Pando: Extremely Scalable BFT Based on Committee Sampling

Xin Wang*§, Haochen Wang*, Haibin Zhang†✉, Sisi Duan*‡§✉
*Tsinghua University
†Yangtze Delta Region Institute of Tsinghua University, Zhejiang
‡Zhongguancun Laboratory, Shandong Institute of Blockchains
§State Key Laboratory of Cryptography and Digital Economy Security
wangxin87@tsinghua.edu.cn, whc20@mails.tsinghua.edu.cn, bchainzhang@aliyun.com, duansisi@tsinghua.edu.cn
✉ Corresponding authors

***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, particularly on low-end machines.**

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. 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.

I. 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, such as sharding-based BFT protocols that operate in a number of BFT shards [4], [5], [6], [7], [8] and using parallelism to improve the scalability [9], [10]. Most of these protocols, however, use an overly strong assumption, e.g., *each* shard does not have more than one-third or half faulty replicas. In recent years, de-coupling block transmission (that carries the bulk data) from the consensus on the order (that agrees on hashes or digital signatures) [11], [12], [13], [14], [15] has been shown to be promising in improving the performance and scalability of the system without introducing additional assumptions. However, while these protocols mark significant milestones for scalable BFT, they can support roughly a hundred replicas in the WAN environment on relatively low-end machines.

Accordingly, it is still an open problem to scale BFT to, say, 1,000 replicas.

The overhead of existing approaches, briefly. Since de-coupling block transmission from consensus is one of the most promising approaches to improve the scalability, we focus on this model in this work. By taking a deeper look at the bottleneck when the system further scales beyond a hundred replicas, the main bottlenecks are the communication overhead and the computational overhead. The communication becomes prohibitively high as n grows, mainly because all replicas need to communicate *directly* with each other. Such a paradigm is also known as an n-to-n communication. Meanwhile, existing approaches use threshold signatures (or a set of $O(n)$ signatures) for quorum certificates (QCs) [11], [14], [13], [15] 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.

BFT from committee sampling. To circumvent the bottleneck on communication and computational overhead, 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 and *conveys* some information to all replicas. The nature of the κ-to-n communication pattern makes these protocols more communication-efficient. Such protocols have been studied in both the synchronous setting [17], [18], [19], [20], [21] and the asynchronous setting [22], [23]. For these committee-based protocols, a possible workflow is to sample a committee, have

[1]Data source (accessed in Apr 2025): https://www.beaconcha.in/

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230273
www.ndss-symposium.org

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 [16]	$f < n/3$	$O(Ln^2 + \kappa n^3)$	$O(\kappa n^3)$	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 I: Communication complexity of BFT systems that decouple block transmission from consensus on the order. L is the size of the input (i.e., a block proposal) of every replica, κ is the length of the cryptographic security parameter, and ϵ is a small constant that 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.

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. 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[2] yet.

The challenge for committee sampling-based BFT. Committee sampling-based protocols all suffer from a limitation that they can only achieve *near-optimal resilience*. Namely, in a partially synchronous network, optimal resilience requires $f < n/3$ (i.e., $n \geq 3f + 1$), where f is the number of faulty replicas. However, committee sampling-based approaches can only achieve $f < (1/3 - \epsilon)n$, where ϵ is a small constant and $\epsilon \in (0, 1/3)$. One may want to make ϵ close to 0 to claim near-optimal resilience.

In practice, however, the committee size has to be extremely large for ϵ to be close to 0. For instance, Algorand [25], [26] is the first practical Byzantine agreement (BA) protocol from committee sampling. Algorand mentions that to limit the probability of safety and liveness violation (failure rate for short) to 10^{-9}, and to set $\epsilon = 0.12$ (i.e., the system has 80% correct replicas), a committee size of 2,000 replicas [25, Figure 3] is needed. This applies to *all* committee sampling-based protocols. Unfortunately, also as mentioned above, most BFT protocols can only achieve decent performance with fewer than a hundred replicas. For the case of Algorand, it can deliver 2 MBytes of transactions in 22 seconds, using a committee size of 2,000 replicas.

Accordingly, the research question of our work is:

Can we build a scalable BFT protocol from committee sampling by supporting small committee sizes while achieving a practically low failure rate?

[2]Atomic broadcast is only syntactically different from BFT [24]. Informally, atomic broadcast does not involve the role of the *clients*.

Pando in a nutshell. 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 [27]. To be specific, we consider a weakly adaptive adversary, the most commonly used assumption in the literature [19], [20], [21]. Briefly speaking, Pando extends the framework that decouples block transmission from the agreement on the block order to committee sampling-based ones to lower both the communication and computational overhead. As summarized in Table I, our work reduces the communication complexity of the *transmission* and *consensus* processes, the crucial building blocks in the framework that decouples block transmission from consensus. 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.

To answer the research question on the practical size of the committee, we propose a partially synchronous atomic broadcast protocol for the consensus process, a crucial process in the framework that decouples block transmission from consensus. In Pando, we exploit the idea of sampling multiple committees in the agreement on each block — a feature that is naturally needed for adaptively secure BFT from committee sampling — to use a probabilistic analysis to significantly lower the committee size. Namely, to achieve the same 10^{-9} failure rate, Pando only requires a committee size of 200, instead of 2,000 in previous work! Such a unique feature makes it possible to use committee sampling-based approach in practical settings.

Our theoretical contributions include building a committee sampling-based consistent broadcast [28] and atomic broadcast protocol. Additionally, compared to prior committee-based approaches, our approach uses 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 roughly the same (except with a small probability) as that in the entire system. We believe our results are beneficial for protocols beyond committee sampling ones, e.g., sharding-based BFT [29], [30].

Our contributions. We make the following contributions.

- We propose Pando, an adaptively secure and scalable BFT protocol. Compared to prior work that also decouples 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.

II. System Model and Building Blocks

Threat model and assumptions. 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, where ϵ is a small constant and $0 < \epsilon < 1/3$. Byzantine faulty replicas may fail arbitrarily, including software bugs, hardware errors, and adversarial attacks. Non-faulty replicas are called *correct replicas*. 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]: An honest replica P_i sends a message to multiple replicas; the adversary can corrupt P_i either before or after it sends the message to all receivers.

We consider a partially synchronous network where there exists a Global Stabilization Time (GST), after which the network becomes synchronous.

We follow prior works [31], [32], [33], [24] 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 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 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 $\lambda = \kappa$ and with $1 - \mathsf{negl}(\kappa)$ probability, each committee has no more than t faulty replicas. Slightly abusing the notation, we also use the term QC in the committee to denote $\lambda - t$ signatures from committee members.

Definitions of BFT and ABC. 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.

A. Building Blocks

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 ComProve()/ComVerify() oracle. We follow prior works [19], [20], [21] and define a ComProve()/ComVerify() oracle as a committee sampling function. We present

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

public parameters: let p_{mine} be the mining probability
local parameters: let $call_i \leftarrow \perp$ for any $i \in [n]$
function ComProve(m, i)
  if $call_i = \perp$ **then**
    let $b \leftarrow 1$ with probability p_{mine} or 0 otherwise
    $call_i \leftarrow b$
  return $call_i$
function ComVerify(m, j)
  return $call_j$

in Algorithm 1 the functionality of ComProve() and ComVerify() [21]. ComProve() is parameterized by the total number of replicas and a *mining* probability p_{mine}. It is specified by two functionalities: ComProve() and ComVerify(). In particular, a replica P_i can query ComProve(m, i) to check whether it is an eligible member of the committee, where m is the designated input. If m is changed, ComProve(m, i) will return a different value (i.e., $call_i$ is related to m). Following the prior work [21], m is the input to the VRF function in practice. The query of the ComProve() function is also called a *mining* attempt. Upon receiving a mining attempt for the first time, 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, ComVerify(m, i) returns the same answer for all future identical queries.

We use the notation C_x^y to denote the committees, where the subscript x specifies the corresponding process (e.g., transmission, consensus) 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 ComProve() and ComVerify() functions as follows: replica P_i queries ComProve$(t||e||j, i)$ to learn whether it is a committee member where $||$ denotes concatenation; after P_i queries the ComProve() function, any replica P_k queries ComVerify$(t||e||j, i)$ to verify whether P_i belongs to $C_{t,e}^j$.

We instantiate ComProve() and 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 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 ComVerify$(t||e||j, i)$, the function returns true if the VRF evaluation is lower than D.

III. Motivation

A. Review of Existing De-coupling Approaches

Existing works that decouple block transmission from consensus [11], [13], [12], [14], [15], [34] usually involve three processes:

- A transmission process where each replica sends a proposal 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 will never be reversed);
- After an agreement is reached, replicas that do not hold the proposals run the state transfer process to obtain the proposals from other replicas. Since each replica holds the hash of each proposal from the QCs, the collision-resistance property of the hash function ensures that all replicas obtain the same proposal.

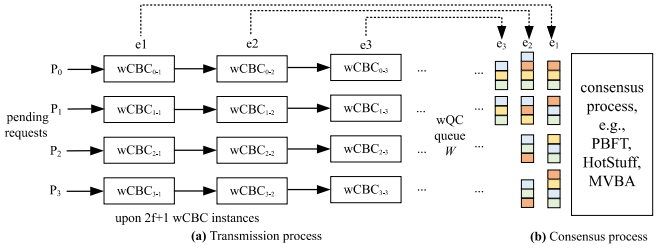


Fig. 1: The Star framework [14].

As an example, we show the Star framework in Fig. 1. 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 sends 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.

Existing works use different protocols in different processes. Narwhal [11] and Bullshark [12] use the directed acyclic graph (DAG) data structure and CBC in the transmission process. Dumbo-NG [13] uses a pipeline mode of CBC that is slightly different from that in Star. In the consensus process, Narwhal uses HotStuff [32], and Bullshark employs a partially synchronous variant of DAG-Rider [35].

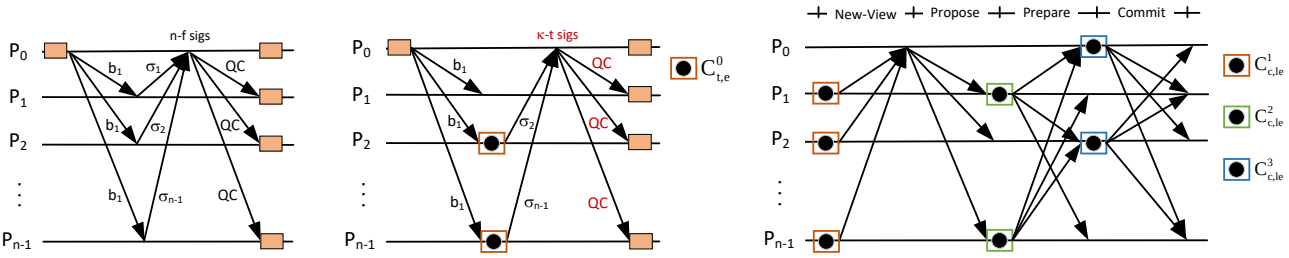
By default, in the state transfer process, each replica requests the missing proposals from all other replicas. Dumbo-NG uses erasure coding to achieve a more communication-efficient approach (called "retrieval" in the paper). All these state transfer approaches involve all-to-all communication and achieve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

The feature that decouples 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 a throughput of 256 ktx/sec, significantly higher than conventional protocols.

B. The Scalability Bottlenecks

If we further scale the 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 the transmission process, the all-to-all communication for block proposal (due to n parallel CBC instances) seems to be



(a) Conventional consistent broadcast (CBC) protocol.

(b) Our scalable CBC approach.

(c) Our scalable atomic broadcast protocol.

Fig. 2: Overview of our approach.

unavoidable. However, collecting $O(n)$ signatures and including them in the proposal may again consume high network bandwidth, especially when n is large. 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, each signature has $O(\kappa + n \log n)$ size, 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 $O(\kappa)$ [32], [33], [14], [36], [37], [16]. However, threshold cryptosystems may suffer from performance degradation as n grows [38]. In practice, most implementations use a set of $O(n)$ digital signatures (e.g., ECDSA) instead [32], [33], [14], [37], [16]. The communication complexity, however, is increased accordingly as mentioned above.

IV. Technical Overview of Pando

Scalable consistent broadcast (CBC) for the transmission process. We show the conventional CBC protocol in Fig. 2a. Our transmission process improves CBC using only one technique, as shown in Fig. 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 $\lambda = \frac{3\alpha}{\epsilon^2} \ln \frac{1}{\delta}$, 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 rate and α is a small constant (see Lemma 1 below and proof in Appendix A). Here, λ can be viewed as a security parameter independent of n. 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.

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 A.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 $2\kappa/3$.*

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 roughly 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 Fig. 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.

An interesting fact is that since we sample three committees in each epoch, we could lower the committee size to improve the performance further. Informally speaking, if δ is the probability that each committee has more than one-third of faulty replicas, the probability of safety and liveness

violation of our protocol becomes $O(\delta^2)$! Accordingly, given the same *desirable* probability of safety and liveness violation, the committee size we use in Pando is an order of magnitude lower than existing works.

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. We show the proof of communication complexity in our full paper [39]. 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 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 approach with $O(\kappa n)$ messages and $O(L\kappa n^2)$ communication.

Remark on adaptive adversary model. The adaptive adversary model is used in theory to capture a more powerful adversary. One crucial design principle for the committee sampling-based approach, as mentioned above, is to make it too late to corrupt a committee member. In practice, the concrete implementation of a point-to-point channel (e.g., TCP) often employs a transmit-ack-retransmit pattern. In this way, a more powerful adversary can learn some information from the channel before the message is sent to all replicas. Our work assumes atomic sends (Sec. II) to rule out such behavior theoretically. We leave it as an interesting future work how adaptive adversaries can be realized in practice and what impact they can create on BFT protocols.

V. The Pando Protocol

A. 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 (denoted as 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 $W[le]$, which 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.

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) **do**
3: $O \leftarrow$ Obtain(le, m)
4: obtain the non-overlapped transactions in O and deliver in a deterministic order
5: set $ce \leftarrow le$

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: $M \leftarrow$ select(Q)
5: send (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 $C^i_{t,e}$ **do**
8: let qc_i be the set of valid signatures
9: **wait until** $|proposals[e]| \geq n - f$
10: $e \leftarrow e + 1$
11: InitEpoch(e)
12: **upon** receiving (Proposal$, e, M, qc_j$) from P_j s.t. $j \in [n]$ **do**
13: **if** $P_i \in C^j_{t,e}$ **then**
14: $h \leftarrow Hash(M)$
15: create a signature σ_i for (e, h, j) and send to P_j
16: $proposals[e][j] \leftarrow M$
17: $W[e-1] \leftarrow W[e-1] \cup qc_j$

B. The Transmission Process

The transmission process can be viewed as a scalable version of pipelined consistent broadcast (CBC). Below, we present a pipelining mode, where a replica sends the QCs for the prior epoch and also a new block to all replicas. 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 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 ComVerify() function, as described in Sec. II. 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 InitEpoch(e) function (line 2). In this function, P_i obtains a batch of transactions M from its queue Q and then sends a (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 14). For each replica P_i, upon receiving a proposal (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 it 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 16-17). 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 (PROPOSAL) messages before entering the next epoch (line 9).

C. The Consensus Process

The consensus process is shown in Algorithm 4 and we use an atomic broadcast protocol to instantiate the consensus process. The protocol has four phases: NEW-VIEW, PROPOSE, PREPARE, and COMMIT. 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 $le \mod n$ 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 45). In each epoch, the protocol proceeds as follows.

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 12). By default, P_i uses $W[le]$ as the proposal for the current epoch. Additionally, if $height(qc_{high})$ is lower than $le-1$, some block in 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 14-16). After that, P_i creates a block b with content W_i, the height le, and hash of qc_{high}. Then, P_i sends a (PROPOSE, b, le, qc_{high}) message to all replicas (line 18). 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 21 and Algorithm 5, lines 1-5). 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 sends a (PREPARE, $hash(b), le, \sigma_i$) message to all replicas, where σ_i is a signature for $(1, hash(b), le)$.

Algorithm 4 The consensus process for replica P_i

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

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: **function** VerifyQCs(W_j, e)
7: **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 σ_k is a valid signature for $(e, *, \ell)$ **then**
8: **return** $true$
9: **else**
10: **return** $false$
11: **function** CheckLeader(e, i)
12: **if** $i = e \mod n + 1$ **then**
13: **return** $true$
14: **else**
15: **return** $false$
16: **function** ObtainMissing(ce, le, m)
17: $\boldsymbol{m} \leftarrow \perp$
18: **for** $e \in [ce, le]$ **do**
19: **if** $\exists W_e$ s.t., $W_e \in m$ **then**
20: $\boldsymbol{m}[e] \leftarrow W_e$
21: **else**
22: wait for m_e from block b proposed in epoch e
23: $\boldsymbol{m}[e] \leftarrow m_e$
24: **return** $\boldsymbol{m}$

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** $qc_j \in m$ **do**
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: send (Distribute, $j, proposals[e][j]$) to all replicas
9: **upon** receiving (Distribute, j, M) from P_k **do**
10: **if** $P_k \in C^j_{s,e}$ **and** $Hash(M)$ matches that corresponding to qc_j **then**
11: $O \leftarrow O \cup M$
12: **wait until** $|O| = |m|$
13: clear $W[e]$ and remove transactions in O from Q

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 27).

If a replica P_i belongs to $C^3_{c,le}$, it creates a signature for $(2, hash(b), le)$ and then sends (Commit, h, le, σ_i) to all replicas (lines 28-30), 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 (Commit, h, le, σ_i) to all replicas (lines 31-34).

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 38). 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(ce, le, m) function to obtain the missing values between $ce + 1$ and $le - 1$ (lines 39-40). 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 19-20), 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 21-23). This is because 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 [31], [32]). Then P_i *a-delivers* the value.

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

D. 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. 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 9-10). 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.

E. 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, every committee member 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.

Formally, our protocol achieves the following properties. We provide the proof and analysis of complexities in the supplementary material.

Theorem 1 (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 $1 - O(\delta^2)$.

Theorem 2 (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 $1 - O(\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}})$.*

VI. Analysis of Probability of Achieving Safety and Liveness

We analyze the concrete probability of safety and liveness violation of Pando in our full paper [39] 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 is violated if in the consensus process, a correct replica *a-delivers* m and another correct replica *a-delivers* m' and $m \neq m'$. As shown in Theorem 9, 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 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 Fig. 3. We also show some examples of the concrete probabilities in Table II and Table III. In the tables, 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. 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^{-8} so the protocol fails once every 100 million epochs), *how much* resilience needs to be sacrificed for each n. As shown in Table II and Table III, n does not have to be impractically large in our system. For example, in Table II, 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 lower probability of safety and liveness violation (e.g., 10^{-8} or 10^{-9}) with the same n, ϵ has to be higher, as shown in Fig. 3 and Table III.

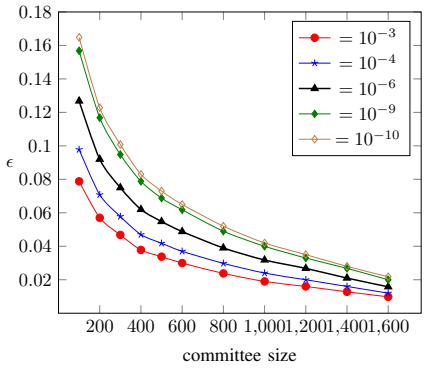


Fig. 3: Committee size vs. ϵ (n = 2,000) to limit the probability of violating safety and liveness to 10^{-3}, 10^{-4}, 10^{-6}, 10^{-9}, and 10^{-10} respectively.

Pando vs. existing works. It now becomes clear why Pando can reduce the committee size of existing works and why Pando is more efficient. In fact, we can use the results in Table III as those for the state-of-the-art committee sampling-based approach for achieving $1 - 10^{-4}$ probability. Briefly speaking, this is because the probability of safety violation in this work is $O(\delta^2)$ while previous works have a probability of $O(\delta)$. Compared to prior work, Pando has a much smaller committee size given the same failure probability and ϵ. For example, for n =1,000, Pando (0.2) has $\epsilon = 0.067$ (top-right cell of Table II) and a committee size of 200. To achieve the same $1 - 10^{-4}$ probability and with $\epsilon = 0.067$, existing work requires roughly 400 committees (second row of the rightmost column in Table III.

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

$n =$	100	200	300	400	500	1000
Pando (0.2)	0.193	0.133	0.123	0.103	0.091	0.067
Pando (0.4)	0.123	0.093	0.076	0.063	0.059	0.041
Pando (0.6)	0.093	0.063	0.053	0.046	0.041	0.029
Pando (0.8)	0.053	0.038	0.033	0.028	0.023	0.017

TABLE II: 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.

$n =$	100	200	300	400	500	1000
Pando (0.2)	0.253	0.198	0.177	0.153	0.137	0.102
Pando (0.4)	0.173	0.133	0.113	0.098	0.089	0.064
Pando (0.6)	0.123	0.093	0.08	0.068	0.061	0.044
Pando (0.8)	–	0.053	0.05	0.041	0.037	0.027

TABLE III: The value of ϵ for the system to achieve safety and liveness with a probability of at least $1 - 10^{-8}$. Hyphen means no ϵ value can make the desirable probability at least $1 - 10^{-8}$.

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 rate 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 our full paper [39], the probability of liveness violation is $O(\delta^{\frac{\frac{1}{9}-\epsilon^2}{\epsilon^2}})$ for $\epsilon < 0.192$ or $O(\delta^2)$ for $\epsilon \in [0.192, 0.333)$.

VII. Implementation and Evaluation

We implement Pando in Golang[3]. We compare the performance of Pando with Star [14], Narwhal-HS [11], and Algorand [25]. We implement Star in our library and assess Narwhal using their open-source implementation[4]. We assess these two protocols as they have the same partial synchrony assumption as ours.

Our codebase involves around 10,000 LOC for the protocols 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[5] for erasure coding. We use the Golang-based VRF implementation[6] to instantiate the ComProve() and ComVerify() oracle. The VRF scheme we use achieves adaptive security under the random oracle assumption.

We evaluate the performance of our protocols on Amazon EC2 using up to 500 virtual machines (VMs) and up to 1,000 replicas (mainly because we are not allowed to launch more than 500 instances on our AWS account). 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 may use each instance to run multiple 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 run our protocols for several epochs and report the results when the performance becomes stable. We repeat each experiment five times and report the average performance. The default transaction size is 250 bytes.

When evaluating Pando, we vary the committee sizes from $0.2n$ to n. 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 II. To achieve a failure rate of lower than 10^{-4}, Pando (0.6) needs to set $\epsilon = 0.093$ when $n = 100$, i.e., $f < 0.24n$. When n is larger, ϵ can be much lower. For instance, for n =1,000, Pando (0.4) can support $f < 0.292n$. To have a lower failure rate, the committee size has to be larger, as summarized in Fig. 3.

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 smaller 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. 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) 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, the CPU is usually the bottleneck of the system. In contrast, for the large-scale network, the network bandwidth is the bottleneck.

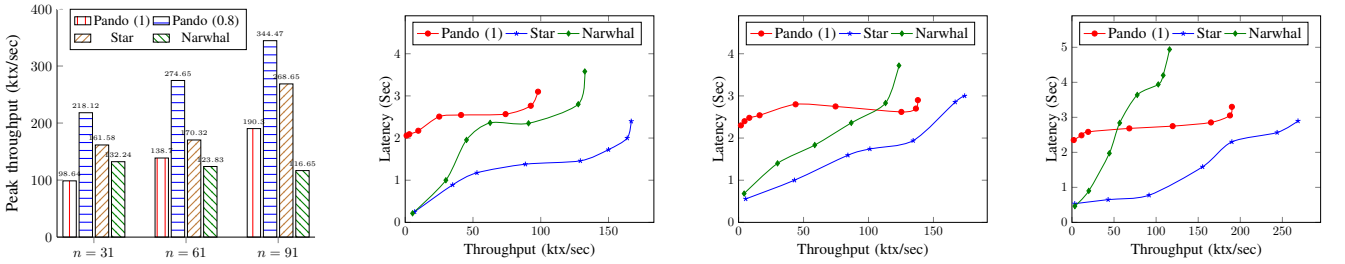
Comparison of Pando, Narwhal, and Star. We first assess the peak throughput of Pando, Narwhal, and Star. We were not able to successfully run Narwhal and Star for a network beyond 100 replicas as we met a frequent "connection refused" error due to high communication costs. We believe this is mainly because our experiments are launched on low-end VMs. Besides the fact that experiments cannot be launched for larger network sizes, even for the network size where the experiments can be launched, the latency grows significantly as n grows. This is due to the high memory consumption required for network communication. Accordingly, our comparison

[3]Our codebase: https://doi.org/10.5281/zenodo.16959662 or https://github.com/DSSLab-Tsinghua/Pando

[4]https://github.com/MystenLabs/narwhal

[5]https://github.com/klauspost/reedsolomon

[6]https://github.com/yoseplee/vrf

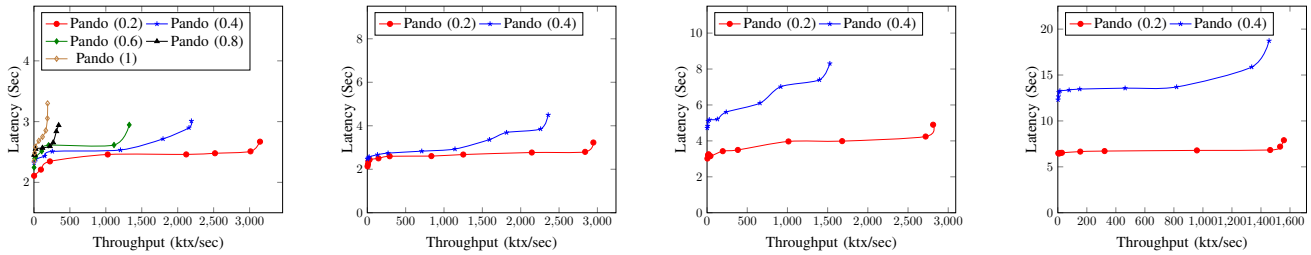


(a) Peak throughput of Star, Narwhal and Pando as f grows.

(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 $n = 91$.

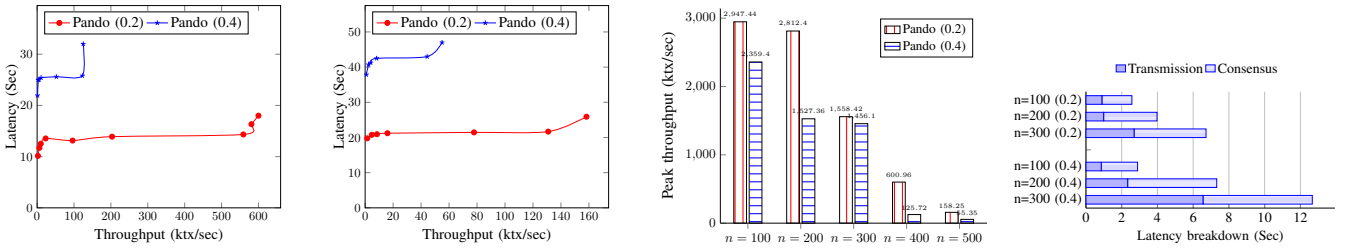


(e) Latency vs. throughput of Pando in WAN using different committee sizes for $n = 91$.

(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 $n = 91$ with 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.

Fig. 4: Performance of the protocols.

focuses on the setting for $n < 100$. We report the peak throughput of Pando (1), Pando (0.8), Star, and Narwhal in Fig. 4a and latency vs. throughput for $n = 31, 61, 91$ in Fig. 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 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 that of 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 Fig. 4e, the performance of Pando is higher (higher peak throughput, lower latency) when the committee size is smaller. This is expected as having a small committee size will lower both communication and computational costs. The drawback is that for a network of 91 replicas, ϵ has to be larger for smaller committee sizes, as summarized in Table II.

network size	protocol	latency (ms)
$n = 100$	Pando (0.2)	96
	Pando (0.4)	215
	Pando (1)	249
$n = 200$	Pando (0.2)	271
	Pando (0.4)	335
	Pando (1)	9131
$n = 300$	Pando (0.2)	707
	Pando (0.4)	911
	Pando (1)	27203

TABLE IV: Latency of the state transfer process of Pando.

Comparison with Algorand. As mentioned previously, Algorand was the first practical VRF-based committee sampling protocol. To achieve a failure rate of lower than 10^{-9} and limit the system has at least 80% correct replicas, the committee size has to be 2,000 for Algorand [25]. We are not able to launch such a large-scale experiment due to the limit of resources. Alternatively, we compare the performance of Pando using both the results reported in the Algorand paper and a local small-scale deployment. As reported in the paper, using 1,000 VMs to run n =50,000 replicas, Algorand can deliver 2 MBytes of transactions in 22 seconds. To achieve the same 10^{-9} failure rate, the committee size only needs to be 200, i.e., Pando (0.4) for $n \geq 500$. Our experimental results show that Pando (0.4) delivers 13.84 MByte of transactions in 1 second (as shown in Fig. 4j), about 154x that of Algorand. Meanwhile, we also evaluate Algorand in a local testnet with $n = 7$ where all replicas act as committee members. Our evaluation results show that Algorand achieves a peak throughput of 5.72 ktx/s. With the same settings, Pando (1) achieves a peak throughput of 75.21 ktx/s, about 13.15x that of Algorand.

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 Fig. 4m, we show the CPU and bandwidth usage of Pando (0.8). It can be seen that the CPU usage and bandwidth usage grow as b grows. When the CPU is fully used, the throughput does not grow anymore. Additionally, in Fig. 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 resource is fully used. For other cases, as the committee size is smaller, the CPU usage and bandwidth usage are also lower. Thus, the protocol achieves its peak throughput using an even larger batch size.

Computational overhead. In each epoch, each replica needs to query $O(n)$ VRF functions. In the instances we use, the latency for each ComProve() function is 2.8ms, and the latency of the ComVerify() function is 0.1ms. Even if n is large, the oracle does not create significant overhead.

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 Fig. 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.

size	protocol	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$
250B	Pando (0.2)	2947.44	2812.40	1558.42	600.96	158.25
	Pando (0.4)	2359.40	1527.36	1456.10	125.72	55.35
250KB	Pando (0.2)	3.62	3.30	2.48	0.98	0.42
	Pando (0.4)	3.06	2.09	1.34	0.52	0.23

TABLE V: Peak throughput (ktx/sec) of Pando under different transactions sizes.

Scalability and latency breakdown. We report the peak throughput of Pando for $n = 100$ to 500 in Fig. 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 Fig. 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 reduced using approaches such as aggregate signatures.

The latency of the state transfer process. We report the latency of the state transfer process of Pando for $n = 100, 200, 300$ in Table IV. We evaluate Pando (0.2), Pando (0.4), Pando (1). As shown in the table, the latency of the state transfer process of Pando with a smaller committee size is consistently lower than Pando (1), especially when n is large. For $n = 300$, the latency of Pando (0.4) is only 3.3% of that for Pando (1).

Evaluation using different transaction sizes. We evaluate the performance of Pando using 250KB transaction size. As shown in Table V, the performance of Pando is lower as the transaction size is larger. This is expected, as higher network bandwidth is needed to disseminate the transactions.

instance	vCPU	memory (GiB)	bandwidth (Gbps)	batch size	peak tps (ktx/sec)
m5.xlarge	4	16	up to 10	100,000	2947.43
m5.large	2	8	up to 10	100,000	2443.05
m4.xlarge	4	16	0.75	100,000	1316.31
t2.micro[7]	1	1	up to 0.72	5,000	95.37

TABLE VI: Peak throughput of Pando (0.2) under different low-bandwidth and on-premise cluster.

Experiments using low-end VMs. We assess the performance of Pando using low-end VMs for $n = 100$, as summarized

[7]AWS t2.micro provides a baseline level of CPU performance with the ability to burst above the baseline. The bandwidth of t2.micro was tested as 0.06 Gbps sustained with 0.72 Gbps bursts [40].

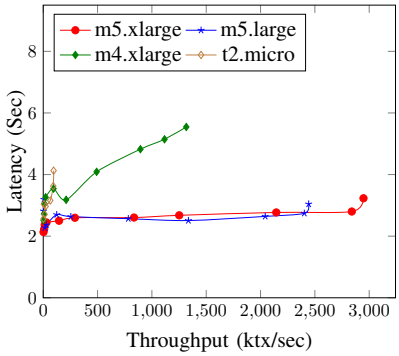


Fig. 5: Latency vs. throughput of Pando (0.2) for $n = 100$ under different low-bandwidth and on-premise cluster.

instance	# VM	bandwidth (Gbps)	batch size	peak tps (ktx/sec)
m5.2	200	up to 10	-	-
m5.2	500	up to 10	-	-
m5n.2	200	up to 25	5,000	62.57
m5n.2	500	up to 25	5,000	73.2

TABLE VII: Peak throughput of Pando for $n = 1,000$ using different number of VMs.

in Table VI and Fig. 5. For Pando (0.2), we use four VMs: *m5.xlarge*, *m5.large*, *m4.xlarge*, and *t2.micro*. When the CPU and memory become lower, the throughput decreases. For VMs with the same CPU and memory, the throughput becomes lower on the bandwidth-restricted VMs. These results are expected, as our protocols require both memory and bandwidth for computation and communication. Even with the *t2.micro* instance (the lowest setup we assess), Pando still achieves over 95.37 ktx/sec throughput.

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. As summarized in Fig. 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., $m5n.2xlarge$), Pando achieves a throughput of up to 62.57 ktx/sec. For VM with better configuration but lower network bandwidth (e.g., $c5.4xlarge$), Pando only achieves a throughput of 1.6 ktx/sec, as we were not able to run the experiments with a larger batch size.

We launch an additional experiment on 500 instances to repeat the 1,000-replica experiments. As summarized in Table VII, Pando achieves a throughput of up to 73.2 ktx/sec, slightly better than 62.57 ktx/sec with 200 instances. The performance is not that different, so our experiments using fewer instances validate the practicality of our protocol.

Summary of deployment concerns. Based on the experiments, we believe that deploying Pando on machines with high network bandwidth and moderate CPU will improve the performance of Pando. Our experiments show that 8 vCPU is good enough, but VMs with higher bandwidth will further improve the performance of the system.

VIII. Related Work

More discussion about Algorand vs. Pando. Algorand [25], [26] 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, as mentioned in the introduction, the design of Pando allows for a much smaller committee size.

Byzantine agreement (BA) and Byzantine broadcast (BC) 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 of the BA and BC protocols, assuming the existence of a sampler [19], [22], [20], [21].

Abraham et al. [19] proposed a binary BA with subquadratic communication complexity. Meanwhile, it revisits VRF-based committee sampling by Algorand and formalizes a $\mathcal{F}_{mine}$ function for committee sampling. Later work all follow this notion, including ours (specifically, the ComProve() and ComVerify() oracle in Sec. II match the $\mathcal{F}_{mine}$ abstraction). In the asynchronous setting, Blum, Katz, Liu-Zhang, and Loss [22] presented a BA protocol achieving subquadratic communication complexity under the adaptive adversary setting assuming $f < (1 - \epsilon)n/3$ (interchangeable with our $f < (1/3-\epsilon)n$ assumption). Additionally, a line of work studies Byzantine broadcast (a problem limited to the synchronous setting) assuming $f < (1 - \epsilon)n$, and uses committee-based approaches to optimize the communication [20], [21], [41].

This paper studies BFT. Since BFT is different from BA and BC, the protocol designs are fundamentally different.

Partially synchronous BFT. Partially synchronous BFT has been widely studied in the literature [42]. Starting from PBFT [31], an impressive number of practical BFT protocols are proposed (e.g., [43], [44], [45], [46], [47]). HotStuff [32] provides a three-phase solution that achieves linear message complexity, and many efforts have been made to reduce the number of phases required [33], [48], [49], [50]. A recent work ProBFT [51] studies partially synchronous BFT with

optimized latency under a probabilistic model and a static adversary assumption. Our ABC protocol of the consensus process is a scalable version of prior protocols such as PBFT and HotStuff under the adaptive adversary assumption.

BFT with adaptive security. Protocols that are secure in the static adversary model might not be adaptively secure [52], [53]. Specifically, protocols that rely on threshold cryptography of committee sampling might not be adaptively secure, and performance usually degrades compared to those under static security model [54], [55].

Asynchronous BFT. The celebrated FLP result [56] rules out the possibility of deterministic consensus in asynchronous environments, so asynchronous must be probabilistically live. Asynchronous BFT protocols have been extensively studied [38], [57], [37], [58], [54], [59], [60], [61]. Our transmission process and state transfer process are fully asynchronous.

IX. Conclusion

We present Pando, a practical and scalable BFT from committee sampling. 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.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2701700, the National Natural Science Foundation of China under 92267203 and 62272043, Beijing Natural Science Foundation under M23015, Yangtze Delta Region Institute of Tsinghua University, Zhejiang (No. LZZLX24F007), WeBank scholars program, and Tsinghua Shuimu Scholar program.

X. Research Ethics Considerations

This research is committed to the principles of research ethics. In particular, we adhere to the following principles:

- **Respect for persons**: Our research does not involve human subjects or personal data. We respect the work of other researchers and properly cite all relevant prior work.
- **Beneficence**: Our research studies scalable Byzantine fault-tolerant protocols from committee-based sampling approaches. It offers a nice way to improve the scalability of blockchain systems and other related areas.
- **Justice**: We have formally proved the correctness of the protocol. Any user's benefits are equally protected if the protocols correctly implemented.
- **Respect for law and public interest**: Our research complies with all applicable laws and regulations. We have considered the broader societal implications of more secure blockchain systems.

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 *Security and Privacy (SP)*, 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] N. Giridharan, F. Suri-Payer, I. Abraham, L. Alvisi, and N. Crooks, "Motorway: Seamless high speed bft," in *SOSP*, 2024.

[16] 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.

[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," *JACM*, 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*, 2020.

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

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

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

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

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

[29] Y. Xu, J. Zheng, B. Düdder, T. Slaats, and Y. Zhou, "A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance," in *NDSS*, 2024.

[30] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy," in *CCS*, 2022, pp. 683–696.

[31] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *TOCS*, vol. 20, no. 4, pp. 398–461, 2002.
[32] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *PODC*, 2019.
[33] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," *DSN*, 2022.
[34] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving the latency of dag-based bft," in *Security and Privacy (SP)*, 2025.
[35] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC*. ACM, 2021, pp. 165–175.
[36] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*. Springer, 2001, pp. 524–541.
[37] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols." in *CCS*, 2020.
[38] H. Zhang and S. Duan, "PACE: Fully parallelizable bft from reproposable byzantine agreement," in *CCS*, 2022.
[39] X. Wang, H. Wang, H. Zhang, and S. Duan, "Pando: Extremely scalable BFT based on committee sampling," Cryptology ePrint Archive, Paper 2024/664, 2024. [Online]. Available: https://eprint.iacr.org/2024/664
[40] "Bandwidth of t2.micro instance," https://repost.aws/questions/QUM2vwaKIsQHGGt6Y8uYG5OA/how-much-bandwidth-is-t2-micro-instance-type#ANTVXwCHtxThqpTP0RpeV8yQ.
[41] D. Collins, S. Duan, J. Loss, C. Papamanthou, G. Tsimos, and H. Wang, "Towards optimal parallel broadcast under a dishonest majority," in *FC*, 2025.
[42] X. Wang, S. Duan, J. Clavin, and H. Zhang, "Bft in blockchains: From protocols to use cases," *ACM Computing Surveys (CSUR)*, 2022.
[43] J.-P. Bahsoun, R. Guerraoui, and A. Shoker, "Making BFT protocols really adaptive," in *IPDPS*. IEEE, 2015, pp. 904–913.
[44] J. Li and D. Maziéres, "Beyond one-third faulty replicas in byzantine fault tolerant systems." in *NSDI*, 2007.
[45] S. Duan, H. Meling, S. Peisert, and H. Zhang, "BChain: Byzantine replication with high throughput and embedded reconfiguration," in *OPODIS*, 2014, pp. 91–106.
[46] 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.
[47] S. Duan and H. Zhang, "Foundations of dynamic bft," in *Security and Privacy (SP)*, 2022, pp. 1317–1334.
[48] 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.
[49] 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.
[50] X. Sui, S. Duan, and H. Zhang, "BG: A modular treatment of BFT consensus," *TIFS*, 2024.
[51] D. Avelãs, H. Heydari, E. Alchieri, T. Distler, and A. Bessani, "Probabilistic byzantine fault tolerance," in *PODC*, 2024, pp. 170–181.
[52] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure multi-party computation," in *STOC*, 1996, pp. 639–648.
[53] 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.
[54] C. Liu, S. Duan, and H. Zhang, "Epic: Efficient asynchronous bft with adaptive security," in *DSN*, 2020.
[55] H. Zhang, C. Liu, and S. Duan, "How to achieve adaptive security for asynchronous bft?" *JPDC*, vol. 169, pp. 252–268, 2022.
[56] 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.
[57] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*, 2016, pp. 31–42.
[58] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous bft closer to practice," *NDSS*, 2022.
[59] S. Duan, M. K. Reiter, and H. Zhang, "BEAT: Asynchronous bft made practical," in *CCS*. ACM, 2018, pp. 2028–2041.
[60] S. Duan, X. Wang, and H. Zhang, "Practical signature-free asynchronous common subset in constant time," in *CCS*, 2023.
[61] H. Zhang, S. Duan, B. Zhao, and L. Zhu, "Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft," in *Usenix Security*, 2023.

APPENDIX A
PROOF OF CORRECTNESS

A. 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 $0 < \epsilon < \frac{1}{3}$, 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. Suppose $P_i, i \in [c]$ be the i-th committee member, and 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, the probability that a committee member is faulty is equal to the fraction of faulty replicas among all replicas (i.e., α), so $\Pr(X_i = 1) = \alpha$, for each $i = 1, 2, \cdots, c$, as shown in Table VIII.

x	1	0
$\Pr(X_i = x)$	α	$1-\alpha$

TABLE VIII: 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 in the committee. 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 1.1. *Let α^* be the fraction of correct replicas in the system that hold 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* $1 - \mathit{negl}(\kappa)$.

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* $1-\mathit{negl}(\kappa)$*, 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 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 2.1. *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 be the size of the committee. Then the probability that no more than one correct replica is elected as a committee member is:

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

□

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:

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

□

Corollary 4.1. *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* $1 - \mathit{negl}(\kappa)$*, 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* $1 - \mathit{negl}(\kappa)$*, 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. □

B. 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 $height(qc_{high}) \geq e$. Additionally, at least $t+1$ correct replicas in $C^2_{c,e'}$ accept the (*PROPOSE*) message only if $height(qc_{high}) \geq e$.*

Proof. We know that P_i receives $\kappa - t$ matching messages from $C^3_{c,e}$. According to Corollary 2.1, 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 1.1, 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 1.1, 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 9 (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 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 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 9, 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:

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

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 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 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, $m = m'$.

We leave the discussion about Case-2 to Case-4 to our full paper. To conclude, the probability that safety is violated is $O(\delta^2)$. □

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, 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 11. *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 $\kappa - t$ (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 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 12. *If a correct replica P_i queries 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 11, at least one correct replica has completed epoch e. Furthermore, according to Lemma 10, P_i eventually receives $\kappa - t$ matching (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 13 (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 $\kappa - t$ (COMMIT, $h, e, -$) messages with the same h. According to Lemma 10, any correct replica eventually receives $\kappa - t$ (COMMIT, $h, e, -$) messages with the same h. Furthermore, P_i either directly *a-delivers* some value or obtains some value from the ObtainMissing() function. According to Lemma 12, 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
ARTIFACT APPENDIX

A. Description & Requirements

1) How to access: The artifact can be accessed via the stable URL: https://doi.org/10.5281/zenodo.16959662.

2) Hardware dependencies: Our test environment requires a commodity desktop machine with at least an x86-64 CPU with 8 cores and 16 GB of RAM running a recent Linux operating system.

3) Software dependencies: Our experiments run on Ubuntu 22.04. To compile our code of protocols, we require Go1.23.3 linux/amd64. We also use Python for log summarization. Make sure you have Python 3.10 (or a later version) installed. To prevent evaluators from downloading dependencies, the codebase we provided includes all necessary dependencies, which are located in "*Pando/src/*".

4) Benchmarks: None

B. Artifact Installation & Configuration

1) Installation: After installing Go, the artifact can be installed by running the following commands. Download the repository from https://doi.org/10.5281/zenodo.16959662 and decompress "*./Pando.zip*" manually or follow the command:

- `unzip Pando.zip -d Pando`

Next, set up the environment variable and disable Go modules mode under "*./Pando*" folder.

- `cd Pando`
- `export GOPATH=$PWD && export GOBIN=$PWD/bin`
- `go env -w GO111MODULE=off`

Compile the code by running the following commands.

- `bash ./scripts/install.sh`

If no errors are reported after completing the commands above, the compilation is successful. Executable files are expected to appear under "*Pando/bin*".

2) Configuration: Modify the configuration file "*Pando/etc/conf.json*" to choose the optional arguments. The *id* of each server should be unique. In our repository, we provide the configuration with 31 servers (id from 0 to 30) in "*Pando/etc/conf.json*".

Using the default configuration file, the codebase can be run locally on a single machine. If multiple machines or another configuration are used for evaluation, please also modify the *host* and *port* of all servers manually.

Generate the ECDSA key pairs for servers with IDs $0, \cdots, 30$ and a client with ID 1,000 by running the command:

- `bin/keygen 1 0 30 && bin/keygen 1 1000`

If keys are successfully generated, they are located under "*Pando/etc/key*".

C. Experiment Workflow

The artifact contains four folders with three log summarization scripts. Folder "*etc*" stores configuration files and key pairs for servers. The "*scripts*" contains bash scripts to run the experiments automatically. The "*src*" includes the source code and dependencies for Pando system, and the "*var*" stores all execution logs.

D. Major Claims

- (C1): PANDO achieves lower latency and higher throughput when the system committee size decreases. This is proven by the experiment (E1) whose results are illustrated in Figure 4e, Section VII of the paper.
- (C2): PANDO achieves a steady growth in latency vs. throughput as system batch size increases. This is proven by the experiments (E2) whose results are illustrated in Figure 4f-4j, Section VII of the paper.
- (C3): For the latency breakdown of PANDO system, the latency of the consensus process is higher than the transmission process. This is proven by the experiments (E3) whose results are illustrated in Figure 4l, Section VII of the paper.

E. Evaluation

We conduct three types of experiments: Pando with different committee sizes (E1), latency vs. throughput experiments (E2), and latency breakdown experiments (E3). E1, E2 and E3 are used to validate Claim C1, C2 and C3, respectively.

[Notes on scaled-down experiments] Most results reported in the paper require access to the Amazon EC2. In this artifact appendix, we provide the procedure for reproducing scaled-down experiments using one machine.

[Notes on possible errors] Due to large resource consumption on one machine, the execution could stop to confirm blocks (no increasing epochs). When this failure happens, just terminate all processes using the following command and then restart the experiment:

- `bash ./scripts/killProcess.sh`

The artifact may cause different results each time and has to run multiple times for the expected results. Please run the experiment multiple times and consider the average value.

1) Experiment (E1): [10 human-minutes]:

We assess latency vs. throughput for Pando for $n = 31$ by varying the committee size as $0.2n, 0.4n$ and $0.8n$.

[How to] Modify the configuration file and run the experiments in 3 scenarios under "*./Pando*" folder.

[Preparation] Modify the configuration file "*Pando/etc/conf.json*". The argument $committeeSizes$ needs to be modified based on the scenario setups. For example, when we evaluate scenario Pando (0.8), $committeeSizes$ needs to be modified to 0.8 in "*Pando/etc/conf.json*".

[Execution] E1 has to run the experiment in 3 scenarios. Below, we show the execution procedures for Pando (0.8).

Run the experiment under "*./Pando*" folder:

- `bash ./scripts/runE1.sh`

This script will first run all 31 servers in the background (approximately 15 seconds in total). The following outputs are expected to be displayed on the terminal:

```
2025/07/14 09:52:08 **Starting replica 0
2025/07/14 09:52:08 Use ECDSA for authentication
open
↪  /home/starly/Pando/var/log/0/20250714_Normal.log
2025/07/14 09:52:08 [User] User starly
2025/07/14 09:52:08 User [starly]
```

```
2025/07/14 09:52:08 **Storage option: Data are
↪  stored at consensus replicas
2025/07/14 09:52:08 >>>Running Pando protocol!!!
open  /home/starly/Pando/var/log/0/20250714_Eva.log
2025/07/14 09:52:08 Start transmission process...
2025/07/14 09:52:08 Start Pando consensus process...
2025/07/14 09:52:08 ready to listen to port :11000
2025/07/14 09:52:13 ############### epoch 0...
2025/07/14 09:52:14 ############### epoch 1...
...
```

Since all outputs of 31 servers will display in one terminal immediately, it is difficult to check each printout manually. Just wait for a few seconds (approximately 15 seconds), the terminal will display `epoch 1...` every second, which means that all servers are listening to the client request.

Next, the script will send client requests, including 5 transaction blocks, to Pando servers (approximately 60 seconds in total, from epoch 1 to 5).

At the end of the experiment, the output will display:

```
############### epoch 5...
client-start: no process found
```

After the experiment is successfully launched, evaluation logs can be found in the folder "*./Pando/var/log*". Now calculate the performance with the command:

- `python3 summarizeE1.py`

The latency and throughput of scenario Pando (0.8) will be printed, and the result will be recorded in "*Pando/resultE1.txt*".

[Repeat the experiment with two other scenarios.] To complete E1, one need to repeat the experiment with Pando (0.4) and Pando (0.2) with exactly the same procedures above. Do not forget to change the argument $committeeSizes$ in file "*Pando/etc/conf.json*" before each experiment.

[Results] After finishing the three experiments mentioned above for Pando (0.2), Pando (0.4) and Pando (0.8), one should observe from "*Pando/resultE1.txt*", that the latency should lower and the throughput should higher when the committee size is smaller. This validates Claim C1.

2) Experiment (E2) : [20 human-minutes]:

We assess latency vs. throughput for Pando (0.2) by varying the batch size as $100, 300, 500, 1000, 3000$ and 5000.

[How to] Run the experiments in 6 scenarios with the command under "*./Pando*" folder.

[Preparation] Modify the configuration file "*Pando/etc/conf.json*". The argument $committeeSizes$ needs to be modified to 0.2.

Also, remember to modify the $batchSize$ argument based on the scenario setups. For example, when we evaluate Pando with batch size 5000, we need to modify the $batchSize$ to 5000 in "*Pando/etc/conf.json*".

[Execution] E2 has to run the experiment in 6 scenarios. Evaluators need to modify the $batchSize$ configuration before running different E2 scripts. Here we show the detailed instructions for batch size 5000 below.

Run the experiment under "*./Pando*" folder:

- `bash ./scripts/runE2_batchsize5000.sh`

The command will run all 31 servers before submitting client requests like that in E1. The expected outputs should be similar to those in the E1 experiment.

After execution is completed, calculate the scenario performance with the command under "*./Pando*" folder:

- `python3 summarizeE2.py`

The latency and throughput of 5000 batch size will be printed, and the result will be recorded in "*./Pando/resultE2.txt*".

[Repeat the experiment with five other scenarios] To complete E2, one needs to repeat the experiment with $100, 300, 500, 1000$ and 3000 batch sizes. Remember to modify the $batchSize$ argument in "*Pando/etc/conf.json*" before running E2 scripts. We provide scripts for each scenario below.

Run 100 *batch size scenario:*

- `bash ./scripts/runE2_batchsize100.sh`

Run 300 *batch size scenario:*

- `bash ./scripts/runE2_batchsize300.sh`

Run 500 *batch size scenario:*

- `bash ./scripts/runE2_batchsize500.sh`

Run 1000 *batch size scenario:*

- `bash ./scripts/runE2_batchsize1000.sh`

Run 3000 *batch size scenario:*

- `bash ./scripts/runE2_batchsize3000.sh`

[Results] After completing the procedures above for batch size $100, 300, 500, 1000, 3000$ and 5000, one should observe from "*./Pando/resultE2.txt*" that the trend of latency vs. throughput should align with the data shown in Figure 4f-4j in the paper. Namely, the throughput of Pando is expected to increase when the batch size grows. Note that the results may vary depending on the machine used. For example, the latency may not appear to be monotonically increasing as the batch size increases. This validates Claim C2.

3) Experiment (E3) : [5 human-minutes]:

We assess latency breakdown of the transmission process and the consensus process for Pando (0.2) with 31 servers.

[How to] Run the experiment under "*./Pando*" folder.

[Preparation] Modify the configuration file "*Pando/etc/conf.json*". The argument $committeeSizes$ and needs to be modified to 0.2.

[Execution] Run the experiment with the command under "*./Pando*" folder:

- `bash ./scripts/runE3.sh`

The command runs all 31 servers and submits 5 transaction blocks to the servers. The output should be similar to that in the E1 experiment.

After completing the experiment, calculate the latency using the command under "*./Pando*" folder:

- `python3 summarizeE3.py`

The latency breakdown of Pando (0.2) will be printed, and the result will be recorded in "*./Pando/resultE3.txt*".

[Results] After completing E3, one should observe from "*./Pando/resultE3.txt*" that the trend of the latency breakdown is aligned with the data shown in Figure 4l in the paper. Namely, the latency of the consensus process is higher than that of the transmission process. This validates Claim C3.