine fault-tolerant (BFT) protocols—handling arbitrary failures and attacks—are nowadays the de facto model of permissioned blockchains and are being increasingly used in permissionless blockchains [1], [2]. However, BFT protocols are known to suffer from the scalability doom, i.e., their performance degrades significantly as the number of replicas grows. In this regard, BFT is in sharp contrast to permissionless blockchains that usually consist of a large number of replicas, e.g., over a million[1] in Ethereum [3].

To overcome the scalability challenge, several approaches have been introduced, including sharding-based BFT protocols that operate in a number of BFT shards [4]–[8]. Most of these protocols, however, use an overly strong assumption requiring that *each* shard does not have more than one-third or half faulty replicas; meanwhile, cross-shard transactions cause consistency issues (requiring, e.g., rollback) and significant communication overhead. Conventional BFT approaches introduce techniques such as parallelism [9], [10] or de-coupling block transmission from consensus [11]–[14]. While these protocols mark significant milestones for scalable BFT, they can support roughly a hundred replicas only in the WAN environment. It is an open problem to scale BFT to, say, 1,000 replicas.

The overhead of existing approaches, briefly. The main bottlenecks for existing BFT approaches are the communication overhead and the computational overhead. The communication becomes prohibitively high as n grows. Meanwhile, existing approaches use threshold signatures (or a set of $O(n)$ signatures) for quorum certificates (QCs) to lower the communication and the authenticator complexity. The computational overhead they caused at a single replica is proportionally higher when n increases, thereby hurting scalability.

Our approach. We propose Pando, an adaptively-secure and scalable BFT protocol in the partially synchronous model, where there exists an unknown upper bound on message transmission and processing [16]. We follow the

1. Data source (accessed in June 2024): https://www.beaconcha.in/

protocols	resilience	transmission	consensus	timing
Narwhal [11]/Bullshark [12]	$f < n/3$	$O(Ln^2 + \kappa n^4)$	$O(\kappa n^3)$	partial sync.
Tusk [11]	$f < n/3$	$O(Ln^2 + \kappa n^4)$	$O(\kappa n^3)$	async.
Dumbo-NG [15]	$f < n/3$	$O(Ln^2 + \kappa n^3)$	$O(\kappa n^3)$	async.
Star [14]	$f < n/3$	$O(Ln^2 + \kappa n^3)$	$O(\kappa n^3)$	partial sync.
Pando (this work)	$f < (1/3 - \epsilon)n$	$O(Ln^2 + \kappa^2 n^2)$	$O(\kappa^2 n^2)$	partial sync.

TABLE 1: Communication complexity of BFT systems that decouple block transmission from consensus on the order. L is the size of input (i.e., a block proposal) of every replica, κ is the length of the cryptographic security parameter, and ϵ is a small constant and can come close to 0 with appropriately chosen parameters. Following all prior work, we simply use $O(\kappa)$ as the committee size and doing so ensures the needed security bound. We assume all protocols instantiate the quorum certificates (QCs) with a set of digital signatures. In practice, the size of QCs for all protocols (including ours) can be optimized using aggregate signatures.

framework that de-couples block transmission from the agreement on the block order, as this model is known to achieve high performance under high concurrency of transactions [11], [14], [15]. As summarized in Table 1, our work reduces the communication complexity of the *transmission* and *consensus* processes, the crucial building blocks in the model that de-couples block transmission from consensus. Additionally, we also improve the message complexity of the *state transfer* process (which is used for data synchronization) from $O(n^2)$ to $O(\kappa n)$.

Our approach is inspired by a line of work on scalable Byzantine agreement and Byzantine broadcast, where a small committee of $O(\kappa)$ replicas is selected among n (sufficiently large) replicas. Such protocols have been studied in both the synchronous setting [17]–[21] and the asynchronous setting [22], [23]. For these committee-based protocols, a possible workflow is to sample a committee, have the committee members reach an agreement, and then ask the committee members to convey the results to all replicas. However, such an approach works only in the static security model, where the adversary is restricted to choosing the set of corrupted replicas at the start of the protocol but fails to work in the adaptive security model, where the adversary can choose the set of corrupted replicas at any moment during the execution of the protocol based on the state it accumulated. (For instance, PBFT [24] attains adaptive security, while HoneyBadgerBFT [25] achieves static security only.) In fact, prior work on scalable Byzantine agreement and Byzantine broadcast has been focused on the adaptive security model, and it is less interesting to study statically secure protocols. Also, note that the line of work has not explored practical BFT or atomic broadcast protocols yet.

In this work, we design and implement the first practical committee-based BFT protocol in the adaptive adversary model. Compared to prior committee-based approaches, our approach utilizes the Chernoff bound in a novel manner to provide a new bound on committee size. The core is to bound the committee size such that the fraction of Byzantine replicas in the committee remains the same (except with a small probability) as that in the entire system. Additionally, our protocol gains in improved communication compared to *all* prior work that de-couples block transmission from consensus. Our work utilizes different techniques to optimize the communication complexity of all three building blocks, as summarized below.

- For the transmission process (for disseminating proposals), we provide a communication-efficient consistent broadcast (CBC) protocol [26], a crucial building block for the transmission process of all the protocols of the same kind. Based on the improved CBC protocol, we provide a transmission process that achieves $O(Ln^2 + \kappa^2 n^2)$ communication, where L is the size of each replica's input and κ is the committee size. Each quorum certificate (QC) generated by the transmission process consists of only $O(\kappa)$ digital signatures (or an aggregate signature with $O(\kappa + \kappa \log \kappa)$ size) such that its communication cost does not grow as n increases. In contrast, all prior practical QC implementations consist of $O(n)$ signatures or an aggregate signature with $O(\kappa + n \log n)$ size.
- We instantiate the consensus process (for agreement on the transaction order) with a partially synchronous atomic broadcast protocol [27] that has $O(|M|n + \kappa^2 n)$ communication and $O(\kappa n)$ messages, where $|M|$ is the size of input to the atomic broadcast protocol. Compared to prior work (e.g., HotStuff [28] has $O(|M|n + \kappa n^2)$ communication and $O(n)$ messages if QCs are instantiated by digital signatures and PBFT [24] has $O(|M|n + \kappa n^2)$ communication as well), our protocol gains in improved communication for $n > \kappa$. We use the new atomic broadcast protocol in the consensus process that achieves $O(\kappa^2 n^2)$ communication.
- Finally, we provide a simple yet efficient state transfer process (for proposal synchronization) with $O(\kappa n)$ messages. Our state transfer process is more efficient than prior constructions involving all-to-all communication (and thus requiring $O(n^2)$ messages).

Note that our communication improvement focuses on the κ term. The improvement is more evident with n growing, especially when we look at concrete complexity—which is validated via our experiments.

Our contributions. We make the following contributions.

- We propose Pando, an adaptively secure and scalable BFT protocol. Compared to prior work that also de-couples block transmission from agreement on the order, our work optimizes both the communication and computational cost of the underlying building blocks.
- Our work explores the new BFT design from the committee sampling approaches which to date have mostly

been studied in the theoretical community with a focus on Byzantine agreement or Byzantine broadcast only. The only price is that the protocol requires $f < (1/3 - \epsilon)n$. Namely, the protocol achieves near-optimal resilience only (due to the ϵ parameter). In Pando, the value of ϵ can come close to 0, when n gets moderately large.

- We implement our protocol and evaluate its performance on Amazon EC2. We show Pando can easily scale to 1,000 replicas in the WAN network and achieve a throughput of 62.57 ktx/sec.

2. Related Work

Partially synchronous BFT. Partially synchronous BFT has been widely studied in the literature [29]. Starting from PBFT [24], an impressive number of practical BFT protocols are proposed (e.g., [30]–[34]). HotStuff [28] provides a three-phase solution that achieves linear message complexity, and many efforts have been made to reduce the number of phases required [35]–[38]. Our new atomic broadcast protocol in the consensus process is a scalable version of prior protocols such as PBFT and HotStuff: our protocol has three phases of communication similar to that in PBFT; the *locked* blocks for safe view changes (i.e., leader election) follows the HotStuff technique.

Scalable BFT. Beginning with Narwhal [11], Bullshark [12], Dumbo-NG [13], and Star [14] use a framework that de-couples the transmission of block proposals (also called the transmission process) from the agreement on the order of the blocks (called the consensus process). Our protocol also follows the framework. As shown in Bullshark [12], Bullshark and Narwhal share almost identical throughput in normal cases, and BullShark offers almost 2x the throughput of Mir-BFT [9] at the same latency. The most recent partially synchronous BFT protocol that separates transmission from consensus, Star [14], is shown to achieve 2.38x the throughput of Narwhal when $n = 91$. We choose Star and Narwhal for the performance comparison.

Besides those works mentioned in the introduction, many alternative solutions can improve the scalability of BFT, such as using trusted hardware [39], [40] and network/topology-level optimization [41].

Byzantine agreement and Byzantine broadcast at scale. King and Saia [18] presented the first committee sampling based Byzantine agreement protocol in the synchronous setting and the protocol achieves $O(n^{1.5})$ communication. The committee sampling mechanism was called the *sampler* protocol, and an ideal sampler is assumed. In particular, the sampler samples "*subsets of elements such that all but a small number contain at most a fraction of bad elements close to the fraction of bad elements of the entire set*". Many works improved the complexity of communication in the Byzantine agreement and Byzantine broadcast protocols, assuming the existence of a sampler. Abraham et al. [19] proposed a binary Byzantine agreement with subquadratic communication complexity. In the asynchronous setting, Blum, Katz, Liu-Zhang, and Loss [22] presented a Byzantine agreement protocol achieving subquadratic communication complexity under the adaptive adversary setting assuming $f < (1-\epsilon)n/3$ (Note that this is interchangeable with our $f < (1/3 - \epsilon)n$ assumption). Additionally, a line of work studies Byzantine broadcast, a problem limited to the synchronous setting where $f < (1 - \epsilon)n$, and uses committee-based approaches to optimize the communication [20], [21].

Algorand [42], [43] is a practical committee sampling based Proof-of-Stake protocol. The VRF-based committee sampling mechanism is a practical instantiation of the sampler notion by King and Saia [18]. Our protocol also adopts the VRF-based committee sampling mechanism by Algorand. Both Algorand and Pando assume a partially synchronous network. Our Pando protocol is different from Algorand. First, Pando achieves a more balanced network bandwidth utilization by employing a leaderless feature for block proposals. Namely, *all* n replicas can create a block proposal, and replicas agree on at least $n - f$ proposals at a time. In contrast, Algorand only agrees on one block proposal at a time. Accordingly, Pando is more efficient than Algorand. Second, the design of Pando allows for a much smaller committee size. In particular, the probability for the system to achieve security properties such as safety and liveness is the same for the two protocols, but Pando requires a much smaller committee size. For instance, to limit the probability of safety violation to 10^{-9} and to set $\epsilon = 0.12$ (i.e., the system has 80% correct replicas), Algorand needs a committee size of 2,000 replicas [42, Figure 3]. In contrast, Pando only needs a committee size of 200 (Figure 3). This is achieved by carefully designing the consensus process (i.e., our partially synchronous atomic broadcast protocol). In terms of experimental comparison, we focus on the comparison of Pando versus protocols in the same category (BFT protocols that also de-couple block transmission from consensus) and do not compare Pando with Algorand.

Proof-of-Stake (PoS). Ethereum's PoS [44] utilizes the concept of the committee to aggregate the votes (i.e., attestations) from replicas to improve performance. The random coins of committee sampling are generated on-chain via the beacon chain. In Delegated PoS (DPOS) [45], a committee is first selected according to certain rules and the committee is in charge of reaching an agreement and conveying the results to the replicas. Such an approach, as mentioned in the introduction, fails to achieve adaptive security.

BFT with adaptive security. Protocols that are secure in the static adversary model might not be adaptively secure [46], [47]. BFT using statically secure threshold cryptography (e.g., threshold signatures or threshold encryption) may not be adaptively secure. Indeed, designing BFT in the adaptively secure model is more challenging. Meanwhile, adaptive security may come with a price. For instance, EPIC [48] and Hale [49] studied how to achieve adaptive security in the asynchronous model and showed that while practical asynchronous BFT in the adaptively secure model is possible, the performance degrades up to 30% compared to its counterpart in the static model. Note that a lot of conventional partially synchronous protocols (that do not use threshold signatures) achieve adaptive security [48], e.g., PBFT [24] and BFT-SMaRt [50]. Namely, these protocols all use a view change protocol (i.e., leader election) that rotates the leaders. The protocols thus achieve $O(n)$ time under both static adversary assumption and adaptive adversary assumption, i.e., if f continuous leaders are faulty, the system achieves zero throughput in the interim.

Asynchronous BFT. The celebrated FLP result [51] rules out the possibility of deterministic consensus in asynchronous environments, so asynchronous must be probabilistically live. Asynchronous BFT protocols have been extensively studied [25], [48], [52]–[57]). Our transmission process and state transfer process are fully asynchronous.

3. System Model and Building Blocks

BFT. We study Byzantine fault-tolerant state machine replication (BFT) protocol. In a BFT protocol, clients *submit* transactions (requests) and replicas *deliver* them. The client obtains a final response to the submitted transaction from the replica responses.

A BFT system with n replicas, $\{P_1, \cdots, P_n\}$, can tolerate $f < (1/3 - \epsilon)n$ Byzantine failures, which is *optimal*. Following prior work on scalable Byzantine agreement, this paper considers *near-optimal resilience*, i.e., $f < (1/3-\epsilon)n$, where ϵ is a small constant and $0 < \epsilon < 1/3$.

We consider a partially synchronous network where there exists a Global Stabilization Time (GST), after which the network becomes synchronous. We consider a (weakly) adaptive adversary. Such an adversary can selectively corrupt the replicas while the protocol is running but cannot perform "after-the-fact-removal" and retroactively erase the messages the replica sent before they become corrupted. Additionally, we assume "atomic sends" [22] where an honest replica P_i can send a message to multiple replicas and the adversary can corrupt P_i either before or after it sends the message to all receivers.

We follow prior works [24], [27], [28], [35] and define several notations. A Byzantine quorum is a set of replicas. If we consider a system with n replicas and f Byzantine failures, a Byzantine quorum consists of $\lceil \frac{n+f+1}{2} \rceil$ replicas, or simply $2f+1$ out of $n = 3f+1$ replicas. A set of signatures generated by a Byzantine quorum is called a *quorum certificate (QC)* or a *certificate*.

In this work, we sample a set of $\lambda = O(\kappa)$ committee members, where κ is the length of the security parameter. Following prior protocols, we consider $\lambda = \kappa$ for simplicity. Our protocol ensures that except with negligible probability, the number of faulty replicas in each committee is $t < \lfloor \frac{\kappa-1}{3} \rfloor$. Slightly abusing the notation, we also use the term QC in the committee to denote $\kappa - t$ signatures from committee members.

A BFT protocol we consider in this work satisfies the following properties with probability $1 - \mathsf{negl}(\kappa)$, where $\mathsf{negl}(\kappa)$ is a negligible function in κ.

- **Safety**: If a correct replica *delivers* a transaction tx before *delivering* tx', then no correct replica *delivers* a transaction tx' without first *delivering* tx.
- **Liveness**: If a transaction tx is *submitted* to all correct replicas, then all correct replicas eventually *deliver* tx.

BFT protocols do not need to expose an explicit order for blocks of transactions, but the concrete constructions may assign an order to each block. In this work, we use *height* to denote the order of a block. Namely, in a chain of blocks, the height of each block is the number of blocks on the chain rooted by the genesis block. For a QC qc, we use the function $height(qc)$ to denote the height of the block for qc. Each replica uses a tree-based data structure to store the blocks proposed by all the replicas. Block b *extends* b' if b extends the branch led by b'.

Atomic broadcast. We also use atomic broadcast as a building block. Atomic broadcast is only syntactically different from BFT; in atomic broadcast, a replica *a-broadcasts* messages and all replicas *a-deliver* messages. An atomic broadcast protocol satisfies the following properties with probability $1 - \mathsf{negl}(\kappa)$.

- **Safety**: If a correct replica *a-delivers* a message m before *a-delivering* m', then no correct replica *a-delivers* a message m' without first *a-delivering* m.
- **Liveness**: If a correct replica *a-broadcasts* a message m, then all correct replicas eventually *a-deliver* m.

Here, we restrict the API of atomic broadcast such that only a single replica *a-broadcasts* a transaction. One can alternatively allow all replicas to *a-broadcast* transactions.

3.1. Building Blocks

Consistent broadcast (CBC). We review the definition of consistent broadcast (CBC). A CBC protocol is specified by *c-broadcast* and *c-deliver* such that the following properties hold:

- **Validity**: If a correct replica p *c-broadcasts* a message m, then p eventually *c-delivers* m.
- **Consistency**: If two correct replicas *c-deliver* two messages m and m', then $m = m'$.
- **Integrity**: For any message m, every correct replica *c-delivers* m at most once. Moreover, if the sender is correct, then m was previously *c-broadcast* by the sender.

The $\mathsf{ComProve}()$/$\mathsf{ComVerify}()$ oracle. We follow prior works [19]–[21] and define a $\mathsf{ComProve}()$/$\mathsf{ComVerify}()$ oracle as a committee sampling function. We present in Algorithm 1 the functionality of $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$ [21]. $\mathsf{ComProve}()$ is parametered by the total number of replicas and a *mining* probability p_{mine}. It is specified by two functionalities: $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$. In particular, a replica P_i can query $\mathsf{ComProve}(m, i)$ to check whether it is an eligible member of the committee, where m is the designated input. The query of the $\mathsf{ComProve}()$ function is also called a *mining* attempt. Upon receiving a mining attempt for the first time, $\mathsf{ComProve}()$ flips a random coin and returns a binary result. It returns 1 with mining probability p_{mine}. If 1 is returned,

P_i is part of the committee. After P_i has successfully made a mining attempt, $\mathsf{ComVerify}(m, i)$ returns the same answer for all future identical queries to any replica.

In this work, we use the notation C_x^y to denote the committees, where the subscript x specifies the corresponding process (i.e., transmission, consensus, or state transfer) and epoch number, and the superscript y denotes the instance number. For instance, $C_{t,e}^j$ denotes the committee used in the transmission process for the j-th instance in epoch e. In this case, we can instantiate the $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$ functions as follows: replica P_i queries $\mathsf{ComProve}(t||e||j, i)$ to learn whether it is a committee member where $||$ denotes concatenation; after P_i queries the $\mathsf{ComProve}()$ function, any replica P_k queries $\mathsf{ComVerify}(t||e||j, i)$ to verify whether P_i belongs to $C_{t,e}^j$.

We instantiate $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$ with the Verifiable Random Function (VRF). In particular, depending on the committee size, we set up a difficulty parameter D. When P_i generates a VRF evaluation for $t||e||j$ (the $\mathsf{ComProve}(t||e||j, i)$ function). P_i belongs to $C_{t,e}^j$ if the VRF evaluation is lower than D. When P_i sends some message to other replicas, P_i also includes the VRF evaluation to the replicas. When P_k queries $\mathsf{ComVerify}(t||e||j, i)$, the function returns true if the VRF evaluation is lower than D.

4. Motivation and Overview

4.1. Review of Existing De-coupling Approaches

Narwhal [11], Dumbo-NG [15], Bullshark [12], and Star [14] all employ a framework that de-couples block dissemination from the agreement on block order. Such a framework usually involves three processes: a transmission process where each replica creates a proposal, sends to all replicas, and collects matching signatures from a sufficiently large fraction of replicas to form a quorum certificate (QC)—each QC proves that the corresponding transactions are valid and available; a consensus process where replicas reach an agreement on the order of the QCs (so the order of the transactions can be finalized); after an agreement is reached, replicas that do not hold the proposals run state transfer with other replicas.

Algorithm 1 The $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$ oracle. m is a tuple that consists of the designated inputs of the function.

```
1: public parameters: let $p_{mine}$ be the mining probability
2: local parameters: let $call_i \leftarrow \perp$ for any $i \in [n]$
3: function COMPROVE($m, i$)
4:     if $call_i = \perp$ then
5:         let $b \leftarrow 1$ with probability $p_{mine}$ or 0 otherwise
6:         $call_i \leftarrow b$
7:     end if
8:     return $call_i$
9: end function
10: function COMVERIFY($m, j$)
11:     return $call_j$
12: end function
```

As an example, we show the Star framework in Figure 1 (as Star outperforms other protocols). In Star, the transmission process is a pipelining mode of weak consistent broadcast (wCBC) instances. The protocol is epoch-based and each epoch consists of n parallel wCBC instance. In each instance, each replica P_i broadcasts its proposal to the replicas and expects to collect a *weak* quorum certificate (wQC) of $f+1$ matching signatures. In each epoch, at least $n-f$ wQCs are expected to be collected. In the consensus process, the $n-f$ wQCs are used as input. As the input of the consensus process consists of only wQCs instead of the message payload, the consensus process does not become the bottleneck of the system anymore. Star uses PBFT or Dashing [14] as the consensus process. Finally, after an agreement on the order of the wQCs is reached in the consensus process, replicas that have not received the corresponding proposals need to synchronize with other replicas via a state transfer process.

Star, Narwhal, Bullshark, and Dumbo-NG utilize different protocols in different processes. Narwhal and Bullshark utilize the direct acyclic graph (DAG) data structure and CBC in the transmission process. Dumbo-NG uses a pipeline mode of CBC that is slightly different from that in Star. In the consensus process, Narwhal uses HotStuff, and Bullshark employs a partially synchronous variant of DAG-Rider [58].

By default, in state transfer process, each replica requests the missing proposals from all other replicas. Dumbo-NG uses erasure coding to achieve a more communication-efficient approach (called "retrieval" in the paper). All these state transfer approaches involve all-to-all communication and achieve $O(n^2)$ messages.

The feature that separates block proposals from consensus makes such protocols achieve great scalability. For example, when deployed in WAN with 91 replicas (using m5.xlarge instances on AWS), Star achieves throughput of 256 ktx/sec, significantly higher than conventional protocols.

4.2. The Scalability Bottlenecks

If we further scale the existing system to a larger number of replicas, performance may degrade significantly due to both communication overhead and computational overhead.

Communication overhead. Most existing protocols rely on all-to-all communication, so it is not surprising that the performance degrades significantly as n further grows. In

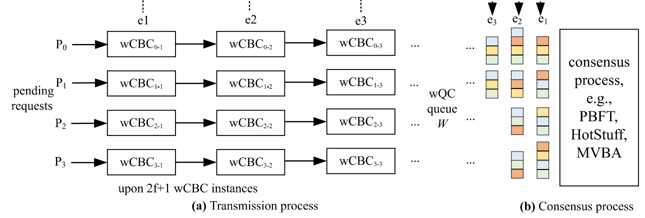
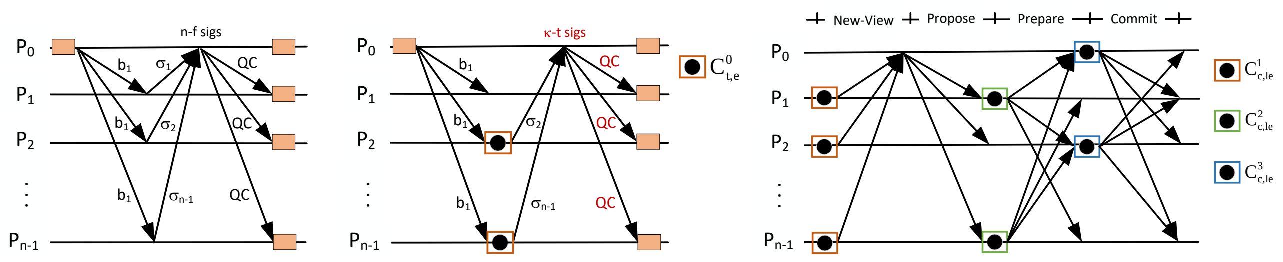


Figure 1: The Star framework [14].



(a) Conventional consistent broadcast (CBC) protocol.

(b) Our scalable CBC approach.

(c) Our scalable atomic broadcast protocol.

Figure 2: Overview of our approach.

the transmission process, the all-to-all communication for block proposal (due to n parallel CBC instances) seems to be unavoidable. However, collecting $O(n)$ signatures and including them in the proposal may again consume high network bandwidth and degrade the performance as n increases. Additionally, the input to the consensus process consists of $O(n)$ QCs and each QC consists of $O(n)$ signatures. As n grows, the communication overhead to the consensus process becomes more significant. Note that even if we use an aggregate signature to replace a set of $O(n)$ digital signatures, the communication cost of each signature is $O(\kappa + n \log n)$, which still grows as n increases.

Computational overhead. Threshold signature is a common technique to lower the communication complexity of the protocols and optimize system performance. Many protocols use threshold signatures to reduce the size of each QC from $O(\kappa n)$ to $O(\kappa)$ [14], [15], [28], [35], [53], [59]. However, threshold cryptosystems may suffer from performance degradation as n grows [52]. In practice, most implementations use a set of $O(n)$ digital signatures (e.g., ECDSA) instead [14], [15], [28], [35], [53]. The communication complexity, however, is increased accordingly as mentioned above.

4.3. Technical Overview

Scalable consistent broadcast (CBC) for the transmission process. We show the conventional CBC protocol in Figure 2a. Our transmission process improves CBC using only one technique, as shown in Figure 2b: instead of letting all replicas reply with a signature to the sender (e.g., P_0), we sample a committee of κ size and only committee members reply with a signature. The underlying idea is that since collecting n digital signatures or using threshold signatures can be expensive when n is large, we can alternatively use the committee-based approach. The leader only needs to collect $O(\kappa)$ signatures as a QC. This immediately brings two benefits. First, instead of having all replicas reply with a signature to each sender, only κ replicas need to do so, so the communication cost does not grow as n grows. Second, as each certificate consists of only $O(\kappa)$ signatures instead of $O(n)$ signatures, the *consensus process* can also be made communication-efficient.

Using a new application of the Chernoff bound, we show that by setting the committee size as $\lambda = \frac{3\alpha}{\epsilon^2} \ln \frac{1}{\delta} = O(\kappa)$, with probability $1 - \mathsf{negl}(\kappa)$, the number of faulty replicas in the committee is less than $t = \lambda/3$, where δ is the desired failure probability and α is a small constant (see Lemma 1). Accordingly, if the sender P_i is correct, with probability $1 - \mathsf{negl}(\kappa)$, at least two-thirds of committee members will reply with a digital signature, so P_i eventually completes the CBC. Following the convention in prior works, we simply use κ as the committee size in this work.

Atomic broadcast at scale for the consensus process. We propose a scalable atomic broadcast protocol. Our insight is also aligned with our improved CBC scheme. In particular, we can already ensure that the fraction of correct replicas in the committee remains the same as the entire system. Instead of letting all replicas exchange their votes, only the committee members send their votes to all replicas, and we can still ensure that at least two-thirds of the committee members will take the same action in each phase of the protocol. The actual proof, as shown in Appendix A, is more involved, but it exploits this insight.

To avoid the security threats in the adaptive security model, we sample three committees in each epoch e of the protocol, denoted as $C^1_{c,e}$, $C^2_{c,e}$, and $C^3_{c,e}$, as illustrated in Figure 2c. After each committee member broadcasts its vote, it will not vote again. Accordingly, even if the committee member is corrupted, it is already *too late* in the weakly adaptive adversary model and the protocol is still live.

The atomic broadcast protocol is communication-efficient due to two reasons. First, the input M of the consensus process is $O(\kappa^2 n)$ instead of $O(\kappa n^2)$ as each QC has $O(\kappa)$ signatures. Second, in each phase of the protocol, only one-to-all or κ-to-all communication is involved and the communication complexity is $O(|M|n + \kappa^2 n)$, where $|M|$ is the size of the input. Note that although protocols like HotStuff only involve one-to-all communication, the communication complexity is $O(|M|n + \kappa n^2)$ if we use digital signatures for the quorum certificates. Our atomic broadcast protocol can be used as a dedicated BFT protocol and is thus of independent interest.

State transfer with $O(\kappa n)$ messages. All prior works achieve $O(n^2)$ messages and involve all-to-all communication, which might be very expensive when n is large. In Pando, we provide a simple yet efficient state transfer ap-

Algorithm 2 The Pando protocol for replica P_i and tag ID

```
1: initialization: start the transmission process and the consensus process
2: upon a-deliver$(le, m)$
3:     $O \leftarrow \mathsf{Obtain}(le, m)$
4:     obtain the non-overlapped transactions in $O$ and deliver in a deterministic order
5:     set $ce \leftarrow le$
```

proach with $O(\kappa n)$ messages and $O(L\kappa n^2)$ communication.

5. The Pando Protocol

5.1. The Generic Workflow

The generic workflow of Pando is presented in Algorithm 2. We also present the utility functions in Algorithm 5. In particular, every replica starts the transmission process and the consensus process when initializing the protocol.

The transmission process is epoch-based, where each replica proposes a batch of transactions in every epoch. A new epoch of the transmission process (Algorithm 3) is started when every replica has a non-empty queue and has received at least $n-f$ proposed messages from the previous epoch. QCs are formed in the transmission process and the queue of QCs W is shared between the transmission process and the consensus process.

The consensus process (Algorithm 4) is also epoch-based: in each epoch, there is a designated leader. For each epoch le, the leader proposes $W[le]$ and $W[le]$ consists of at least $n-f$ QCs. After an agreement is reached, replicas start the state transfer process. If a replica has received the proposals corresponding to the QCs, it delivers the transactions in the proposals. Finally, P_i obtains a set of non-overlapped transactions in O and then delivers the transactions in O in a deterministic order.

5.2. The Transmission Process

The transmission process can be viewed as a scalable version of pipelined consistent broadcast (CBC). In this section, we present a pipelining mode for replicas to propose blocks, where a replica broadcasts the QCs for the prior epoch and also a new block. The pseudocode is shown in Algorithm 3.

The $C^i_{t,e}$ signing committee for each $i \in [n]$. In the transmission process, n committees are sampled for each epoch e. Each committee serves for signing purposes in each CBC instance. For the instance initiated by P_i in epoch e, we use $C^i_{t,e}$ to denote the signing committee, where the subscript t denotes the *transmission* process. The identity of a committee member (i.e., a replica) is not revealed until the replica queries the $\mathsf{ComProve}()$ function and sends a message to the replicas. After a committee member sends out a message, other replicas can verify the identity of the committee member via the $\mathsf{ComVerify}()$ function, as

Algorithm 3 The transmission process for replica P_i and tag ID

```
1: local parameters: let epoch $e \leftarrow 1$, $Q$ be the queue of pending transactions, $proposals$ be the received proposals, $qc_i$ be the latest certificate, $W \leftarrow \perp$ be the queue of certificates.
2: function InitEpoch$(e)$
3:     sample a committee $C^j_{t,e}$ for each $j \in [n]$
4:     $M \leftarrow \mathsf{select}(Q)$
5:     send $(\textsc{Proposal}, e, M, qc_i)$ to all replicas
6:     $h \leftarrow Hash(M)$
7:     upon receiving $\kappa - t$ valid signatures for $(e, h, i)$ from $C^i_{t,e}$
8:         let $qc_i$ be the set of valid signatures
9:     wait until $|proposals[e]| \geq n - f$
10:    $e \leftarrow e + 1$
11:    $\mathsf{InitEpoch}(e)$
12: end function
13: upon receiving $(\textsc{Proposal}, e, M, qc_j)$ from $P_j$ s.t. $j \in [n]$
14:     if $P_i \in C^j_{t,e}$ then
15:         $h \leftarrow Hash(M)$
16:         create a signature $\sigma_i$ for $(e, h, j)$ and send to $P_j$
17:     end if
18:     $proposals[e][j] \leftarrow M$
19:     $W[e-1] \leftarrow W[e-1] \cup qc_j$
```

described in Sec. 3. In the rest of the paper, we omit the details of membership discovery and verification when no ambiguity occurs.

The workflow. To start epoch e, every replica P_i calls the $\mathsf{InitEpoch}(e)$ function (line 2). In this function, P_i obtains a batch of transactions M from its queue Q and then broadcasts a $(\textsc{Proposal}, e, M, qc_i)$ message to all replicas (line 5), where qc_i is the QC formed in epoch $e-1$ (if $e = 1$, $qc_i = \perp$, also known as a genesis block). P_i then waits for $\kappa - t$ matching signatures for (e, h, i) from $C^i_{t,e}$, where h is the hash of M (line 15). For each replica P_i, upon receiving a proposal $(\textsc{Proposal}, e, M, qc_j)$ from P_j, P_i verifies whether it belongs to the committee $C^j_{t,e}$. If so, P_i creates a signature for $(e, Hash(M), j)$ and then sends to P_j. Meanwhile, P_i sets its local parameter $proposals[e][j]$ as M and adds the QC qc_j to its local queue $W[e-1]$ (lines 18-19). Here, qc_j is the QC for the proposal in epoch $e-1$ so qc_j is added to $W[e-1]$.

After P_i collects $\kappa - t$ signatures from $C^j_{t,e}$, the signatures become a QC and the local parameter qc_i is updated accordingly (line 8). Then P_i waits for $n-f$ valid $(\textsc{Proposal})$ messages before entering the next epoch (line 9).

5.3. The Consensus Process

The consensus process is shown in Algorithm 4 and we use a partially synchronous atomic broadcast protocol to instantiate the consensus process. The protocol has four

Algorithm 4 The consensus process for replica P_i

```
1: public parameters: each committee have $\kappa$ replicas and $t \leftarrow \kappa/3$
2: local parameters: let epoch $le \leftarrow 0$, last committed epoch $ce \leftarrow 0$, $lockedQC \leftarrow \perp$, $Received \leftarrow \emptyset$
3: in each epoch $le$, sample three committees $C^1_{c,le}$, $C^2_{c,le}$, and $C^3_{c,le}$
4: ▷ NEW-VIEW phase
5: upon $|W[le]| \geq n - f$
6:     start a timer $\Delta$ and obtain $\ell \leftarrow le \bmod n$
7:     if $P_i \in C^1_{c,le}$ then
8:         send (NEW-VIEW, $le, lockedQC$) to the leader $P_\ell$
9:     end if
10: ▷ PROPOSE phase
11: upon receiving $\kappa - t$ (NEW-VIEW) messages from replicas in $C^1_{c,le}$
12:     if CheckLeader($le, i$) then
13:         $qc_{high} \leftarrow$ the highest QC in (NEW-VIEW) messages
14:         $W_i \leftarrow W[le]$
15:         if $height(qc_{high}) < le - 1$ then
16:             for each $e' \in (height(qc_{high}), le - 1]$
17:                 $W_i \leftarrow W_i \cup W[e']$
18:         end if
19:         create a block $b$ with content $W_i$
20:         broadcast (PROPOSE, $b, le, qc_{high}$)        ▷ a-broadcast
21:     end if
22: ▷ PREPARE phase
23: upon receiving (PROPOSE, $b, e, qc_{high}$) from the leader $P_\ell$ s.t. $le = e$
24:     if $P_i \in C^2_{c,le}$ and CheckLeader($e, \ell$) and IsValid($b$) then
25:         $\sigma_i \leftarrow$ a signature for $(1, hash(b), le)$
26:         broadcast (PREPARE, $hash(b), le, \sigma_i$)
27:     end if
28:     $Received[e] \leftarrow b$
29: ▷ COMMIT phase
30: upon receiving $\kappa - t$ (PREPARE, $h, e, \sigma_j$) from $C^2_{c,le}$ s.t. $le = e$
31:     $lockedQC \leftarrow \kappa - t$ signatures for $(1, h, e)$
32:     if $P_i \in C^3_{c,le}$ then
33:         $\sigma_i \leftarrow$ a signature for $(2, h, le)$
34:         broadcast (COMMIT, $h, le, \sigma_i$)
35:     end if
36: upon receiving $t+1$ (COMMIT, $h, e, \sigma_j$) from $C^3_{c,le}$ s.t. $le = e$
37:     if $P_i \in C^3_{c,le}$ and $P_i$ has not sent (COMMIT) then
38:         $\sigma_i \leftarrow$ a signature for $(2, h, le)$
39:         broadcast (COMMIT, $h, le, \sigma_i$)
40:     end if
41: ▷ Deliver
42: upon receiving $\kappa - t$ (COMMIT, $h, e, \sigma_j$) from $C^3_{c,le}$ s.t. $le = e$
43:     let $m$ be the content in the block $b$ and $h = hash(b)$
44:     if $ce + 1 \neq le$ then
45:         $\boldsymbol{m} \leftarrow$ ObtainMissing($ce + 1, le, m$)
46:         a-delivers each $m_e \in \boldsymbol{m}$ according to the epoch numbers
47:     else
48:         a-deliver($le, m$)        ▷ a-deliver event
49:     end if
50:     set $le \leftarrow le + 1$, $ce \leftarrow le$
51: ▷ View Change
52: upon $\Delta$ times out
53:     set $le \leftarrow le + 1$
```

phases: NEW-VIEW, PROPOSE, PREPARE, and COMMIT.

Algorithm 5 Utilities

```
1: function ISVALID($b$)
2:     if $b$ extends the block for $lockedQC$ and for any $W_e \in b$ for epoch $e$ and VerifyQCs($W_e, e$) returns true and $e' \geq ce$ where $e'$ is the epoch number for any QC included in $b$ then
3:         return $true$
4:     else
5:         return $false$
6:     end if
7: end function
8: function VERIFYQCS($W_j, e$)
9:     if $|W_j| \geq n - f$ and for each $qc_\ell \in W_j$, each $\sigma_k \in qc_\ell$ from $P_k$, ComVerify($t||e||1||\ell, k$) returns 1 and $\sigma_k$ is a valid signature for $(e, *, \ell)$ then
10:         return $true$
11:     else
12:         return $false$
13:     end if
14: end function
15: function CHECKLEADER($e, i$)
16:     if $i = e \bmod n + 1$ then
17:         return $true$
18:     else
19:         return $false$
20:     end if
21: end function
22: function OBTAINMISSING($ce, le, m$)
23:     $\boldsymbol{m} \leftarrow \perp$
24:     for $e \in [ce, le]$
25:         if $\exists W_e$ s.t., $W_e \in m$ then
26:             $\boldsymbol{m}[e] \leftarrow W_e$
27:         else
28:             wait for $m_e$ from the block $b$ proposed in epoch $e$
29:             $\boldsymbol{m}[e] \leftarrow m_e$
30:         end if
31:     return $\boldsymbol{m}$
32: end function
```

The protocol is epoch-based. To differentiate the epoch number from that in the state transfer process, we use le to denote the latest epoch number of the system and ce to denote the last epoch where some value has been *a-delivered*. Every replica also maintains a $lockedQC$, which is updated in the COMMIT phase of every epoch.

The $C^1_{c,le}$, $C^2_{c,le}$, and $C^3_{c,le}$ committees. In each epoch le, three committees are sampled, where the subscript c denotes the *consensus* process. The $C^1_{c,le}$, $C^2_{c,le}$, and $C^3_{c,le}$ committees are used in the NEW-VIEW phase, PREPARE, and COMMIT phases, respectively.

The workflow. There is a designated leader in each epoch le. We use $le \bmod n$ to denote the identity of the leader. Every replica also starts a timer Δ. In case no value is *a-delivered* before Δ expires, replicas enter the next epoch (line 52).

NEW-VIEW phase. Every replica P_i first identifies whether it belongs to $C^1_{c,le}$. If so, it sends a (NEW-VIEW, $le, lockedQC$) message to the leader P_ℓ of epoch le (line 7-8), where $lockedQC$ is a local parameter.

PROPOSE phase. After receiving at least $\kappa - t$ (NEW-VIEW) messages from $C^1_{c,le}$, the leader obtains qc_{high}, the QC with the largest height (i.e., epoch number). If P_i is the leader (i.e., $i = le \mod n$), P_i then obtains the height of qc_{high} (line 13). By default, P_i uses $W[le]$ as the proposal for the current epoch. Additionally, if $height(qc_{high})$ is lower than $le - 1$, it means that the epoch lower than $le - 1$ is not *a-delivered*. In this case, P_i also proposes for epochs between $height(qc_{high})$ and $le - 1$. In particular, for each e' between $height(qc_{high})$ and $le - 1$, P_i appends $W[e']$ to its proposal W_i (lines 15-17). After that, P_i creates a block b with content W_i, the height le, and hash of qc_{high}. Then, P_i broadcasts a (PROPOSE, b, le, qc_{high}) message to all replicas (line 20). Here, we say P_i *a-broadcasts* b.

PREPARE phase. Every replica waits for the proposal from the leader. Upon receiving a (PROPOSE, b, e, qc_{high}) message from the leader P_ℓ, P_i verifies whether b is valid (line 24 and Algorithm 5, lines 1-7). Namely, b is valid if b extends the block of P_i's local $lockedQC$ and each W_e in the proposal consists of $n - f$ valid QCs. After that, if P_i belongs to $C^2_{c,e}$, it broadcasts a (PREPARE, $hash(b), le, \sigma_i$) message to all replicas, where σ_i is a signature for $(1, hash(b), le)$.

COMMIT and DELIVER phases. Every replica expects $\kappa - t$ (PREPARE) messages from $C^2_{c,e}$. If so, the signatures included in the (PREPARE) messages form a QC and every replica updates its local $lockedQC$ (line 31).

If a replica P_i belongs to $C^3_{c,le}$, it creates a signature for $(2, hash(b), le)$ and then sends a (COMMIT, h, le, σ_i) message to all replicas (lines 32-34), where $h = hash(b)$. If P_i belongs to $C^3_{c,le}$, receives $t + 1$ matching (COMMIT) messages from replicas in $C^3_{c,le}$, and has not sent a (COMMIT) message, P_i also sends a (COMMIT, h, le, σ_i) message to all replicas (lines 36-39).

Finally, after each replica receives $\kappa - t$ matching (COMMIT, h, le, σ_i) messages, it is ready to *a-deliver* block b (and the hash of b is h). Before that, P_i also checks whether its last committed epoch is $ce = le - 1$ (line 44). If so, P_i fetches block b (either stored locally or from other replicas) and then *a-delivers* m, the content in block b. Otherwise, P_i queries the ObtainMissing() function to obtain the missing values between $ce + 1$ and $le - 1$ (lines 45-46). In the ObtainMissing(ce, le, m) function, there are two cases for each epoch $e \in [ce, le]$:

- A set of QCs for epoch e is included in m (Algorithm 5, lines 25-26), i.e., the leader has previously included W_e in its proposal. In this case, P_i can include W_e in its output and *a-delivers* the value.
- QCs for epoch e are not included in m (Algorithm 5, lines 27-29). This might be caused by the fact that some correct replica has previously *a-deliverd* some value in epoch e but P_i has not. In this case, P_i waits for a QC from $C^3_{c,e}$ and then synchronizes the proposed block b from other replicas (We ignore the details of how replicas obtain the proposed block based on the hash value as the approach largely follows prior works [24], [28]). Then P_i *a-delivers* the value.

Afterward, P_i *a-delivers* the proposed values sequentially according to the epoch numbers.

Algorithm 6 The state transfer process for replica P_i

```
 1: function OBTAIN($e, m$)
 2:     sample a committee $C^j_{s,e}$ for each $j \in [n]$
 3:     $O \leftarrow \perp$
 4:     for $qc_j \in m$
 5:         if $proposal[e][j] \neq \perp$ then
 6:             $O \leftarrow O \cup proposals[e][j]$
 7:             if $P_i \in C^j_{s,e}$ then
 8:                 broadcast (DISTRIBUTE, $j, proposals[e][j]$)
 9:             end if
10:         end if
11:     upon receiving (DISTRIBUTE, $j, M$) from $P_k$
12:         if $P_k \in C^j_{s,e}$ and $Hash(M)$ matches that corresponding to $qc_j$ then
13:             $O \leftarrow O \cup M$
14:         end if
15:     wait until $|O| = |m|$
16:     clear $W[e]$ and remove transactions in $O$ from $Q$
17: end function
```

5.4. State Transfer

We provide a state transfer mechanism that only involves κ-to-all communication so the message complexity is $O(\kappa n)$. The idea is aligned with our transmission and consensus process and we show the pseudocode in Algorithm 6.

In our state transfer mechanism, n committees are sampled and each one is denoted as $C^j_{s,e}$. Committee members in $C^j_{s,e}$ are in charge of helping other correct replicas collect the proposal from P_j. Namely, if the QC from P_j (denoted as qc_j) is *a-delivered* in the consensus process, every correct replica P_i that belongs to $C^j_{s,e}$ and meanwhile holds the proposal will send a message (DISTRIBUTE, $j, proposals[e][j]$) to all replicas (lines 5-8), where $proposals[e][j]$ is the proposal P_i previously received from P_j in the transmission process. Any correct replica that receives a (DISTRIBUTE, j, M) message verifies whether the hash of M matches that in the *a-delivered* message in the consensus process (lines 11-12). If so, the replica adds M to its output O. Finally, every correct replica P_i waits for the proposals for every QC in m (i.e., $|O| = |m|$) and completes the state transfer.

5.5. Correctness and Complexity

Correctness. Our protocol is secure under a weakly adaptive adversary. This is because an adversary cannot corrupt too many members in each committee until *it is too late*, except with negligible probability. Namely, in all three processes, every member of each committee only sends a message once. Therefore, even if the adversary learns that the replica is in a committee, the message has already been sent so corrupting the replica is useless.

While we show the proof of correctness in detail in Appendix A, we briefly sketch the correctness below.

Safety. Roughly, the safety of the atomic broadcast protocol ensures the safety of the BFT. For the safety of our atomic broadcast protocol, a crucial observation is that a committee can *convey* the agreement result to all replicas. Informally, the crux is to show that if a correct replica P_i has received $\kappa - t$ matching (COMMIT) messages from $C^3_{c,e}$ in epoch e for an *a-broadcast* message m, no correct replica will *a-deliver* message $m' \neq m$ in epoch e and any correct replica will only vote for a (PROPOSE) message that excludes m in epochs greater than e. The property that no correct replica will *a-deliver* message $m' \neq m$ in epoch e is ensured by the fact that each committee has at least $\kappa - t$ correct replicas with $1 - \mathsf{negl}(\kappa)$ probability (Lemma 1). Indeed, if at least $\kappa - t$ replicas in $C^3_{c,e}$ are correct, no correct replica will *a-deliver* m' by the quorum intersection rule [27].

Additionally, we also need to ensure that every correct replica will only vote for a (PROPOSE) message that excludes m in epochs greater than e. This is achieved by two factors. First, if P_i has received $\kappa - t$ matching (COMMIT) messages from $C^3_{c,e}$ in epoch e, at least $f + 1$ correct replicas in the entire system are locked on m after receiving $\kappa - t$ matching (PREPARE) messages from $C^2_{c,e}$. This can also be proved using the Chernoff bound and we show the correctness in Lemma 2 and Corollary 2. Second, to ensure that a correct leader always proposes the *correct* proposal (that excludes a message already *a-delivered* by at least one correct replica), we introduce a New-View phase where a committee is sampled and the committee members send their $lockedQC$ to the leader P_ℓ. In this way, the leader can receive the highest locked QC.

Liveness. Liveness of the system requires understanding all three processes. First, our transmission process ensures that if a QC is formed, at least $f + 1$ correct replicas have previously received the proposal. We show in Lemma 5 that this happens with probability $1 - \delta$. Second, the probability that a message m *a-broadcast* by a correct replica is not *a-delivered* decreases as the system proceeds. Informally, this is because a correct replica will *a-broadcast* a message that has previously been proposed but still not *a-delivered* yet. Indeed, in our atomic broadcast protocol, we intentionally *bind* the epoch number with the leader. In particular, every leader p_ℓ (where $\ell = le \mod n + 1$) only proposes $W[le]$ (the certificates for proposals in epoch le of the transmission process, line 14 of Algorithm 4). This can greatly simplify the state transfer process but it is possible that some certificates in epochs lower than le cannot be *a-delivered*. To address this issue, the leader p_ℓ also proposes the certificates between epoch $height(qc_{high})$ and $le - 1$ (lines 15-17). In this way, even if some leaders fail to propose any value in their turn, a set of $n - f$ certificates can still be proposed with the help of a correct leader. The amortized number of certificates *a-delivered* in every epoch is thus $n - f$. Finally, our state transfer process samples a committee and we just need to ensure that at least one correct committee member has previously received the proposal from the leader. As we already know that at least $f + 1$ correct replicas in the system have previously received the proposal, it is not difficult to see that with overwhelming probability, at least one correct committee member has received the proposal. We show in Lemma 6 that the probability is $1 - \delta^{\frac{\frac{1}{9} - \epsilon^2}{\epsilon^2}}$.

Complexity. As our protocol only involves one-to-all and κ-to-all communication, the message complexity is $O(\kappa n)$. The communication complexity of the transmission process, consensus process, and state transfer process is $O(Ln^2 + \kappa^2 n^2)$, $O(\kappa^2 n^2)$, and $O(L\kappa n)$, respectively. We leave the detailed complexity analysis in Appendix B.

6. Analysis of Probability of Achieving Safety and Liveness

We analyze the concrete probability of safety and liveness violation of Pando in Appendix C and we summarize our results in this section. In Lemma 1, we show that if we use a committee size of $\frac{3\alpha}{\epsilon^2} \ln \frac{1}{\delta} = O(\kappa)$, with probability $1 - \delta$, the number of faulty replicas in the committee is no more than $t = \lfloor \frac{\kappa}{3} \rfloor$. If we set $\delta = e^{-\omega(\log \kappa)}$, δ is a negligible function. Using δ as a parameter, we analyze the concrete probability of safety and liveness violation.

Probability of safety violation. Safety of the system is violated if a correct replica *a-delivers* m and another correct replica *a-delivers* m' and $m \neq m'$ in the consensus process. As shown in Theorem 1, the probability of safety violation is $O(\delta^2)$.

An interesting fact is that the probability of safety violation is related to the number of phases in the consensus process. Informally, consider the protocol within a view, there are two phases of κ-to-all communication (i.e., the PREPARE phase and the COMMIT phase), and we rely on the committees $C^2_{c,e}$ and $C^3_{c,e}$ to achieve the security properties. Safety is violated only if neither committee has at least $\kappa - t$ correct replicas, i.e., the probability of safety violation is $O(\delta^2)$. Additionally, our proof shows that the probability of safety violation across views is significantly lower than $O(\delta^2)$. Thus, the probability of safety violation of the protocol is bounded by $O(\delta^2)$.

Notably, we can modify the consensus process to have *more* phases to lower the probability of safety violation. For instance, if we have one more phase of communication in the consensus process, the probability of safety violation becomes $O(\delta^3)$.

We use the two-phase protocol shown in Algorithm 4 in our implementation. We show the relationship between the committee size and ϵ in Figure 3. We also show some examples of the concrete probabilities in Table 2. In the table, Pando (x) denotes the setting where the committee size is xn. Here, we use xn for ease of understanding; this could simply be κ instead. Note the if the committee

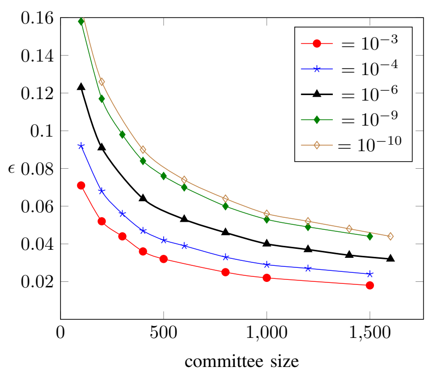


Figure 3: Committee size vs. ϵ to limit the probability of violating safety and liveness to 10^{-3}, 10^{-4}, 10^{-6}, 10^{-9}, and 10^{-10} respectively.

size is $O(n)$, the complexities of the protocol should be changed accordingly and the probability of achieving safety and liveness become $1 - \mathsf{negl}(n)$ instead of $1 - \mathsf{negl}(\kappa)$. The tables aim to show the relationship between ϵ and n. Namely, the goal is to show that given a desirable probability of safety and liveness violation (e.g., 10^{-4} so the protocol fails once every 10,000 epochs), *how much* resilience needs to be sacrificed for each n. As shown in Table 2, n does not have to be impractically large in our system. For example, for $n \geq 400$ and a committee size of more than 160 replicas, the resilience of the system is between $n > 4f$ to $n > 3f$. When n is greater, ϵ is closer to 0. Meanwhile, to achieve an even higher probability of safety and liveness (e.g., 10^{-9}) with the same n, ϵ has to be higher as well as shown in Figure 3.

Probability of liveness violation. We consider that liveness is violated if a transaction m is submitted to the system but is never delivered. Liveness can be violated in three scenarios: 1) No value is *a-delivered* in the consensus process; 2) Some value is *a-delivered* in the consensus process but no correct replica has received the corresponding proposal; 3) Some value is *a-delivered* in the consensus process, at least one correct replica has received the corresponding proposal, but the state transfer fails. As we show in Appendix A, the probability of the first scenario is δ^{2E}, where E is the number of *correct* epochs (the leader in atomic broadcast is correct) after m is submitted and after GST. Therefore, the failure probability of the consensus process is closer to 0 as the system is up and running. Accordingly, the probability of liveness violation of Pando becomes $p_1 + (1 - p_1)p_2$, where p_1 is the probability that no correct replica has received the transaction in the transmission process and p_2 is the probability that state transfer fails. As shown in Theorem 4, the probability of liveness violation is $O(\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}})$ for $\epsilon < 0.19$ or $O(\delta^2)$ for $\epsilon \in [0.19, 0.33)$.

7. Implementation and Evaluation

We implement Pando in Golang. We also implement Star in the same library and assess Narwhal using their open-source implementation [60]. We assess these two protocols as they have the same partial synchrony assumption as ours. Our codebase involves around 10,000 LOC for the protocol and about 1,000 LOC for evaluation. In our implementation, we use gRPC as the communication library. We use HMAC to realize the authenticated channel and use SHA256 as the underlying hash function. We use the Golang-based reed solomon code library[2] for erasure coding. We use the Golang-based VRF implementation[3] to instantiate the $\mathsf{ComProve}()$ and $\mathsf{ComVerify}()$ oracle. The VRF scheme we use achieves adaptive security under the random oracle assumption.

$n =$	100	200	300	400	500	1000
Pando (0.2)	0.193	0.138	0.125	0.104	0.092	0.068
Pando (0.4)	0.138	0.104	0.089	0.074	0.068	0.047
Pando (0.6)	0.125	0.089	0.073	0.063	0.056	0.039
Pando (0.8)	0.104	0.074	0.063	0.053	0.047	0.033

TABLE 2: The value of ϵ for the system to achieve safety and liveness with a probability of at least $1-10^{-4}$. The system requires $f \in [0, \frac{1}{5}n)$, $f \in [\frac{1}{5}n, \frac{1}{4}n)$, and $f \in [\frac{1}{4}n, \frac{1}{3}n)$ for dark gray cells, gray cells, and white cells, respectively.

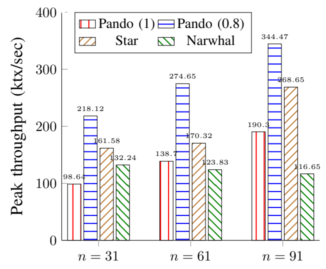
We evaluate the performance of our protocols on Amazon EC2 using up to 200 virtual machines (VMs) and up to 1,000 replicas. By default, we use *m5.xlarge* instances for our evaluation. The m5.xlarge instance has four vCPUs and 16GB memory. For one of the experiments, we use other types of instances. When assessing a setup with fewer than 100 replicas, we use each instance to run one replica. For a setup with more replicas, we use each instance to run five replicas. We deploy our protocols in the WAN setting, where replicas are evenly distributed in four different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland).

We conduct the experiments under different network sizes and batch sizes. We use n to denote the network size and b to denote the batch size. We repeat each experiment five times and report the average performance. The transaction size is 250 bytes.

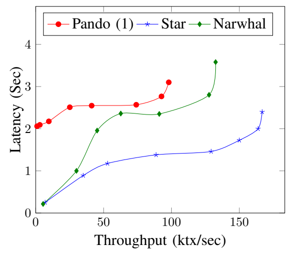
When evaluating Pando, we vary the committee sizes from $0.2n$ to n to assess the performance. Namely, when the committee size is n, the protocol is very close to a conventional protocol, e.g., Star. We intentionally do so to validate our results. We use the notation Pando (x) to denote the experiment with xn committee members. For example, Pando (0.2) uses $0.2n$ committee members and Pando (1) uses n committee members. Notably, for Pando (1), committee sampling is not needed anymore and the failure rate is not subjective to the failure rate δ. Our evaluation still involves the VRF evaluations to assess the overhead created due to committee sampling.

We summarize the required ϵ for our experiments to achieve a failure rate of 10^{-4} in Table 2. To achieve a failure rate of lower than 10^{-4}, Pando (0.6) needs to set $\epsilon = 0.125$ when $n = 100$, i.e., $f < 0.2n$. When n is larger, ϵ can be

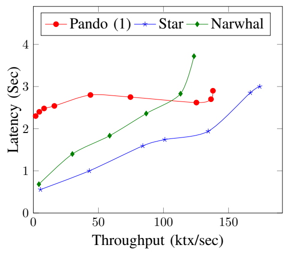
2. https://github.com/klauspost/reedsolomon
3. https://github.com/yoseplee/vrf



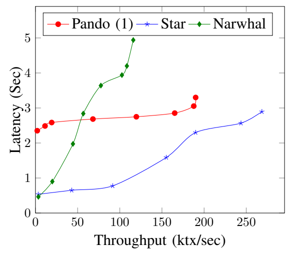
(a) Peak throughput of Star, Narwhal and Pando as f grows.



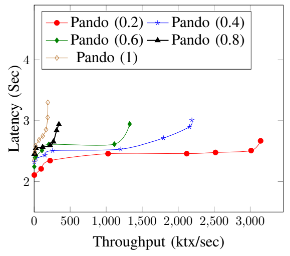
(b) Latency vs. throughput in WAN for $n = 31$.



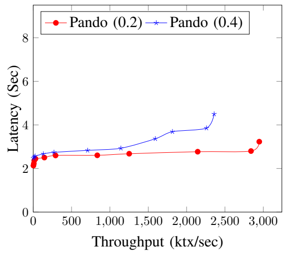
(c) Latency vs. throughput in WAN for $n = 61$.



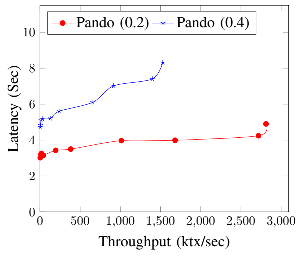
(d) Latency vs. throughput in WAN for $n = 91$.



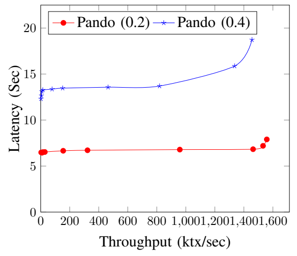
(e) Latency vs. throughput of Pando in WAN using different committee sizes for $n = 91$.



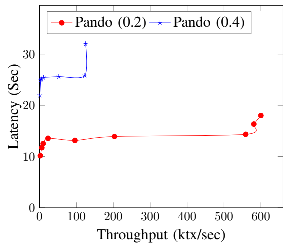
(f) Latency vs. throughput of Pando in WAN for $n = 100$.



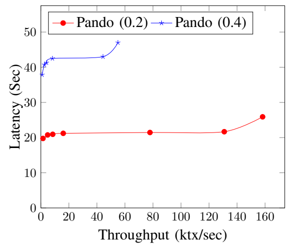
(g) Latency vs. throughput of Pando in WAN for $n = 200$.



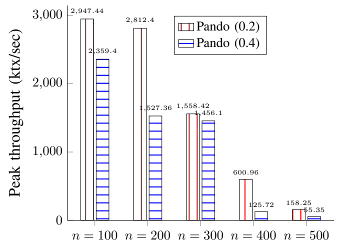
(h) Latency vs. throughput of Pando in WAN for $n = 300$.



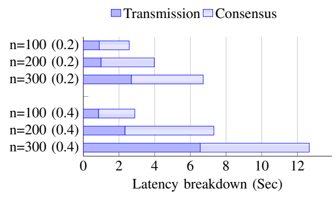
(i) Latency vs. throughput of Pando in WAN for $n = 400$.



(j) Latency vs. throughput of Pando in WAN for $n = 500$.



(k) Peak throughput of Pando as n grows.



(l) Latency breakdown of Pando when $b = 50,000$ for different n.

batch size	ktx/sec	CPU	bandwidth
100	2.49	62%	18.1MB/s
1,000	23.9	140%	18.8MB/s
5,000	117.54	186%	19.1MB/s
10,000	203.59	288%	19.4MB/s
15,000	344.47 (peak)	364%	19.7MB/s

(m) CPU and bandwidth usage of Pando (0.8) for $n = 91$ with different batch sizes. Maximum CPU usage is 400%.

committee	ktx/sec	CPU	bandwidth
0.2n	256.17	172%	5.9MB/s
0.4n	238.8	186%	10.7MB/s
0.6n	222.56	248%	15.2MB/s
0.8n	203.59	288%	19.4MB/s
n	190.30 (peak)	344%	24.3MB/s

(n) CPU and bandwidth usage of Pando for $n = 91$ and $b = 10,000$ for different committee sizes. Maximum CPU usage is 400%.

instance	vCPU	memory (GiB)	bandwidth (Gbps)	batch size	peak tps (ktx/sec)
m5.2	8	32	up to 10	-	-
m5n.2	8	32	up to 25	5,000	62.57
m5.4	16	64	up to 10	100	1.22
c5.4	16	32	up to 10	100	1.6

(o) Peak throughput of Pando for $n = 1,000$ using different instance types.

Figure 4: Performance of the protocols.

much lower. For instance, for n =1,000, Pando (0.4) can support $f < 0.29n$.

We summarize our evaluation results below.

- We were able to run Narwhal and Star using up to 100 replicas. Experiments beyond 100 replicas cannot be successfully launched on the VMs we used. We believe this is in part due to the low-end VMs (only 4 vCPUs). In contrast, we were able to run Pando using up to 500 replicas using the same low-end VMs and 1,000 replicas on VMs with only slightly better configuration.
- If we set the committee size of Pando as n, the performance of Pando is marginally lower than that of Narwhal and Star. If the committee size is lower than n, the performance of Pando starts to increase significantly due to lower communication and computational cost.
- By setting up a committee size of lower than n, Pando is significantly faster than existing protocols (but ϵ is also larger than the setup with a larger n). For example, for $n = 91$, the peak throughput of Pando (0.8) for $f = 30$ is 81.01% higher than Pando (1) and 28.22% higher than Star. Even for $n = 500$, Pando (0.4) still achieves a peak throughput of 158 ktx/sec.
- We conducted experiments for 1,000 replicas using different VMs. Our observation is that for a small-scale network, and CPU is usually the bottleneck of the system. In contrast, for the large-scale network, the network

bandwidth is the bottleneck.

Comparison of Pando, Narwhal, and Star. We first assess the peak throughput of Pando, Narwhal, and Star. We were not able to successfully run Narwhal and Star for a network beyond 100 replicas as we met a frequent "connection refused" error due to high communication costs. We believe this is mainly because our experiments are launched on low-end VMs with restricted network bandwidth. Accordingly, our comparison focuses on the setting for $n < 100$. We report the peak throughput of Pando (1), Pando (0.8), Star, and Narwhal in Figure 4a and latency vs. throughput for $n = 31, 61, 91$ in Figure 4b-4d. Our results show that the performance of Pando (1) is only marginally lower than Star and consistently higher than Narwhal. This is expected as Pando (1) has a committee size of n, so the communication and computational costs are almost identical to conventional protocols. Compared to Star, Pando (1) uses CBC instead of wCBC for the transmission process so the overhead is slightly higher. Additionally, Pando involves more computation due to VRF, so the performance is lower.

Pando (0.8) already consistently outperforms other protocols. For example, the peak throughput of Pando (0.8) for $n = 91$ is 81.01% higher than Pando (1) and 28.22% higher than Star. The improvement is caused by both lower communication and lower computation. Namely, the κ term for the communication becomes more insignificant as n grows.

Pando with different committee sizes. We assess latency vs. throughput for Pando for $n = 91$ by varying the committee size as $0.2n$ to n. As shown in Figure 4e, the performance of Pando is higher when the committee size is smaller. This is expected as having a small committee size will lower both communication and computational costs. The cost is that for a network of 91 replicas, ϵ has to be larger for smaller committee sizes, as summarized in Table 2.

Analysis of CPU and bandwidth usage. To understand why Pando starts to outperform existing protocols even with a committee of $0.8n$ replicas, we further assess the CPU and bandwidth usage of Pando for $n = 91$. In Figure 4m, we evaluate Pando (0.8) for different batch sizes until it achieves the peak throughput. It can be seen that the CPU usage and bandwidth usage grow as b grows. When CPU is fully used, the throughput does not grow anymore. Additionally, in Figure 4n, we fix the batch size as $10,000$ and vary the size of the committee. Among these experiments, Pando (n) is the only instance that achieves its peak throughput, in which case the CPU usage is maximized. For other cases, as the committee size is smaller, the CPU usage and bandwidth usage are also lower and the protocol achieves its peak throughput using an even larger batch size.

Latency vs. throughput. We assess latency vs. throughput of Pando for $n = 100, 200, 300, 400, 500$. For these scalability tests, we run five replicas on each VM. We choose $0.2n$ and $0.4n$ as the committee sizes and report the results in Figure 4f-4j. In general, the performance degrades as n grows. This is expected and similar results have been reported in all prior works. For a committee size of $0.4n$, all of our experiments are completed within 50 seconds (the highest occurs when $n = 500$). If we choose a committee size of $0.2n$, the experiments are completed within 30 seconds. For $n = 200$, the latency and peak throughput of Pando (0.2) are 4.9 seconds and 2,812 ktx/sec, respectively. This result is achieved with a batch size of around 80,000. As there are 200 replicas in total, 16,000 ktx are proposed so such a throughput is thus expected.

Scalability and latency breakdown. We report the peak throughput of Pando for $n = 100$ to 500 in Figure 4k. The throughput degrades significantly as n grows. We believe this is mainly because of the high communication cost and we started to meet the error of "connection refused" for $n > 300$. To further assess the results, we report the latency breakdown of the transmission process and the consensus process in Figure 4l. An interesting finding is that when n is large enough (in our case $n \geq 100$), the latency of the consensus process is even higher than the transmission process. This is mainly because the size of the certificate is very large as we instantiate each QC using a set of signatures. We believe this overhead can be further reduced using approaches such as aggregate signatures.

Experiments using 1,000 replicas. We conducted experiments using 1,000 replicas and were not able to obtain any throughput using the same $m5.xlarge$ VMs. We thus used different types of VMs to run the protocol. As summarized in Figure 4o, unlike small-scale experiments in which the CPU is usually the bottleneck, the network bandwidth is the bottleneck of the system for our 1,000-replica experiments. For VMs with higher network bandwidth (e.g., $m5n.2xlarge$ instance with 8 vCPU, 32GB memory, and up to 25 Gbps bandwidth), we were able to launch the experiments and Pando achieves a throughput of up to 62.57 ktx/sec. For VM with better configuration but lower network bandwidth (e.g., $c5.4xlarge$ instance with 16 vCPU, 32GB memory, and up to 10 Gbps bandwidth), Pando only achieves a throughput of 1.6 ktx/sec, as we were not able to run the experiments with a larger batch size (again due to the "connection refused" errors).

8. Conclusion

We present Pando, the first practical and scalable BFT from committee sampling. To this end, we have provided new communication-efficient and computation-efficient building blocks for BFT, including block transmission, atomic broadcast, and state transfer—all of which are of independent interest.

References

[1] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *EuroSys*, 2018.

[2] E. Buchman, "Tendermint: byzantine fault tolerance in the age of blockchains," 2017.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[4] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *CCS*. ACM, 2016, pp. 17–30.

[5] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: A fast blockchain protocol via full sharding," in *CCS*, 2018, pp. 931–948.

[6] S. Rahnama, S. Gupta, R. Sogani, D. Krishnan, and M. Sadoghi, "Ringbft: Resilient consensus over sharded ring topology," in *EDBT*, 2022, pp. 2:298–2:311.

[7] Y. Amir, C. Danilov, J. Kirsch, J. Lane, D. Dolev, C. Nita-Rotaru, J. Olsen, and D. Zage, "Scaling byzantine fault-tolerant replication to wide area networks," in *DSN*. IEEE, 2006, pp. 105–114.

[8] T. Crain, C. Natoli, and V. Gramoli, "Red belly: A secure, fair and scalable open blockchain," in *Symposium on Security*. IEEE, 2021, pp. 466–483.

[9] C. Stathakopoulou, T. David, M. Pavlovic, and M. Vukolic, "[solution] mir-bft: Scalable and robust BFT for decentralized networks," *J. Syst. Res.*, vol. 2, no. 1, 2022.

[10] C. Stathakopoulou, M. Pavlovic, and M. Vukolić, "State-machine replication scalability made simple (extended version)," in *Eurosys*, 2022.

[11] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Eurosys*, 2022, pp. 34–50.

[12] N. Giridharan, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Bullshark: DAG BFT protocols made practical," in *CCS*, 2022.

[13] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," in *CCS*, 2022.

[14] S. Duan, H. Zhang, X. Sui, B. Huang, C. Mu, G. Di, and X. Wang, "Dashing and star: Byzantine fault tolerance from weak certificates," in *Eurosys*, 2024.

[15] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-NG: Fast asynchronous bft consensus with throughput-oblivious latency," in *CCS*, 2022, pp. 1187–1201.

[16] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[17] E. Boyle, R. Cohen, and A. Goel, "Breaking the o($\sqrt{}$ n)-bit barrier: Byzantine agreement with polylog bits per party," in *PODC*, 2021, pp. 319–330.

[18] V. King and J. Saia, "Breaking the $o(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary," *Journal of the ACM (JACM)*, vol. 58, no. 4, p. 18, 2011.

[19] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of byzantine agreement, revisited," in *PODC*, 2019, pp. 317–326.

[20] T.-H. H. Chan, R. Pass, and E. Shi, "Sublinear-round byzantine agreement under corrupt majority," in *PKC*, 2020, pp. 246–265.

[21] G. Tsimos, J. Loss, and C. Papamanthou, "Gossiping for communication-efficient broadcast," in *CRYPTO*, 2022.

[22] E. Blum, J. Katz, C.-D. Liu-Zhang, and J. Loss, "Asynchronous byzantine agreement with subquadratic communication," in *TCC*. Springer, 2020, pp. 353–380.

[23] A. Bhangale, C.-D. Liu-Zhang, J. Loss, and K. Nayak, "Efficient adaptively-secure byzantine agreement for long messages," in *Asiacrypt*. Springer, 2022, pp. 504–525.

[24] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *TOCS*, vol. 20, no. 4, pp. 398–461, 2002.

[25] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*, 2016, pp. 31–42.

[26] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *CCS*, 1994, pp. 68–80.

[27] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[28] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *PODC*, 2019.

[29] X. Wang, S. Duan, J. Clavin, and H. Zhang, "Bft in blockchains: From protocols to use cases," *ACM Computing Surveys (CSUR)*, 2022.

[30] J.-P. Bahsoun, R. Guerraoui, and A. Shoker, "Making BFT protocols really adaptive," in *IPDPS*. IEEE, 2015, pp. 904–913.

[31] J. Li and D. Maziéres, "Beyond one-third faulty replicas in byzantine fault tolerant systems." in *NSDI*, 2007.

[32] S. Duan, H. Meling, S. Peisert, and H. Zhang, "BChain: Byzantine replication with high throughput and embedded reconfiguration," in *OPODIS*, 2014, pp. 91–106.

[33] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *DSN*. IEEE, 2019, pp. 568–580.

[34] S. Duan and H. Zhang, "Foundations of dynamic bft," in *Security and Privacy (SP)*, 2022, pp. 1317–1334.

[35] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," *DSN*, 2022.

[36] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," in *FC*, 2022, p. 296–315.

[37] N. Giridharan, F. Suri-Payer, M. Ding, H. Howard, I. Abraham, and N. Crooks, "Beegees: Stayin' alive in chained BFT," in *PODC*, 2023, pp. 233–243.

[38] X. Sui, S. Duan, and H. Zhang, "BG: A modular treatment of BFT consensus," *TIFS*, 2024.

[39] S. Duan, K. Levitt, H. Meling, S. Peisert, and H. Zhang, "ByzID: Byzantine fault tolerance from intrusion detection," in *SRDS*. IEEE, 2014, pp. 253–264.

[40] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "Damysus: streamlined bft consensus leveraging trusted components," in *Eurosys*, 2022, pp. 1–16.

[41] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *SOSP*, 2021, pp. 35–48.

[42] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[43] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[44] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," *arXiv preprint arXiv:2003.03052*, 2020.

[45] D. Larimer, "Delegated proof-of-stake (dpos)," 2014.

[46] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure multi-party computation," in *STOC*, 1996, pp. 639–648.

[47] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin, "Efficient multiparty computations secure against an adaptive adversary," in *Eurocrypt*. Springer, 1999, pp. 311–326.

[48] C. Liu, S. Duan, and H. Zhang, "Epic: Efficient asynchronous bft with adaptive security," in *DSN*, 2020.

[49] H. Zhang, C. Liu, and S. Duan, "How to achieve adaptive security for asynchronous bft?" *UPDC*, vol. 169, pp. 252–268, 2022.

[50] J. Sousa, E. Alchieri, and A. Bessani, "State machine replication for the masses with bft-smart," in *DSN*, 2014, pp. 355–362.

[51] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *JACM*, vol. 32, no. 2, pp. 374–382, 1985.

[52] H. Zhang and S. Duan, "PACE: Fully parallelizable bft from reproposable byzantine agreement," in *CCS*, 2022.

[53] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols." in *CCS*, 2020.

[54] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous bft closer to practice," *NDSS*, 2022.

[55] S. Duan, M. K. Reiter, and H. Zhang, "BEAT: Asynchronous bft made practical," in *CCS*. ACM, 2018, pp. 2028–2041.

[56] S. Duan, X. Wang, and H. Zhang, "Practical signature-free asynchronous common subset in constant time," in *CCS*, 2023.

[57] H. Zhang, S. Duan, B. Zhao, and L. Zhu, "Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft," in *Usenix Security*, 2023.

[58] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC*. ACM, 2021, pp. 165–175.

[59] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*. Springer, 2001, pp. 524–541.

[60] Narwhal and Tusk, "Narwhal and tusk," 2021. [Online]. Available: https://github.com/MystenLabs/narwhal

Appendix A. Proof of Correctness

A.1. The Transmission Process

(Chernoff Upper Tail Bound). Suppose $\{X_n\}$ is the independent $\{0,1\}$-random variables, and $X = \sum_i X_i$. Then for any $\tau > 0$:

$$\Pr\left(X \geq (1+\tau)E(X)\right) \leq \exp\left(-\frac{\tau \cdot \min\{\tau, 1\} \cdot E(X)}{3}\right)$$

Lemma 1. *Let $\alpha = \frac{1}{3} - \epsilon$ be the fraction of faulty replicas in the system and ϵ is a small constant where $0 < \epsilon \leq \frac{1}{3}$, δ be the desired failure probability. If the number of the replicas in the committee is greater than $\frac{3\alpha}{\epsilon^2}\ln\frac{1}{\delta} = O(\kappa)$, then with probability $1 - \mathsf{negl}(\kappa)$, the number of faulty replicas in the committee is less than $t = \kappa/3$ and the number of correct replicas in the committee is more than $2\kappa/3$.*

Proof. We model the committee election process as a c-times independent and repeated experiment, where c is the size of the committee; in the one-time experiment, a determinate replica is chosen randomly to be a committee member. This is equivalent to the process in which each replica calls $\mathsf{ComProve}()$ to check whether it is a committee member. Suppose P_i be the determinate committee member chosen in the i-th experiment, it is either correct or corrupt. Let the random variable X_i be 1 if P_i is faulty and X_i be 0 otherwise. Since n is sufficiently large, a faulty replica is chosen for the committee in a single experiment with a fixed probability α, so $Pr\left(X_i = 1\right) = \alpha$, for each $i = 1, 2, \cdots, c$, as shown in Table 3.

x	1	0
$Pr(X_i = x)$	α	$1-\alpha$

TABLE 3: Distribution of random variable X_i.

Let the random variable Y such that $Y = X_1 + \cdots + X_c$. Then Y represents the total number of faulty replicas chosen in the c-times independent and repeated experiments. Based on the above analysis and probability theory, we have $E(Y) = \alpha c$. According to the Chernoff Bound, we have

$$\begin{aligned}\Pr\left(Y \geq \frac{c}{3}\right) &= \Pr\left(Y \geq (\alpha+\epsilon)c\right)\\ &= \Pr\left(Y \geq (1+\frac{\epsilon}{\alpha})E(Y)\right)\\ &\leq \exp\{-\frac{\epsilon^2 E(Y)}{3\alpha^2}\}\\ &= \exp\{-\frac{c\epsilon^2}{3\alpha}\}\\ &\leq \delta \quad (\text{since} \quad c \geq \frac{3\alpha}{\epsilon^2}\log\frac{1}{\delta})\end{aligned}$$

The failure probability of the protocol δ is a negligible function in some statistical security parameters. As a special case, assuming that ϵ is a arbitrarily small positive constant, $0 < \epsilon < \frac{1}{3}$ and the mining difficulty parameter is $p_{mine} = \frac{3\alpha}{\epsilon^2 n}\ln\frac{1}{\delta}$, then $\delta = e^{-\omega(\log\kappa)}$ would be a negligible function. The lemma thus holds. ☐

Corollary 1. *Let α^* be the fraction of correct replicas in the system that holds some value v. If we sample a committee of $\frac{3\alpha}{\epsilon^2}\ln\frac{1}{\delta} = O(\kappa)$ size, $\alpha^*\kappa$ committee members hold value v with probability $1 - \mathsf{negl}(\kappa)$.*

Lemma 2. *In the transmission process, if P_i receives $\kappa - t$ signatures from committee $C^i_{t,e}$ for (e, h, i), then with probability $1 - \mathsf{negl}(\kappa)$, at least $f+1$ correct replicas in the system have received the proposed message M from P_i and the hash of M is h.*

Proof. Towards a contradiction, we assume fewer than $f+1$ correct replicas have received M. Suppose at most $f < (1/3 - \epsilon)n$ correct replicas in the system have received the proposed message M from P_i, and the hash of M is h. After these correct replicas call $\mathsf{ComProve}()$, fewer than $(1/3 - \epsilon)\kappa$ replicas in $C^i_{t,e}$ have received M since $p_{mine} = \kappa/n$. According to Lemma 1, there are at most $\kappa/3$ faulty replicas in $C^i_{t,e}$ with probability $1-\delta$. If P_i receives $\kappa - t$ signatures from $C^i_{t,e}$ for (e, h, i), at least $\kappa - t = 2\kappa/3$ replicas in $C^i_{t,e}$ have received M. This leads to a contradiction as there are only κ replicas in the committee. The lemma thus holds. ☐

Corollary 2. *In epoch e of the consensus process, given that each committee has at most t faulty replicas, the following holds: 1) if a correct replica receives $\kappa - t$ (PREPARE) messages with hash h from $C^2_{c,le}$, at least $f+1$ correct replicas in the system have received the (PROPOSE) message where the proposed block b satisfies $hash(b) = h$. 2) if a correct replica receives $\kappa - t$ (COMMIT) messages with hash*

*h from $C^3_{c,le}$, at least $f+1$ correct replicas in the system have received κ (*PREPARE*) messages from $C^2_{c,le}$ and set their $lockedQC$ to qc for $(2,h,le)$.*

Lemma 3. *Let $\alpha = \frac{1}{3} - \epsilon$ be the fraction of faulty replicas in the system, δ be the desired failure probability and ϵ be a small constant and $0 < \epsilon < \frac{1}{3}$. If the number of the replicas in the committee is greater than $\frac{3\alpha}{\epsilon^2} \ln \frac{1}{\delta} = O(\kappa)$, then with probability $1 - \frac{2+3\epsilon}{1-3\epsilon} \cdot \delta^{\frac{3-9\epsilon}{\epsilon}}$, there exists at least one correct replica in the committee.*

Proof. We bound the probability that there exists at most one correct replica in each committee. Since n is sufficiently large, the probability that one faulty replica be elected as a committee member is $\alpha = \frac{1}{3} - \epsilon$ (correspondingly, the probability that one correct replica is elected as a committee member is $1-\alpha = \frac{2}{3}+\epsilon$). Let c the the size of the committee. Then the probability that no more than one correct replica is elected as a committee member is:

$$\begin{aligned}
&\left(\frac{1}{3}-\epsilon\right)^c + c \cdot \left(\frac{2}{3}+\epsilon\right) \cdot \left(\frac{1}{3}-\epsilon\right)^{c-1} \\
=&\left(\frac{1}{3}(1-3\epsilon)\right)^c + c \cdot \frac{2+3\epsilon}{1-3\epsilon} \cdot \left(\frac{1}{3}(1-3\epsilon)\right)^c \\
=&\frac{1}{3^c} \cdot (1-3\epsilon)^c + \frac{c}{3^c} \cdot \frac{2+3\epsilon}{1-3\epsilon} \cdot (1-3\epsilon)^c \\
\le&\frac{2c}{3^c} \cdot \frac{2+3\epsilon}{1-3\epsilon} \cdot (1-3\epsilon)^c \\
\le&\frac{2+3\epsilon}{1-3\epsilon} \cdot (1-3\epsilon)^{\frac{3\alpha}{\epsilon^2}\ln\frac{1}{\delta}} \quad (\text{since} \quad c = \frac{3\alpha}{\epsilon^2}\ln\frac{1}{\delta}) \\
\le&\frac{2+3\epsilon}{1-3\epsilon} \cdot \exp\left(-\frac{9\alpha}{\epsilon}\ln\frac{1}{\delta}\right) \\
=&\frac{2+3\epsilon}{1-3\epsilon} \cdot \delta^{\frac{3-9\epsilon}{\epsilon}} \\
=&O\left(\delta^{\frac{3-9\epsilon}{\epsilon}}\right)
\end{aligned}$$

□

Lemma 4. *Let $\alpha = \frac{1}{3} - \epsilon$ be the fraction of faulty replicas in the system, δ be the desired failure probability, and ϵ be a small constant where $0 < \epsilon < \frac{1}{3}$. If the number of the replicas in the committee is greater than $\frac{3\alpha}{\epsilon^2} \ln \frac{1}{\delta} = O(\kappa)$, then with probability $1 - \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$, there exist at least $t+1$ correct replicas in the committee where $t = \kappa/3$.*

Proof. We bound the probability that there are no more than $\kappa/3+1$ correct replicas in the committee. Since n is sufficiently large, the probability that one faulty replica is elected as a committee member is $\alpha = \frac{1}{3} - \epsilon$ (correspondingly, the probability that one correct replica is elected as a committee member is $1-\alpha = \frac{2}{3}+\epsilon$). Let c be the committee size. Then the probability that there are no more than $c/3+1$ correct replicas in the committee is:

$$\begin{aligned}
\Pr\left(Y \ge \frac{2c}{3}\right) &= \Pr\left(Y \ge (\alpha + \frac{1}{3} + \epsilon)c\right) \\
&= \Pr\left(Y \ge (1 + \frac{\frac{1}{3}+\epsilon}{\alpha})E(Y)\right) \\
&\le \exp\{-\frac{(\frac{1}{3}+\epsilon)E(Y)}{3\alpha}\} \\
&= \exp\{-\frac{(\frac{1}{3}+\epsilon)c}{3}\} \\
&\le \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}} \quad (\text{since} \quad c \ge \frac{3\alpha}{\epsilon^2}\log\frac{1}{\delta})
\end{aligned}$$

□

Corollary 3. *In the transmission process, if P_i receives $\kappa - t$ signatures from committee $C^i_{t,e}$ for the tuple (e,h,i), the probability that none of correct committee members have received M is $\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$.*

Lemma 5. *In the transmission process for any epoch e, if a QC qc_j is formed where P_j is the sender, with the probability of $1 - \textsf{negl}(\kappa)$, at least $f+1$ correct replicas have received the proposal from P_j.*

Proof. The probability that $t+1$ correct committee members in $C^j_{t,e}$ have received the proposal from P_j is the same as the fact that there exist fewer than $\kappa - t$ correct replicas in the committee. According to Lemma 1, the probability is $1-\delta$ and δ is a negligible function. Then following an argument similar to that for Lemma 2, this lemma holds. □

Lemma 6. *Assuming that at least $f+1$ correct replicas have received a proposal from P_j, with the probability of $1 - \textsf{negl}(\kappa)$, the state transfer fails such that some correct replicas fail to receive the proposal from $C^j_{s,e}$.*

Proof. State transfer fails if the committee $C^j_{s,e}$ does not have any correct replica that has previously received the proposal from P_j. The probability is the same as that there are fewer than $t+1$ correct replicas in $C^j_{s,e}$, i.e., $\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$ by Lemma 4, a negligible function. □

A.2. The Consensus Process

Lemma 7. *If a correct replica P_i receives $\kappa - t$ matching messages from $C^3_{c,e}$ in epoch e, the (*PROPOSE*, b, e', qc_{high}) message by a correct leader in epoch $e' > e$ satisfies $height(qc_{high}) \ge e$. Additionally, at least $t+1$ correct replicas in $C^2_{c,e'}$ accept the (*PROPOSE*) message only if $height(qc_{high}) \ge e$.*

Proof. We know that P_i receives $\kappa - t$ matching messages from $C^3_{c,e}$. According to Corollary 2, at least $f+1$ correct replicas in the system have set their $lockedQC$ to a QC qc for $(2,h,e)$. Now, in any epoch $e' > e$, at the beginning of epoch e', a committee $C^1_{c,e}$ is sampled, and the committee members send their $lockedQC$ to the leader. According to Corollary 1, the leader will receive the QC and update its

qc_{high} accordingly. If the leader provides qc_{high}, the height of which is lower than e, at least $f+1$ correct replicas in the system have set their $lockedQC$ to qc. According to Corollary 1, at least $t+1$ correct replicas in $C^2_{c,e}$ will not accept the (PROPOSE) message. The lemma thus follows. □

Lemma 8. *If a correct replica P_i has received $\kappa - t$ matching (COMMIT) messages from $C^3_{c,e}$ in epoch e, in which the QC is for $(2,h,e)$, any correct replica eventually receives a QC for $(2,h,e)$.*

Proof. As P_i has received $\kappa - t$ matching (COMMIT) messages from $C^3_{c,e}$ for $(2,h,e)$, at least $\kappa - 2t \geq t+1$ correct replicas have sent (COMMIT) messages. According to our protocol, every replica in $C^3_{c,e}$ that has not sent a (COMMIT) message will also send a (COMMIT) message after receiving $t+1$ matching messages. Therefore, P_j eventually receives $\kappa - t$ matching (COMMIT) messages and obtains a QC for $(2,h,e)$. □

Theorem 1 (Safety)**.** *Let the probability that each committee has more than t faulty replicas be δ and the probability that the hash function is not collision-resistant be 0. If a correct replica a-delivers a message m before a-delivering m', then with probability $1 - O(\delta^2)$, no correct replica a-delivers a message m' without first a-delivering m.*

Proof. As the input of each epoch is a set of QCs and correct replicas only *a-deliver* messages sequentially, no correct replica will *a-deliver* any value m that has already been *a-delivered*.

Now we assume that a correct replica P_i *a-delivers* m in epoch e_1 and *a-delivers* m' in epoch e_2 and $e_2 > e_1$. Another correct replica P_j *a-delivers* m in e'_1 and m' in e'_2 and $e'_2 < e'_1$. We prove the correctness by contradiction.

Without loss of generality, we assume $e_1 < e'_2$ (the correctness follows vice versa). We show that if P_i *a-delivers* m in epoch e_1, P_j also *a-delivers* m' in e_1, $m = m'$.

If P_i *a-delivers* m, there are two cases: Case 1) P_i has received $\kappa - t$ matching signatures for $(2,h,e_1)$ from C^3_{c,e_1} in epoch e_1, where h is the hash of m; Case 2) P_i has *a-delivered* some value in epoch $e' > e_1$ and then *a-delivers* m via the $\mathsf{ObtainMissing}()$ function. Similarly, if P_j *a-delivers* m', there are two cases: Case 3) P_j has received $\kappa - t$ matching signatures for $(2,h',e_1)$ from C^3_{c,e_1} in epoch e_1, where h' is the hash of m'; Case 4) P_j has *a-delivered* some value in epoch $e'' > e_1$ and then *a-delivers* m via the $\mathsf{ObtainMissing}()$ function. In the following, we show that in any combination of the two cases, $m = m'$.

Case-1: Case 1 (for P_i) and Case 3 (for P_j). As the committee C^3_{c,e_1} has κ replicas among which at most $\kappa/3$ replicas are faulty with probability $1 - \mathsf{negl}(\kappa)$, at least one correct replica has sent a signature for both $(2,h,e_1)$ and $(2,h',e_1)$, a contradiction. Additionally, according to the collision-resistance of the hash function, $m = m'$.

Probability of safety violation for Case-1: According to the definition, a correct replica in C^3_{c,e_1} will never send signatures for inconsistent values. P_i receives $\kappa - t$ matching messages for $(2,h,e_1)$ from C^3_{c,e_1}. Let the set of $\kappa - t$ replicas that send matching (COMMIT) messages be S_1. Meanwhile, P_j receives $\kappa - t$ matching messages for $(2,h',e_1)$ from C^3_{c,e_1}. Let the set of replicas that send $\kappa - t$ matching messages be S_2. According to the proof in Theorem 1, a safety violation occurs only when S_1 or S_2 has fewer than $\kappa - 2t$ correct replicas.

There are two sub-cases if safety is violated: 1) none of S_1 or S_2 has any correct replicas; 2) there is at least one correct replica P_k in S_1 and there is at least one correct replica P_ℓ in S_2 and $k \neq \ell$.

For sub-case 1 (Case-1-SC1), faulty committee members can already cause a safety violation. The probability SC1 occurs only if the C^3_{c,e_1} committee has fewer than $t+1$ correct replicas. By Lemma 4, the probability of safety violation of sub-case 1 is: $\Pr(Case\text{-}1\text{-}SC1) = \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$.

We now analyze sub-case 2 (Case-1-SC2). First, this case causes a safety violation only if there are fewer than $\kappa - t$ correct replicas so the probability is $p_1 = \delta$.

Second, we analyze the probability that sub-case 2 leads to a safety violation. Since P_k has sent a (COMMIT) message for $(2,h,e_1)$, it has previously received $\kappa - t$ matching (PREPARE) messages for $(1,h,e_1)$ from C^2_{c,e_1}. Let the set of replicas be S_3. Meanwhile, as P_ℓ has sent a (COMMIT) message for $(2,h',e_1)$, it has previously received $\kappa - t$ matching (PREPARE) messages for $(1,h',e_1)$ from C^2_{c,e_1}. Let the set of replicas be S_4. The probability that there does not exist a correct replica in $S_3 \cap S_4$ is the same as the probability that the C^2_{c,e_1} committee has fewer than $\kappa - t$ correct replicas, i.e., $p_2 = \delta$.

Put them together, the probability that sub-case 2 leads to a safety violation is: $\Pr(Case\text{-}1\text{-}SC2) \leq p_1 p_2 = \delta^2$.

The probability that Case-1 leads to a safety violation is then:

$$\begin{aligned}\Pr(Case\text{-}1) &= \Pr(Case\text{-}1\text{-}SC1) + \Pr(Case\text{-}1\text{-}SC2)\\ &\leq \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}} + \delta^2.\end{aligned}$$

Case-2: Case 1 (for P_i) and Case 4 (for P_j). If P_j *a-delivers* some value m'' in epoch $e'' > e_1$, m'' consists of proposals between the height of qc_{high} (in the (PROPOSE) message) and e''. We first show that the $height(qc_{high}) \geq e_1$. Then, we show that P_j will eventually receive a QC for epoch $e_1$in the $\mathsf{ObtainMissing}()$ function and then *a-deliver* m'. Finally, we show $m = m'$.

We begin with $height(qc_{high}) \geq e_1$. If P_i receives $\kappa - t$ matching (COMMIT) messages in epoch e_1, by Lemma 7, in the proposal of any epoch greater than e_1, at least $t+1$ correct replicas will not accept a (PROPOSE) message for $height(qc_{high}) < e_1$ with probability $1 - \mathsf{negl}(\kappa)$. Now, assume that when P_j *a-delivers* some value in epoch e'', the height of the qc_{high} in the (PROPOSE) message is lower than e_1. Therefore, at least $\kappa - t$ replicas in $C^2_{c,e''}$ have accepted the (PROPOSE) message and created a signature. This is a violation as at least $t+1$ correct replicas in $C^2_{c,e''}$ will not accept the message.

We now show that P_j eventually obtains a QC for $(2,h,e_1)$ for epoch e_1 in the $\mathsf{ObtainMissing}()$ function.

According to Lemma 8, P_j eventually obtains a QC for $(2, h, e_1)$. After that P_j has either received m' from the leader such that the hash of m' is h, or synchronized m' from other replicas.

According to the collision-resistance of the hash function, $m = m'$.

Probability of safety violation for Case-2: P_i receives $\kappa - t$ matching (COMMIT) messages from C^3_{c,e_1} in epoch e_1. Additionally, P_j *a-delivers* some value in epoch e'' and the height of the qc_{high} in the (PROPOSE) message is lower than e_1. At least $\kappa - t$ replicas in $C^2_{c,e''}$ have accepted the (PROPOSE) message with qc_{high}. Among them, fewer than $\kappa - 2t$ are correct. In a partially synchronous environment, the probability that this occurs is the same as the probability that $C^2_{c,e''}$ has fewer than $\kappa - t$ correct replicas, i.e., $p_1 = \delta$.

There are three sub-cases for C^3_{c,e_1} in epoch e_1: there are fewer than $t + 1$ correct replicas in C^3_{c,e_1}; there are at least $\kappa - t$ correct replicas in C^3_{c,e_1}; there are fewer than $\kappa - t$ correct replicas in C^3_{c,e_1}. The probability of the three sub-cases is $\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$ (by Lemma 4), $1 - \delta$, δ, respectively.

Sub-case 1 (Case-2-SC1) directly leads to a safety violation. By Lemma 4, the probability that there are fewer than $t + 1$ correct replicas in C^3_{c,e_1} is: $\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$. Meanwhile, any committee $C^*_{c,e}$ for $e_1 < e < e''$ has at least $\kappa - t$ correct replicas. The probability that each $C^*_{c,e}$ has at least $\kappa - t$ correct replicas is $1 - \delta$. Together with the case that there are fewer than $\kappa - t$ correct replicas in $C^2_{c,e''}$, the probability that SC1 leads to a safety violation is:

$$\begin{aligned}\Pr(Case\text{-}2\text{-}SC1) &< p_1 \cdot \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}} \\ &= \delta^{\frac{\frac{1}{9}}{\epsilon^2}}.\end{aligned}$$

For sub-case 2 (Case-2-SC2), P_i receives $\kappa - t$ matching (COMMIT) messages for $(2, h, e_1)$. Among them, at least $\kappa - 2t$ messages are sent by correct replicas. Any correct replica P_k in the $\kappa - 2t$ set has received $\kappa - t$ matching (PREPARE) messages for $(1, h, e_1)$. According to Corollary 2, at least $f + 1$ correct replicas receive $\kappa - t$ matching (PREPARE) messages for $(1, h, e_1)$. According to the protocol, these $f + 1$ correct replicas will not vote for qc_{high}. Therefore, sub-case 2 leads to a safety violation when 1) $C^2_{c,e''}$ has fewer than $\kappa - t$ correct replicas; 2) there are at least $\kappa - t$ correct replicas in C^3_{c,e_1}; 3) there are fewer than $\kappa - t$ correct replicas in C^2_{c,e_1}; 4) Any committee $C^*_{c,e}$ for $e_1 < e < e''$ has at least $\kappa - t$ correct replicas. The probability of 1), 2), 3) is δ, $1 - \delta$, δ, respectively. Therefore, the probability that sub-case 2 leads to a safety violation is:

$$\Pr(Case\text{-}2\text{-}SC2) < \delta(1 - \delta)\delta = \delta^2(1 - \delta) < \delta^2.$$

Sub-case 3 (Case-2-SC) leads to a safety violation when both sub-case 3 occurs and there are fewer than $\kappa - t$ correct replicas in $C^2_{c,e''}$. Meanwhile, any committee $C^*_{c,e}$ for $e_1 < e < e''$ has at least $\kappa - t$ correct replicas. Therefore,

$$\Pr(Case\text{-}2\text{-}SC3) < \delta^2.$$

To conclude, the probability that Case-2 leads to a safety violation is:

$$\begin{aligned}\Pr(Case\text{-}2) &= \Pr(Case\text{-}2\text{-}SC1) + \Pr(Case\text{-}2\text{-}SC2) \\ &\quad + \Pr(Case\text{-}2\text{-}SC3) \\ &< \delta^{\frac{\frac{1}{9}}{\epsilon^2}} + \delta^2 + \delta^2 = O(\delta^2).\end{aligned}$$

Case-3: Case 2 (for P_i) and Case 3 (for P_j). Correctness is similar to Case-1 (i.e.,case 1 for P_i and case 4 for P_j) and we omit the details here.

Probability of safety violation for Case-3: The analysis for this case is similar to that for Case-1 and we omit the details.

Case-4: Case 2 (for P_i) and Case 4 (for P_j). Both P_i and P_j *a-deliver* some value in epoch greater than e_1. Here, there are two sub-cases: 1) at least one correct replica P_ℓ has received $\kappa - t$ (COMMIT) messages in epoch e_1; 2) none of the correct replicas has received $\kappa - t$ (COMMIT) messages in epoch e_1. In the first sub-case, we know that the height of qc_{high} in the (PROPOSE) message by a correct leader for any epoch greater than e_1 must be greater than e_1 according to Lemma 7. In this case, since $e' > e_1$ and $e'' > e_1$, the *a-delivered* message will not consist of any value for epoch e_1. According to Lemma 8, both P_i and P_j eventually receive the same qc for epoch e_1. It is then not difficult to see that $m = m'$. In the second sub-case, the case is identical to case 1 for some correct replicas and case 3 for some correct replicas. It is then not difficult to see that $m = m'$.

As P_j *a-delivers* m in epoch e_1, $e_1 = e'_1$. We also know that P_j *a-delivers* m' in e'_2 and P_i *a-delivers* m' in epoch e_2. Therefore, $e'_2 < e_2$. Following a similar argument as above, we know that if P_j *a-delivers* m' in e'_2, P_i must have *a-delivered* m' in e'_2 as well, a contradiction with $e'_2 < e_2$.

Probability of safety violation for Case-4: The analysis for this case is similar to that for Case-2 and we omit the details.

□

Lemma 9. *In every epoch e, if at least one correct replica P_i receives $\kappa - t$ (COMMIT, $h, e, -$) messages with the same h, every correct replica P_j eventually receives $\kappa - t$ (COMMIT, $h, e, -$) messages.*

Proof. We assume that Δ is properly set up. If a correct replica P_i receives $\kappa - t$ (COMMIT, $h, e, -$) messages with the same h, the messages are sent from committee members in $C^3_{c,e}$. As the committee $C^3_{c,e}$ has at least $t+1$ correct replicas, all correct replicas will eventually receive $t + 1$ (COMMIT) messages with the same h and any correct replica that has not sent a (COMMIT) message will send one to all replicas. Therefore, every correct replica P_j eventually receives $\kappa - t$ (COMMIT, $h, e, -$) messages. □

Lemma 10. *In every epoch e, if at least one correct replica P_i receives $\kappa - t$ (COMMIT, $h, e, -$) messages with the same h, for the block b proposed by the leader (the hash of b is h and the QCs with the lowest epoch number in b is e'),* *at least one correct replica has already a-delivered some values in any epoch lower than e'.*

Proof. If at least one correct replica P_i receives $\kappa - t$ $(\text{COMMIT}, h, e, -)$ messages with the same h, at least $t+1$ replicas in $C^2_{c,e}$ have sent (PREPARE) messages with the same h, among which at least one is correct. According to the $\mathsf{IsValid}(b)$ function, every correct replica in $C^2_{c,e}$ sends a (PREPARE) message only if it has completed every epoch lower than e'. The lemma thus holds. □

Lemma 11. *If a correct replica P_i queries $\mathsf{ObtainMissing}(ce, le, m)$, the function eventually returns some $\boldsymbol{m}$.*

Proof. P_i iterates every $e \in [ce, le]$ and there are two cases: some QCs W_e has already been included in m; QCs are not included in m. For the first case, $\boldsymbol{m}[e]$ is set as W_e. We now focus on the second case. In this case, P_i has not completed epoch e, but the proposer (leader in epoch le) believes that epoch e has already been completed. Here, P_i simply waits for the proposal of epoch e, and we show that P_i eventually obtains the proposed block b. According to Lemma 10, at least one correct replica has completed epoch e. Furthermore, according to Lemma 9, P_i eventually receives $\kappa - t$ matching $(\text{COMMIT}, h, e, -)$ messages. Based on the hash value h, P_i is able to obtain the original proposal b (possibly synchronized from other replicas). □

Theorem 2 (Liveness). *Let the probability that each committee has more than t faulty replicas be δ. If a correct replica a-broadcasts a message m, then all correct replicas eventually a-deliver m with probability $1 - \delta^{2E}$, where E is an epoch number.*

Proof. If a correct replica P_i *a-broadcasts* a message m in epoch e, it has received $\kappa - t$ $(\text{COMMIT}, h, e, -)$ messages with the same h. According to Lemma 9, any correct replica eventually receives $\kappa - t$ $(\text{COMMIT}, h, e, -)$ messages with the same h. Furthermore, P_i either directly *a-delivers* some value or obtains some value from the $\mathsf{ObtainMissing}()$ function. According to Lemma 11, every correct replica eventually obtains some $\boldsymbol{m}$. The collision resistance of the hash function ensures that the value of every correct replica *a-delivers* is m.

Consider the case where the leader is correct and the leader proposes m in epoch e, liveness is violated only if none of $C^2_{c,e}$ and $C^3_{c,e}$ have at least $\kappa - t$ correct replicas. By Lemma 1, the probability of this case is δ^2.

According to the protocol, replicas will move to a new view if replicas do not *a-deliver* any value in epoch e. We also additionally require every correct leader to propose a value for epoch e even if it enters a new epoch $e' > e$. Without loss of generality, assuming that the correct leader proposes m in epoch 1 and every correct leader continues to propose m if m has not been *a-delivered* yet. After GAT, the probability that m is not *a-delivered* is therefore bounded by δ^{2E}, where E is the number of epochs after m was submitted and the leader in these epochs are correct. □

Appendix B. Complexity Analysis

We discuss the communication complexity of Pando.

Lemma 12. *The communication complexity of the transmission process is $O(Ln^2 + \kappa^2 n^2)$.*

Proof. The communication complexity of this process is bounded by the function $\mathsf{InitEpoch}(e)$, where the leader P_i sends a message $(\text{PROPOSAL}, e, M, qc_i)$ to all replicas. Each qc_i consists of κ digital signatures so the length is $O(\kappa^2)$. As there are n such instances, the communication complexity is shown as follows.

$$\sum_{i=1}^{n} O\left(n(L + \kappa^2)\right) = O(Ln^2 + \kappa^2 n^2)$$

□

Lemma 13. *The communication complexity of our atomic broadcast protocol is $O(|M|n + \kappa^2 n)$, where $|M|$ is the size of input. The communication complexity of the consensus process in Pando is $O(\kappa^2 n^2)$.*

Proof. In the (NEW-VIEW) phase, κ committee replicas send their $lockedQC$ to the leader. The size of $lockedQC$ is κ^2, so the communication of this phase is $O(\kappa^3)$.

In the (PROPOSE) phase, the leader broadcasts its proposal M and the evidence of its identity (VRF evaluation), and a QC to all replicas. The length of the VRF evaluation is $O(\kappa)$ and the length of the QC is $O(\kappa^2)$, so the communication is $O(|M|n + \kappa^2 n + \kappa^3)$.

In the (PREPARE) and (COMMIT) phases, κ replicas broadcast their signatures to n replicas, so the communication is $O(\kappa^2 n)$.

The communication complexity of the atomic broadcast protocol is thus $O(|M|n + \kappa^2 n + \kappa^3)$. As we consider $n > \kappa$, the complexity is $O(|M|n + \kappa^2 n)$.

Using the atomic broadcast protocol in the consensus process, the input consists of $n - f$ QCs and the length of each QC is $O(\kappa^2)$. Therefore, the communication complexity is thus $O(\kappa^2 n^2)$. □

Lemma 14. *The communication complexity of state transfer protocol is $O(L\kappa n^2 + \kappa^2 n^2)$.*

Proof. In the state transfer process, κ replicas are sampled for each j and each sampled replica sends a proposal and a VRF evaluation to all replicas. The communication complexity is shown as follows.

$$\sum_{i=1}^{n} O\left(\kappa n(L + \kappa)\right) = O(L\kappa n^2 + \kappa^2 n^2)$$

□

Appendix C. Safety and Liveness of Pando

Theorem 3 (Safety)**.** *Let the probability that each committee has more than t faulty replicas be δ. If a correct replica delivers a transaction tx before delivering tx', then no correct replica delivers a transaction tx' without first delivering tx with probability $1 - O(\delta^2)$.*

Proof. Safety of atomic broadcast (i.e., consensus process) ensures that any correct replica *a-delivers* a set of QCs in every epoch. For any qc_j, every correct replica obtains M, the hash of which is h according to Lemma 5. Every correct replica *a-delivers* the same set of transactions in O in every epoch. As correct replicas deliver transactions in O in the same deterministic order and correct replicas will not deliver the same transaction twice, the theorem thus holds.

By Theorem 3, the probability that safety is violated for Pando is the same as that for the atomic broadcast protocol. By Theorem 1, the probability is $O(\delta^2)$. □

Theorem 4 (Liveness)**.** *Let the probability that each committee has more than t faulty replicas be δ. If a transaction tx is submitted to all correct replicas, then all correct replicas eventually deliver tx with probability $1-O(\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}})$.*

Proof. If a transaction tx is submitted to all correct replicas, eventually in some epoch, tx will included in the proposal by at least one correct replica. As the network is eventually synchronous, the qc for the proposal containing transaction tx will eventually be received by all correct replicas. At least $n-f$ QCs will be *a-delivered* in the consensus process according to the liveness of atomic broadcast (Theorem 2), among which at least $f+1$ QCs are proposed by correct replicas. Therefore, it is not difficult to see that tx will be eventually *a-delivered* by correct replicas.

Liveness is violated under three cases: 1) No value is *a-delivered* in the consensus process; 2) Some value is *a-delivered* in the consensus process but no correct replica has received the corresponding proposal; 3) Some value is *a-delivered* in the consensus process, at least one correct replica has received the proposal, and the state transfer fails.

According to Theorem 2, the probability that no value is *a-delivered* in the consensus process is δ^{2E}. If the transaction m is *a-delivered*, liveness is then violated under two cases: 1) No correct replica has received the corresponding proposal; 2) At least one correct replica has received the corresponding proposal, and the state transfer fails.

Given each QC, the probability that no correct replicas have received the corresponding proposal is $p_1 = \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$ by Corollary 3. Additionally, by Lemma 6, the probability that state transfer fails is $p_2 = \delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}}$.

Therefore, the probability that liveness is violated for our protocol is: $p_1 + (1-p_1)p_2 = O(\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}})$. □