rBFT: a Revamped Two-Stage BFT from Delegated Committee

Huizhong Li
ICT/CAS, WeBank Blockchain Team
lihuizhong21@mails.ucas.ac.cn

Shichen Wu
WeBank Blockchain Team
shichenw@mail.sdu.edu.cn

Mingfei Zhang
Shandong University
mingfei.zh@outlook.com

Yue Huang
Tsinghua University
y-huang22@mails.tsinghua.edu.cn

Linpeng Jia
ICT/CAS
jialinpeng@ict.ac.cn

Sisi Duan
Tsinghua University
duansisi@tsinghua.edu.cn

Yi Sun
ICT/CAS
sunyi@ict.ac.cn

***Abstract*—Byzantine fault-tolerant (BFT) protocol from delegated committee is an approach in improving the performance and scalability of blockchains. Notable industrial examples include Delegated Proof-of-Stake (DPoS) by Tron, Polkadot, and Solana, and Proof-of-Staked-Authority (PoSA) by Binance. In these protocols, a subset of nodes is first selected to form a committee, then the committee members reach an agreement and disseminate the results to all nodes. Although these approaches allow the committee members to be rotated periodically, the security of the system is built upon a strong assumption that no committee can have more than certain fraction of faulty nodes (e.g., one-third in a partially synchronous network).**

In this paper, we provide a revamped two-stage design to model BFT from delegated committee without making the strong assumption. Namely, the only assumption is that in a partially synchronous network, the entire system does not have more than one-third faulty nodes. We propose rBFT, a practical BFT protocol that has a fast path where only committee members participate and a slow path where all nodes in the system are involved. We show that, not surprisingly, the fast path is extremely fast, and under reasonable assumptions such as alive-but-corrupt nodes and rational nodes, only fast path can be triggered. Meanwhile, even under conventional Byzantine failures where the slow path is triggered, our protocol is still practical enough.

I. Introduction

Blockchains are known to suffer from the scalability issue, i.e., the performance of the system degrades significantly as the number of nodes grows. One of the scalability bottlenecks is the underlying consensus mechanism, also known as Byzantine fault-tolerant (BFT) protocols [1], [2], [3], [4], [5]. Despite many efforts that have been made to scale blockchain systems, e.g., Ethereum has over one million consensus validators (called nodes in our work) now[1], the performance of the consensus mechanism is still not satisfactory due to the high communication cost and computational cost.

Blockchains from delegated committee. To overcome the high communication cost and computational cost, several industrial systems choose to lower the number of *consensus nodes*, i.e., the number of parties that participate in the BFT protocol. Notable examples include Delegated Proof-of-Stake (DPoS) consensus mechanism used by TRON [6] and Polkadot [7], and the Proof-of-Staked-Authority (PoSA) [8] mechanism used by Binance. In both DPoS and PoSA, the committee members are selected via delegation of users' stake (i.e., informally account balance of blockchain account holders), but the committee members are rotated periodically. For example, Binance has 116 nodes in the entire system and selects 21 committee members at a time[2], most of which are selected according to the amount of their delegated stake. The committee members run the Clique protocol [9], [10] to agree on the order of the blocks, and non-committee members passively listen to the agreement results. The members are rotated once every day [3]. To ensure the correctness of the entire system, the assumption is that no committee has more than one-third faulty members.

The challenges with the two-stage design. Although a small committee is nice in the sense that the performance of the system can be made high, the assumption that no committee has more than one-third faulty members can be overly strong. Also, it is arguably much easier to corrupt more than one-third committee members rather than corrupting one-third nodes in the entire system. The takeover of DPoS of Steem blockchain [11] is an example. In 2020, 20 out of 21 committee members were controlled by an external party who then created a hard fork, i.e., double spending. The original Steem users were forced to leave the system.

If we consider the entire system as a whole, the current workflow of DPoS and PoSA falls into a two-stage pattern: (1) In stage 1, the committee members are selected, and then run a BFT protocol to reach an agreement on the order of blocks; (2) In stage 2, committee members disseminate the agreement results to all nodes in the system, who act as *learners* that passively accept the results. No hard forks can be detected, and there is no mechanism to recover from faulty committees.

Our starting point in this work is a two-stage BFT from

[1]Data source: https://www.beaconcha.in/ (accessed in Apr 2026)

[2]Source of data: https://bscscan.com/validators (accessed in Mar 2026).

[3]Source of the data: https://docs.bnbchain.org/bnb-smart-chain/validator/overview/ (accessed in Mar 2026)

committee sampling, where a small committee of λ nodes is first sampled among all n nodes. Statistially speaking, if we allow a small constant of error and λ is large enough, i.e., the entire system tolerates f failures where $f < (1/3 - \epsilon)n$ and $\epsilon \in (0, 1/3)$, the probability that the committee has more than one-third faulty members is negligible. Theoretically, there is a long line of research on building communication-efficient Byzantine broadcast and Byzantine agreement protocols from committee sampling [12], [13], [14], [15], [16], [17], [18]. Practically, Algorand [19] exploits committee sampling in its consensus mechanism to achieve satisfactory performance even when n is large. Unfortunately, the committee size λ has to be extremely large to make the probability of no faulty committee low. For instance, Algorand mentions that to achieve a failure probability of $1 - 10^{-9}$ and to set $\epsilon = 0.12$, a committee needs to have at least 2,000 nodes [19, Figure 3]. Such an observation introduces a circularity issue since BFT protocols are well known not to be that practical with over 2,000 nodes.

The research problem. In this paper, we study BFT from delegated committee where the committee members are randomly sampled. We aim to answer the following research question:

Can we build a revamped two-stage BFT from committee sampling that has a small committee size and efficiently handles faulty committees?

To answer the question above, we focus on the partially synchronous model [20] and static adversary model, i.e., there exists an unknown upper bound on message transmission and processing, and the adversary corrupts a set of nodes before the system is running. We argue that our assumptions are practical and building protocols under these assumptions can be quite challenging. We provide more discussion in Sec. III.

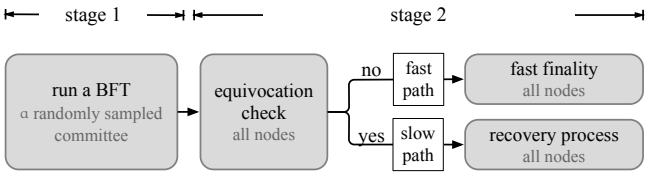


Fig. 1: The two-stage workflow of rBFT.

Our approach. We propose rBFT, a BFT protocol from committee sampling via a revamped two-stage design. We refine the definition of BFT protocol to answer the research question mentioned above, but the ultimate guarantees still remain the same as conventional BFT, i.e., safety and liveness [21]. As shown in Figure 1, our revamped two-stage design and our findings are summarized as follows.

- In stage 1, a committee is sampled and committee members run a BFT protocol under our refined definition. An equivocation detection mechanism is needed under our new definition of BFT to ensure that any correct node can *abort* under faulty committees.
- In stage 2, besides synchronizing the agreement results, all nodes run a protocol for *forensics* [22], [23], identifying whether some nodes in the committee have equivocated. If no equivocation is detected, the fast path is triggered. If equivocation is detected, a recovery process is triggered so all nodes can converge to the same chain.

Analysis via rational protocol design (RPD) and experimentation. We show that, under some reasonable assumptions of *weaker* adversary, such as alive-but-corrupt nodes [24] (i.e., if such nodes cannot break the safety of the system, they will help preserve the liveness) and rational nodes [25] (i.e., nodes that try to maximize a set of *utilities* rather than failing the security properties of the system), all nodes will always trigger the fast path, so the performance of the system is expected to be extremely high. Furthermore, even if all faulty nodes are Byzantine and when the slow path is triggered, our stage-2 protocol is still very lightweight.

One particular analysis approach we use is using the rational protocol design (RPD) [25]. The RPD framework was originally studied in the context of multiparty computation and later used to analyze the rationality of blockchain protocols [26]. By modeling a zero-sum extensive game between a protocol designer and an attacker, RPD provides a nice theoretical framework for designing and analyzing cryptographic protocols under incentive-driven adversaries. We show that under the guidance of RPD, faulty committee members that are rational will never trigger stage 2.

We implement our protocol and assess the performance on a cloud environment using up to 500 nodes. Our experimentation results show that when the fast path is triggered, the performance degradation compared to the vanilla two-stage design is lower than 1%. When the slow path is triggered and when 30% of nodes in the system are Byzantine, the throughput of the entire system is about 35% lower and the latency is about 5% higher.

Our contributions. To summarize, we make the following contributions.

- We study BFT from delegated committee in the static adversary model and provide a revamped two-stage design from committee sampling. We provide a refined definition of BFT to achieve the goal of having both a small committee size and efficient recovery under faulty committees.
- We provide a new protocol called rBFT under our refined definition of BFT with a fast path and a slow path. We show that under slighly weaker adversary assumptions such as alive-but-corrupt nodes and rational nodes, the fast path is always triggered. However, even under conventional Byzantine faulty nodes, the slow path is still very lightweight.
- We implement rBFT using 7,300 new lines of code. Our experimentation using up to 500 nodes shows that rBFT is highly practical under both paths.

II. Related Work

Accountability; slashing; forensics; recovery. Accountable BFT denotes protocols in which equivocating nodes can be caught [27], [28]. The concept was first introduced in distributed systems in general [29], [30], [31], and has since been

extensively studied in the context of blockchains [4], [32], [33], [34]. Casper [4] uses the *slashing* condition to penalize equivocating nodes, and some later works also incorporate the slashing condition into their protocol design [34], [33], [32].

Accountability can be quite useful when the number of faulty nodes exceeds the pre-specified threshold [35], [36]. The problem under such a case is also conceptually similar to BFT forensics [22], [23]. If a recovery process is involved under such a setting, the BFT protocol becomes recoverable [35]. Byzantine nodes are assumed to actively seek to break the safety of the protocol rather than the liveness. To do so, the liveness guarantee is usually weakened. A recent work also studies *accountable liveness* [37], where the behavior to break liveness can be accountable. This is achieved by slightly weakening the network model, where the network becomes asynchronous only periodically.

Our stage-1 protocol can be viewed as an accountable protocol. Since we assume that the fraction of faulty nodes in the entire system does not exceed the pre-specified assumption, we are able to *recover* from excessive faults without weakening the security guarantees or lowering the assumptions.

BFT from committee sampling. As mentioned in the introduction, there is a long line of research in Byzantine broadcast [15], [16], [38] and Byzantine agreement [14], [17], [15], [16], [13] protocols from committee sampling. The majority of the works focus on the adaptive adversary model mostly because the problem is theoretically interesting in such a model. King and Saia [13] presented a synchronous Byzantine agreement protocol with $O(n^{1.5})$ communication. Many follow-up works improve the communication complexity under the same model [14], [17], [15], [16].

To our knowledge, there are two practical BFT from committee sampling so far: Algorand [39] and Pando [40]. Algorand builds a BA protocol from committee sampling and uses it to build the blockchain. As mentioned in the introduction, the committee size has to be extremely large to be practical. Pando addressed the problem using the idea of sampling more than one committee to reach an agreement. In this way, the committee size can be made a degree of magnitude lower and the performance can be made high.

Different from existing works that mostly study BFT under the adaptive adversary model, we lower the assumption and study BFT under the static adversary model. As mentioned in the introduction, we believe our assumption is reasonable. Meanwhile, we address some practical issues such as handling faulty committees which are largely omitted by existing works.

Rational cryptographic protocols and RPD. The study of rational cryptographic protocols has a rich history, particularly in rational multiparty computation (MPC) [41], [42], [43], [44], [45], [46], [47]. Starting from Halpern and Teague [41] who studied secret sharing under rational players, subsequent works have explored bounded-round protocols [42], coalition resilience [44], efficient constructions [45], game-theoretic simulation-based fairness [46], [47], and Byzantine agreement under fully known utilities [48]. The RPD framework we adopt was also used to analyze rational attacks in Bitcoin such as selfish mining [26]. We choose RPD because its game between the protocol designer and the adversary aligns naturally with our protocol design analysis.

Byzantine altruistic rational (BAR) model. The BAR model [49], [50], [51] classifies nodes as Byzantine, altruistic, or rational. BAR-B [51] is a BFT protocol under this model where non-Byzantine nodes are rational, requiring $n \geq 3f+2$ nodes and using accountability to catch equivocating nodes via proof of misbehavior. Unlike BAR which considers both rational and Byzantine nodes simultaneously, we focus on the case where faulty nodes are either all Byzantine or all rational, which we believe better fits the committee sampling setting where only selected members participate in consensus.

Alive-but-corrupt (a-b-c) nodes. The notion of a-b-c nodes was first introduced by Malkhi, Nayak, and Ren [24]. In the context of ABC, the a-b-c nodes actively take actions to break the safety of the protocol. However, if they cannot break the safety, they will help preserve the liveness property. The definition of a-b-c nodes is aligned with our analysis under the RPD model. Namely, as the payoff of breaking liveness is negative or zero, rational attackers are not motivated to break the liveness of the system.

III. System Model and Preliminaries

We consider a protocol with n nodes $\mathcal{P} = \{p_1, \cdots, p_n\}$ running over authenticated channels. Among them, at most f are faulty. We assume $f < (1/3-\epsilon)n$, where $\epsilon \in (0, 1/3)$. We assume a static adversary $\mathcal{A}$ that corrupts a set of nodes before the system is up and running. Faulty nodes can be Byzantine and deviate arbitrarily from the protocol. Nodes that are not faulty are called correct nodes. Under our analysis of the fast path, we also consider *rational* nodes which try to maximize a set of utilities and *alive-but-corrupt* nodes which will not try to break liveness if they cannot break safety. We will define the utilities later.

We consider a partially synchronous network [20], i.e., there exists an unknown upper bound on message transmission and processing. Another interpretation is that there exists an unknown Global Stabilization Time (GST). After GST, the network is synchronous.

Our assumption on static adversary model is reasonable and building such a protocol is still practically challenging. First, such an assumption matches industrial examples such as DPoS and PoSA. Second, although it is theoretically more interesting to study BFT from committee sampling under stronger adversary model such as adaptive adversary (i.e., the adversary can corrupt any node in the middle of the system but the total number of corrupted nodes cannot exceed certain threshold), adaptive adversary is extremely hard to realize in practice, if not impossible. One particular thing we would like to mention is that protocols that might be closer to practice [19], [40] all assume a weakly adaptive adversary with *atomic sends*. Namely, if a correct node is corrupted while it is sending some message m to other nodes, m is

either sent to all receivers, or no correct receiver receives m. It is not clear whether such an adversary can even be realized in practice and how to enforce atomic sends. Last but not least, although practical BFT from committee sampling under adaptive adversary has been shown to achieve decent performance [19], [40], existing works largely omit how to handle faulty committees, even though the occurrence of faulty committees is rare.

We consider that the protocol operates in epochs. In each epoch e, a committee $\mathcal{P}_e$ of λ nodes is selected. Following prior works [14], [16], [15], we define a committee sampling oracle as our committee sampling function $\mathcal{F}_{OC}$. During epoch e, nodes can query $\mathcal{F}_{OC}$ whether it belongs to the committee for epoch $e+1$. When each committee member reveals its identity, all nodes can verify it using $\mathcal{F}_{OC}$. A committee is called *correct committee* if it contains fewer than one-third faulty nodes; otherwise, it is a *faulty committee*.

We follow prior works [2], [21], [3] and define the Byzantine quorum $\mathcal{RQ}(\mathcal{P})$ as a set of $2f+1$ nodes in $\mathcal{P}$.

Byzantine fault-tolerant protocol; atomic broadcast (ABC). We consider atomic broadcast [21] in this work, which is only syntactically different from conventional BFT protocols. Informally, the main difference between ABC and BFT is that ABC does not explicitly consider the role of the "clients" who submit transactions to the system. We define atomic broadcast as follows. ABC is specified by *a-broadcast* and *a-deliver* events. A node *a-broadcasts* blocks and all nodes *a-deliver* some blocks. An atomic broadcast protocol satisfies the following properties:

- **Safety**: If a correct node *a-delivers* a block b before *a-delivering* b', then no correct node *a-delivers* a block b' without first *a-delivering* b.
- **Liveness**: If a correct node *a-broadcasts* a block b, then all correct nodes eventually *a-deliver* b.

The API of ABC does not specify the order of a block. But in practice, most protocols associate each block with a *sequence number* or a *height*. We use height in this work to specify the order of a block.

Notations. We adopt the notion of Nakamoto Consensus (NC) [52], [53] to describe the order of blocks seen by correct nodes. The first block in the system is called the *genesis block*. Let C denote the chain of blocks seen by a correct node p_j. $\mathsf{C} = \perp$ if the only block in the chain is the genesis block. We say that C is *led* by its latest block, denoted as $\mathsf{head}(\mathsf{C})$. If p_j proposes a new block b according to C, b becomes the new head of the chain and we say that b *extends* C. Two blocks b and b' are called *conflicting blocks* if neither extends the other. We use the notation $\mathsf{C}_1||\mathsf{C}_2$ to represent the concatenation of two chains, where C_2 is appended to the end of C_1. We use canonical chain to denote the chain each replica *a-delivers*.

We will use some notations in this paper. We use Π_x^y to denote a protocol run by x nodes, the identities of which belong to y. For instance, $\Pi_\lambda^{\mathsf{com}}$ denote a protocol run by a committee of λ nodes.

Multivalued validated Byzantine agreement (MVBA). MVBA allows each node that has an input to agree on a value that satisfies a predicate Q known by all nodes [54]. The way we present our MVBA protocol follows that of [55]. In particular, MVBA is specified by *mvba-propose* and *mvba-decide* events. An MVBA protocol satisfies the following properties:

- **External validity**: Every correct node that terminates *mvba-decides* v such that $Q(v)$ holds.
- **Agreement**: If a correct node *mvba-decides* v, then any correct node that terminates *mvba-decides* v.
- **Integrity**: If a correct node *mvba-decides* v such that $Q(v)$ holds, then some node *mvba-proposed* v such that $Q(v)$ holds.
- **Termination**: If all correct nodes are activated and all messages sent among correct nodes have been delivered, then all correct nodes *mvba-decide*.

The predicate of MVBA can be signature-based or state-based. For the signature-based predicate, v is usually validated by some data (e.g., digital signatures) [54], [56]. For the state-based predicate, v can be locally validated without requiring signatures [55]. State-based MVBA is equivalent to *index ACS* [57] where the predicate is modeled as the notion of *party validation*. In this work, we use the state-based MVBA/Index ACS in our protocol.

Rational protocol design (RPD). The RPD framework was proposed by Garay, Katz, Maurer, Tackmann, and Zikas [25]. RPD provides a very nice and elegant view of cryptographic protocols from the perspective of game theory. The idea is to model a game between the *protocol designer* and the adversary via a two-phase Stackelberg game [58]. Such a model makes the reasoning of correctness similar to the analysis of cryptographic protocols. The idea is that based on reasonable assumptions on the utilities of rational players, a protocol that is *attack-payoff secure* can tolerate rational players. We use RPD to analyze our protocol under rational nodes. Later in Sec. VI, we will review the formal definitions of RPD.

IV. The Two-Stage Architecture and Overview of Our Work

In this section, we begin with discussing the generic two-stage architecture used by current DPoS and PoSA systems. We show that existing works all make additional and somewhat unrealistic assumptions. We then discuss the possible solutions and their challenges. Finally, we provide an overview of our work.

A. Existing Two-Stage Paradigm

Without loss of generality, we consider an epoch-based system. In each epoch e, a committee is selected. Each epoch has two stages, as summarized as follows.

- **Stage 1.** At the beginning of epoch e, a committee with λ nodes is selected. For now, we do not discuss how the nodes are selected. The committee members execute a $\Pi_\lambda^{\mathsf{com}}$ protocol to agree on the order of the blocks.

- **Stage 2.** After epoch e ends, nodes execute Π_n^{all}. The goal is for all nodes in $\mathcal{P}$ to learn the agreement results (the output of $\Pi_\lambda^{\mathsf{com}}$).

– **Initialization**: epoch $e \leftarrow 0$, parties $\mathcal{P}$, canonical chain $\mathsf{C} \leftarrow \perp$

Epoch e

– Select a committee of λ nodes $\mathcal{P}_e$
– **Stage 1.** If $p_i \in \mathcal{P}_e$, execute $\Pi_\lambda^{\mathsf{com}}$ to obtain the output $\mathsf{C}_{e,i}$ and update its local chain as $\mathsf{C} \leftarrow \mathsf{C}||\mathsf{C}_{e,i}$
– **Stage 2.** After epoch e times out, execute Π_n^{all}, which is instantiated as follows
  – If $p_i \in \mathcal{P}_e$, send $\mathsf{C}_{e,i}$ to all nodes in $\mathcal{P}$
  – If $p_i \notin \mathcal{P}_e$, upon receiving a valid C_e, set $\mathsf{C} \leftarrow \mathsf{C}||\mathsf{C}_e$

Fig. 2: Existing two-stage paradigm. Code for p_i. The protocol is correct if for any e, fewer than $\lfloor \frac{\lambda-1}{3} \rfloor$ nodes in $\mathcal{P}_e$ are Byzantine.

We present the workflow of such a paradigm more formally in Figure 2. The protocol is quite straightforward: the committee members execute a protocol $\Pi_\lambda^{\mathsf{com}}$ to form a chain $\mathsf{C}_{e,i}$, i.e., the order of certain number of blocks; all nodes in the system learn the results. If $\mathsf{C}_{e,i}$ is valid, e.g., a quorum certificate [59], [60], [61], [62] that is not conflicting with each node's canonical chain is available, each node in the system appends $\mathsf{C}_{e,i}$ to its canonical chain. In some industrial systems, stage 2 is a synchronization process where some nodes in the system are *light nodes* that only synchronize the block headers.

As mentioned in the introduction, known systems such as DPoS and PoSA protocols all adopt the above two-stage paradigm. These protocols often select the committee members via delegation of users' stake. A node is a candidate of the committee member if it owns a sufficiently large amount of stake. A set of nodes is then further selected according to certain rules from the candidates.

B. Analysis and Motivation of Our Work

Assuming a partially synchronous network, $\Pi_\lambda^{\mathsf{com}}$ can be instantiated with any secure BFT protocol, e.g., PBFT [2] or HotStuff [3]. It is thus not too difficult to see that the entire protocol is correct (safe and live) only if fewer than $\lfloor \frac{\lambda-1}{3} \rfloor$ nodes in $\mathcal{P}_e$ are faulty.

We argue that the paradigm introduces additional assumptions on top of the default $f < (1/3 - \epsilon)n$ assumption, the actual assumption can be overly strong. Let $\mathcal{P}_e$ be the committee of epoch e and let t_e be the number of faulty nodes in $\mathcal{P}_e$. To make the paradigm above correct, the assumption of the system is now:

$$f < (1/3 - \epsilon)n \textbf{ and for any } e, t_e \leq \lfloor \frac{|\mathcal{P}_e| - 1}{3} \rfloor.$$

To make the system achieve decent performance, the committee size has to be small enough, e.g., Binance has 21 nodes in each committee. If the committee members are not randomly selected, however, it becomes much easier to corrupt more than one-third nodes in the committee than corrupting one-third nodes in the entire system.

A case study on takeover of DPoS. A notable example of corrupting one-third nodes in the committee is the takeover attack to DPoS system of Steem blockchain [11]. Steem has a committee size of 21 nodes. Steem does not randomly select the committeees. Instead, all nodes in the system can cast votes to some nodes, and nodes with the largest number of votes become candidates of the committee. Among the 21 committee members, 20 of them are selected according to the votes from nodes in the system, and one is rotated periodically. In 2020, TRON founder purchased pre-mined coins from the company that launched the Steem blockchain (where pre-mined coins are supposed not to be used). On Mar 2, 2020, several nodes (owners of the pre-mined coins) delegated their voting power to one proxy node which then quickly cast votes to about 20 nodes in the system. Within one hour, the top 20 nodes were controlled by the attacker, who then created a hard fork, i.e., double spending. The original Steem players were forced to leave the system.

Thus, compared to controlling one-third of voting power in the entire system, it is arguably much easier to control one-third of the committee members.

Two-stage BFT from committee sampling. Our starting point is to randomly sample the λ committee members in each epoch. Namely, if we randomly sample the committee members, the problem of corrupting more than one-third faulty committee members is reduced to corrupting more than one-third nodes in the entire system, and the cost is significantly higher.

We identify two major challenges even if the committee members are randomly selected. First, there exists a trade-off of committee size and optimal resilience. Namely, to make sure that for any e, $t_e \leq \lfloor \frac{|\mathcal{P}_e|-1}{3} \rfloor$, the committee size λ has to be very large.

Table I reports the trade-off between committee size λ, fault tolerance threshold ϵ and the corresponding bound on f when uniformly random committees are drawn from a large n. The values are obtained by solving, for each λ and target failure probability δ, the smallest ϵ such that $\Pr[t_e > \lfloor(\lambda-1)/3\rfloor] \leq \delta$ when each committee member is independently faulty with probability $1/3 - \epsilon$ (i.e., the binomial tail). If we set λ to 100, even at the much weaker target $\delta = 10^{-6}$ we need $\epsilon \approx 0.19$ and hence $f < 0.15n$. If we instead push δ down to 10^{-9}, it forces $\epsilon \approx 0.22$ and $f < 0.11n$. In all such regimes, the system fails to achieve optimal resilience (i.e., $f < n/3$).

There exists some potential solutions. For instance, one can alternatively let the committee members run the $\Pi_\lambda^{\mathsf{com}}$ protocol by keeping the committee size small. After an agreement is reached, *all* nodes in the system run another BFT protocol to agree on the results. To achieve decent performance, all nodes can only agree on the results, e.g., a quorum certificate. We show an attempt in Appendix A. Such a solution, however,

λ	$\delta = 10^{-6}$		$\delta = 10^{-9}$		$\delta = 10^{-12}$	
	ϵ	f/n	ϵ	f/n	ϵ	f/n
50	0.243	0.091	0.277	0.057	0.297	0.036
100	0.187	0.146	0.223	0.111	0.248	0.086
500	0.094	0.239	0.117	0.217	0.134	0.199
1000	0.068	0.266	0.085	0.249	0.098	0.235
3000	0.040	0.293	0.050	0.283	0.059	0.275

TABLE I: Trade-off between the committee size λ, the resilience parameter ϵ, and the corresponding fault bound $f < (1/3 - \epsilon)n$ for several target per-epoch faulty-committee probabilities δ. Values are computed from the binomial tail under uniform random sampling with $n \to \infty$.

still requires frequent query of the protocols by all nodes in the system. The performance and scalability of the system are thus expected to be not that satisfactory.

Second, we would like to argue that all known BFT protocols from committee sampling focus on the protocol design and analysis of practical setup of the parameters such as λ and ϵ. To our knowledge, no existing works consider the handling of faulty committees, i.e., if any of the sampled committee has more than one-third faulty nodes, the entire protocol fails to achieve its security goals. In practical systems, even if the probability of having a faulty committee is low (e.g., $1-10^{-9}$), it is still possible that some selected committee is faulty. Thus, we will need to consider *recovery* under faulty committees.

- **Initialization**: epoch $e \leftarrow 0$, parties $\mathcal{P}$, canonical chain $\mathsf{C} \leftarrow \perp$, equivocation set $ES \leftarrow \emptyset$, configuration $c \leftarrow 0$, abort status $abort \leftarrow false$

Epoch e

- Select a committee of λ nodes $\mathcal{P}_e$
- **Stage 1.** If $p_i \in \mathcal{P}_e$, execute $\Pi_\lambda^{\mathsf{com}}$ to obtain the output $\mathsf{C}_{e,i}$
- **Stage 2.** After epoch e times out, execute Π_n^{all}, which is instantiated as follows
  - If $p_i \in \mathcal{P}_e$, send $\mathsf{C}_{e,i}$ to all nodes in $\mathcal{P}$
  - For any node, wait to receive $\mathsf{C}_{e,j}$ from members in $\mathcal{P}_e$ and perform an equivocation detection
    - Abort upon detecting an equivocation. Agree on a set of parties that have equivocated, add these parties to ES and enter a new configuration $c \leftarrow c + 1$. Set $abort \leftarrow true$ upon successful entering the new configuration. //slow path
    - Otherwise, upon receiving a valid C_e, set $\mathsf{C} \leftarrow \mathsf{C}||\mathsf{C}_e$ and *a-finalize* C upon certain criteria. //fast path

Fig. 3: The revamped two-stage paradigm. Code for p_i. The changes on top of Figure 2 is highlighted in blue.

C. Overview of Our Work

In a nutshell, our solution revamps the two-stage design by involving all nodes to participate in the protocol in the worst case. In this way, we can exploit the nice feature of $f < (1/3 - \epsilon)n$ to build a practical protocol. To do so, our technical pathway consists of two steps.

The two-stage BFT: A refined definition of BFT. we refine the definition of BFT/ABC (Sec. III) called *two-stage BFT*. Different from conventional definition that only involves the *a-broadcast* and *a-deliver* events, our two-stage BFT notion has three events: *a-broadcast*, *a-deliver*, and a new *a-finalize* event. The *a-broadcast* and *a-deliver* events are used to capture the behavior of protocol $\Pi_\lambda^{\mathsf{com}}$. Namely, since our goal is to lower the committee size to make our protocol practical, we can simply relax the requirement on $\Pi_\lambda^{\mathsf{com}}$ and allow forks to occur. On top of that, the *a-finalize* event denotes the *finalization behavior* in blockchains, and we aim to achieve the same safety guarantees for the *a-finalize* event as conventional BFT. In this way, our refined definition can both capture the security guarantees of conventional BFT and also enable more flexibility in the design of $\Pi_\lambda^{\mathsf{com}}$.

Our concrete two-stage design. Under the guidance of our refined two-stage BFT definition, we build a new two-stage protocol as shown in Figure 3. One nice feature of our new definition is that we can build $\Pi_\lambda^{\mathsf{com}}$ from almost any BFT protocols, e.g., PBFT and HotStuff. In our concrete construction, we only need to make some minor modifications on the output of the protocol so that: 1) the agreement result can be validated by all nodes; 2) each node that proposes a block is made *accountable*, i.e., if it equivocates, it can be caught. Both requirements can be easily achieved, and most protocols already achieve both requirements.

In stage 2, there are two paths. In the fast path, all nodes directly enter the next epoch without running another subprotocol. In the slow path, all nodes in the system run a protocol only when equivocation of faulty committee members has been detected. Instead of having all nodes run an expensive protocol on the message payload (i.e., the blocks of transactions), our stage-2 protocol agree on a set of nodes that have equivocated. Once an agreement is reached on a set of faulty committee members, all correct nodes are guaranteed to enter a new configuration by removing these nodes from the system.

Since our fast path does not require a dedicated protocol, it is extremely fast. Meanwhile, since the stage-2 protocol is used to agree on a set of parties with equivocation proofs, it is very lightweight, and can recover from faulty committees.

Analysis under weaker adversary and RPD. We further distinguish two types of faulty nodes: 1) faulty nodes are rational; 2) faulty nodes are Byzantine and fail arbitrarily. We show that, under the guidance of the RPD framework [25] and an analysis of the utilities of rational players, rational players are in fact the same as *alive-but-corrupt* nodes [24]. Also, our refined notion of the two-stage BFT is also attack-

payoff secure. Thus, under rational nodes or alive-but-corrupt adversaries, all nodes will only trigger the fast path.

Meanwhile, our stage-2 protocol handles equivocation in a highly efficient way. Even when all faulty nodes are Byzantine, our protocol is still very efficient.

V. The rBFT Protocol

We are now ready to present our protocol. In this section, we first define the new two-stage BFT notion, and then show our protocol.

A. Definition of Two-Stage BFT

As mentioned in Sec. IV, we provide a refined notion of BFT called two-stage BFT. Specifically, there are three events: *a-broadcast*, *a-deliver*, and *a-finalize*. A secure two-stage BFT satisfies the following properties with probability $1 - \mathsf{negl}(\kappa)$, where $\mathsf{negl}(\kappa)$ is a negligible function in security parameter κ.

- **Universal agreement**: In any epoch e, if a committee $\mathcal{P}_e$ is correct and a correct node *a-delivers* a block b before *a-delivering* b', then no correct node *a-delivers* a block b' without first *a-delivering* b.
- **Fast finality**: In any epoch e, if a committee $\mathcal{P}_e$ is correct and a correct node *a-delivers* a block b before *a-delivering* b', then no correct node *a-finalizes* b' without first *a-finalizing* b.
- **Safe finalization**: If a correct node *a-finalizes* a block b before *a-finalizing* b', then no correct node *a-finalizes* b' without first *a-finalizing* b.
- **Identifiable abort**: If one correct node aborts, all correct nodes eventually agree upon the identity of at least one equivocating node and remove it from the system.
- **Liveness**: If a correct node *a-broadcasts* a block b, then all correct nodes eventually *a-finalize* b.

Compared to the conventional BFT definition, we now define five properties instead of two. Specifically, universal agreement, fast finality, and safe finalization properties are safety properties, and the rest two are liveness properties. Roughly speaking, the *a-finalize* event in our definition plays the role of the *a-deliver* event in conventional BFT. In contrast, *a-deliver* in our definition is relaxed to describe the per-epoch output of $\Pi_\lambda^{\mathsf{com}}$, and forks are allowed.

For safety properties, both universal agreement and fast finality properties are defined under the assumption that a committee $\mathcal{P}_e$ for an epoch e is a correct committee. These two properties roughly match that in the vanilla two-stage BFT, i.e., if the committee is correct, all nodes only need to passively listen to the agreement results. We distinguish the two properties mainly because the fast finality is used to capture the behavior in the fast path of stage 2. Also, the safe finalization property corresponds to the safety property as in conventional BFT, except that the event is changed to *a-finalize* instead of *a-deliver*.

For liveness, the liveness property is almost identical except that all correct nodes *a-finalize* blocks instead of *a-delivering* blocks. Meanwhile, the identifiable abort property is used to capture the behavior of configuration change, where nodes remove some equivocating nodes from the system. This property is useful in the design of stage 2.

It can be seen that there does not exist trivial instantiation for the properties above, mainly because the safe finalization property and the liveness property roughly match the safety and liveness properties of conventional BFT. We introduce the additional properties to better capture the two-stage behavior of BFT from delegated committee.

B. The Protocol

We are now ready to present our rBFT protocol. We show the pseudocode of rBFT in Figure 4. The pseudocode that instantiates the contents in blue of Figure 3 is also highlighted in blue. The detailed workflow is described below.

Stage 1: Agreement on the order of blocks. The selected committee members execute $\Pi_\lambda^{\mathsf{com}}$ to agree on the order of certain number of blocks. We can use any BFT protocol to instantiate $\Pi_\lambda^{\mathsf{com}}$. After an agreement on the order of some block $b \in \mathsf{C}_{e,i}$ is reached, each committee member *a-delivers* b. We also require that the BFT protocol is *verifiable* and any block proposer is *accountable*. For the protocol to be verifiable, an evidence is provided for any output $b \in \mathsf{C}_{e,i}$, proving that b has been *a-delivered*. For a block proposer to be accountable, it needs to include a digital signature for its block proposal.

Many BFT protocols already achieve both requirements, e.g., HotStuff [3] and PBFT [2]. We thus use the notion of QC, i.e., $\mathcal{RQ}(\mathcal{P}_e)$, to denote the evidence that some block is *a-delivered*. For protocols that do not support verifiability of the output, we can simply add another step of communication to create the QC. Namely, after an agreement is reached on block b, each node creates a signature on the hash of b and sends the signature to all nodes in $\mathcal{P}_e$. Two-thirds of matching signatures form a QC. Additionally, each block proposer just includes a signature for its block proposal, so it becomes accountable.

It is worth mentioning that we can also make the behavior of non-leaders accountable, just like what the slashing conditions in Ethereum or BFT forensics [22], [23] are doing. We omit the details on how to identify faulty nodes based on the received agreement results.

Stage 2: A fast path for rational nodes and a slow path for Byzantine nodes. After epoch e times out, all nodes enter stage 2. Each node starts a timer Δ and synchronizes the outputs $\mathsf{C}_{e,j}$ from the committee members. The outputs are stored in a set S. Then, there are two paths.

▷ **Fast path.** If $|S| \leq 1$, the protocol is the same as existing paradigm shown in Figure 2, i.e., each node updates its canonical chain as $\mathsf{C}||\mathsf{C}_e$ (where $S = \mathsf{C}_e$) and proceeds to epoch $e + 1$.

▷ **Slow path.** If $|S| \geq 2$, there are at least two valid outputs in $\Pi_\lambda^{\mathsf{com}}$. In this case, at least one node in $\mathcal{P}_e$ has equivocated. In this case, each node that has $|S| \geq 2$ first extracts equivocation proofs σ_ℓ. Namely, two conflicting signatures by the same node p_ℓ proves that p_ℓ has equivocated, e.g., it *a-broadcasts*

rBFT

- **Initialization**: epoch $e \leftarrow 0$, parties $\mathcal{P}$, canonical chain $\mathsf{C} \leftarrow \perp$, equivocation set $ES \leftarrow \emptyset$, configuration $c \leftarrow 0$, abort status $abort \leftarrow false$, lock status $lock \leftarrow \perp$
- $\Pi_\lambda^{\mathsf{com}}$ is required to be verifiable
- Define the predicate Q of MVBA as $Q(\sigma_j, e, h, \delta) \equiv$ (σ_j consists of valid equivocation proofs for a set of nodes $\mathcal{P}_{equiv}$ for epoch e **and** δ is a valid proof for h **and** (h, δ) is consistent with $lock$, i.e., $lock = \perp$ or h extends or equals $lock.h$)

Epoch e

- Select a committee of λ nodes $\mathcal{P}_e$ from $\mathcal{P}$
- **Stage 1.** If $p_i \in \mathcal{P}_e$
  - Execute $\Pi_\lambda^{\mathsf{com}}$ and output $\mathsf{C}_{e,i}$ //Each block in $C_{e,i}$ is *a-delivered* in $\Pi_\lambda^{\mathsf{com}}$
  - //As a leader of $\mathcal{P}_e$, propose blocks that extend its C and provide $\mathcal{RQ}(\mathcal{P}_e)$, proving that the head of C is *a-delivered*
  - //As a non-leader of $\mathcal{P}_e$, vote for a proposed block if the block is consistent with its own canonical chain
- **Stage 2.** After epoch e times out, execute Π_n^{all}, which is instantiated as follows
  - If $p_i \in \mathcal{P}_e$, send $\mathsf{C}_{e,i}$ to all nodes in $\mathcal{P}$
  - For any p_i, starts a timer Δ
  - Upon receiving a valid $\mathsf{C}_{e,j}$ from p_j, $S \leftarrow S \cup \mathsf{C}_{e,j}$
  - After Δ times out
    - If $|S| \geq 2$
      - Select any two values $\mathsf{C}_e, \mathsf{C}'_e$ from S and extract equivocation proofs σ_ℓ for the set of faulty nodes $\mathcal{P}_{equiv} \leftarrow \{p_\ell, \cdots\}$ in epoch e
      - Send σ_ℓ to all nodes in $\mathcal{P}$
    - Else if $|S| = 1$
      - Parse S as C_e and set $C \leftarrow C || C_e$
      - For any block b in $C_{e'}$ where $e - e' \geq \kappa$ and $b \in C$, *a-finalize* b and all its ancestors.
    - Else, do not update C and proceed to epoch $e + 1$
  - Upon receiving a valid σ_ℓ
    - $ES \leftarrow ES \cup \sigma_\ell$
    - If an (ABORT) message has not been sent yet, execute $(h, \delta) \leftarrow \mathsf{Validate}(\mathsf{C})$ and send $(\text{ABORT}, c, h, \delta)$ to all nodes
    - Set $abort \leftarrow true$
  - Upon receiving $n - f$ $(\text{ABORT}, c, *, *)$ messages
    - Wait until $ES \neq \emptyset$ and parse ES as σ_i
    - Set $lock \leftarrow (h, \delta)$, where (h, δ) correspond to the highest block b in the (ABORT) messages such that b is not conflicting with its *a-finalized* block
    - If MVBA_c has not been started yet, *mvba-propose* (σ_i, e, h, δ) to MVBA_c where e is the epoch of the block corresponding to h
  - After MVBA_c outputs (σ_j, e, h, δ)
    - Obtain the set of faulty nodes $\mathcal{P}_{equiv}$ based on σ_j. Set $\mathcal{P} \leftarrow \mathcal{P} \backslash \mathcal{P}_{equiv}$ and set $c \leftarrow c + 1$
    - Obtain C' based on (h, δ) such that the block corresponding to (h, δ) is the head of C'.
    - Roll back to epoch e by setting $\mathsf{C} \leftarrow \mathsf{C}'$
    - Remove the equivocation proofs from ES for $\mathcal{P}_{equiv}$, and set $abort \leftarrow false$

Fig. 4: rBFT protocol. Code for p_i. $h, \delta \leftarrow \mathsf{Validate}(\mathsf{C})$ is a function, where h is the hash of the highest *a-finalized* block b in C and δ is a proof that b is *a-delivered*.

two different blocks with the same height. Then, p_i sends the proofs to all nodes in $\mathcal{P}$.

Upon receiving any valid equivocation proof σ_ℓ, each node p_i adds σ_ℓ to local set ES and computes rollback point (h, δ), where h is the hash of the highest *a-finalized* block in its current canonical chain and δ is the corresponding delivery proof. The node p_i then aborts and sends $(\text{ABORT}, c, h, \delta)$ to all nodes.

When a node receives $n-f$ abort messages for configuration c, it waits for $ES \neq \emptyset$ and selects equivocation proof σ_i from ES. The node sets $lock$ as (h, δ) where (h, δ) denote the *highest* chain from the (ABORT) messages. The consistency check in Q ensures that any (h, δ) decided by MVBA corresponds to a block that is either identical to or an extension of the block locked by the node. When $lock = \perp$ (i.e., the node starts), the consistency check is trivially satisfied. It then *mvba-proposes* (σ_i, e, h, δ) to MVBA_c, where σ_i are the equivocation proofs. After MVBA_c outputs (σ_j, e, h, δ), each node removes the set of nodes that have equivocated, roll back its state. After that, each node resumes its normal operation, and enters a new configuration $c + 1$.

Finalization. So far, we still have not defined when a block

b can be *a-finalized*. Specifically, given a block b and a chain C such that b is the head of C_e, node p_i *a-finalizes* b if its canonical chain is extended to $\mathsf{C}_{e'}$ and $e' - e = \kappa$. Block b and all its ancestors in C are said to be *a-finalized*.

On the choice of Δ. Δ only affects the fast path. A large Δ slows it down even when no faults occur. A small Δ leaves S empty, and the epoch idles. After GST, any Δ above the message delivery bound avoids idle epochs.

Complexities. We can use any BFT protocol for $\Pi_\lambda^{\mathsf{com}}$ and any MVBA protocol for stage 2. In our implementation, we use PBFT [2] for $\Pi_\lambda^{\mathsf{com}}$ and Speeding-MVBA [63] as the MVBA protocol. Let L be the size of each block, $\Pi_\lambda^{\mathsf{com}}$ achieves a message complexity of $O(\lambda^2)$ and a communication complexity of $O(L\lambda^2 + \kappa\lambda^3)$. Meanwhile, in stage 2, all committee members send their *a-delivered* chain to all nodes in the system. The message complexity is $O(\lambda n)$ and the communication complexity is $O(L\lambda n)$. In the fast path, there is no additional cost. In the slow path, the input of MVBA of each epoch consists of up to $O(\lambda)$ equivocation proofs, i.e., the size of the input is $O(\lambda\kappa)$. Thus, the message complexity is $O(n^3)$ and the communication complexity is $O(\kappa\lambda n^2 + \kappa n^3)$.

C. Discussion and Proof of Correctness

Discussion on the new *a-finalize* event. The probability that the order of a finalized block is reversed is negligible. Namely, if the size of each committee $|\mathcal{P}_e| = O(\lambda)$, where λ is considered a security parameter, the probability that the committee has fewer than one-third faulty nodes is $1-\mathsf{negl}(\kappa)$ (theoretically). It is then not too difficult to see that under a reasonable assumption of κ, the probability that the order of a finalized block is also $1 - \mathsf{negl}(\kappa)$.

We would like to emphasize that under practical assumptions where λ is small enough (e.g., < 100), the probability that the committee has more than one-third faulty nodes cannot be simply omitted. For instance, previous work shows that if $\lambda = 100$ and $\epsilon = 0.091$ (which is even better than the $\epsilon = 0.12$ example for the Algorand case mentioned in the introduction), the probability that a committee has more than one-third faulty nodes is 10^{-4}, which is already quite frequent in practice.

In contrast, our setting can well mitigate the issue caused by selecting a small committee size. Let the failure probability be p. The probability that the order of a finalized block becomes $1 - p^\kappa$. Thus, if κ is set as a *safe* value, the probability that a finalized block is reversed becomes significantly lower.

Also, since we run an MVBA protocol to agree on both the equivocating nodes and a hash h that denotes a *a-delivered* chain, it is not too difficult to see that safe finalization property is still satisfied.

Sketch of proofs. We show the proofs in detail in Appendix C and provide a sketch here. Universal agreement and fast finality all require a committee to be correct, and thus the correctness of the two properties are quite straightforward. Safe finalization is achieved by our stage-2 protocol. Namely, a block is *a-finalized* only when the canonical chain is extended in κ new epochs. Such finality guarantee is conceptually similar to consensus mechanisms such as Nakamoto consensus, and correctness is bounded by the probability that a correct committee is selected. Namely, forks in the canonical chain are formed only when a faulty committee is selected. Thus, as the canonical chains are extended in several epochs, the probability that all the forks are grown by blocks proposed by faulty committees degrades fast. Thus, safe finalization holds probabilistically, and we show that it holds with $1 - \mathsf{negl}(\kappa)$ probability. For identifiable abort, the property holds mainly because each node aborts upon receiving an equivocation proof and the proof is sent to all nodes in the system. Thus, the property holds. Finally, liveness is guaranteed eventually since correct committees are guaranteed to be selected. Our protocol guarantees that after GST, every node will extend its canonical chain in stage 2.

VI. Analysis of rBFT under Alive-but-Corrupt Nodes and Rational Nodes

We analyze the feasibility of $\Pi_\lambda^{\mathsf{com}}$ under the assumption that faulty nodes are rational, i.e., they seek to maximize their utilities rather than breaking the security properties of the system. Our finding is that under our analysis of the utilities, rational nodes are *alive-but-corrupt* nodes, i.e., if they cannot break the safety of the system, they will not try to break liveness. Under this assumption, we conclude that our refined notion of two-stage BFT is also attack-payoff secure. Thus, in our rBFT protocol, if all faulty nodes are rational, the slow path will never be triggered.

In this section, we first review the Rational Protocol Design (RPD) framework, and then analyze our protocol.

A. Review of the Rational Protocol Design (RPD)

RPD builds on simulation-based security definition by Canetti [64]. Simulation-based definitions use the real-world/ideal-world paradigm. In the real world, all nodes execute a protocol Π. In the ideal world, all nodes securely provide their input to the ideal functionality $\mathcal{F}$ and obtain the output. Protocol Π securely realizes $\mathcal{F}$ if, for any real-world adversary $\mathcal{A}$ attacking Π, there is an ideal-world adversary $\mathcal{S}$, i.e., the simulator, that emulates $\mathcal{A}$'s attack. The simulation is successful if no distinguisher $\mathcal{Z}$, i.e., environment, which interacts with all parties, can distinguish between the two worlds. All entities involved in the protocol execution are described in *interactive Turing machines*, denoted as ITM. $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ denotes the random variable corresponding to the output of the environment $\mathcal{Z}$ witnessing the execution of protocol Π against adversary $\mathcal{A}$.

The attack game $\mathcal{G}_\mathcal{M}$. The RPD framework models a zero-sum extensive game between two rational players: the protocol designer D and the attacker A. The game is parameterized by a (multi-party) functionality $\mathcal{F}$ known to both D and A. The goal of the designer is to design a protocol Π for realizing $\mathcal{F}$ from the set of all probabilistic and polynomial-time (PPT) computable protocols. The goal of the attacker is to choose a

PPT adversary $\mathcal{A}$ to attack protocol Π, so Π violates certain security properties of $\mathcal{F}$.

The game is a Stackelberg game [58]. In the Stackelberg game, there are two players: a leader and a follower. The game involves two steps. First, the leader plays the game, and the follower is informed about the leader's move. Second, the follower plays the game, and the game terminates. In the context of RPD, the leader and the follower are the protocol designer D and the attacker A, respectively. The attack game $\mathcal{G}_{\mathcal{M}}$ is represented as (Π, A), where $\mathcal{M}$ is an *attack model*. The attack model $\mathcal{M}$ is denoted as $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$, where $\mathcal{F}$ is the ideal functionality mentioned above, $\langle\mathcal{F}\rangle$ is a *relaxed* functionality that allows for certain security breaches (the attacker wants to provoke), and v is the *utilities* assigned to certain actions. In $\mathcal{G}_{\mathcal{M}}$, D first sends Π to A. Based on Π, A then chooses $\mathcal{A}$ and the game terminates.

The utilities. Based on the attack model $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$, the utilities can be defined. The idea is to model the attacks in the ideal world using the relaxed functionality $\langle\mathcal{F}\rangle$. Informally, the relaxed functionality is vulnerable to some attacks such that some security properties of $\mathcal{F}$ are violated. Based on each security property the attacker A may provoke, one can define a real-valued *payoff*. This is modeled as an event-based utility. In particular, let $(E_1, \cdots, E_\ell)$ denote a vector of disjoint events defined on the views (of the simulator $\mathcal{S}$ and environment $\mathcal{Z}$) in the ideal experiment corresponding to the attacker's utility. Each event E_i is assigned a real payoff γ_i. The ideal expected payoff of a simulator $\mathcal{S}$ is then defined as

$$U_I^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) = \sum_{E_i \in \vec{E}, \gamma_i \in \vec{\gamma}} \gamma_i \Pr[E_i],$$

where the probabilities are taken over random coins of $\mathcal{S}$, $\mathcal{Z}$, and $\langle\mathcal{F}\rangle$.

The *maximal payoff* of an adversarial strategy $\mathcal{A}$ is defined as the expected payoff when executing a protocol Π with the adversary's preferred environment. Formally, it is defined as

$$\hat{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) = \sup_{\mathcal{Z}\in\mathsf{ITM}} \left\{ \inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}} \left\{ U_I^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) \right\} \right\},$$

where $\mathcal{C}_{\mathcal{A}}$ denotes the class of simulators that are "good" for $\mathcal{A}$, i.e., $\mathcal{C}_{\mathcal{A}} = \{\mathcal{S} \in \mathsf{ITM} | \forall \mathcal{Z} : \mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}\}$.

Attack-payoff security. Based on the utility, the notion of *attack-payoff security* can be defined for a protocol Π given an attack model $\mathcal{M}$. Informally, a protocol is attack-payoff secure if the utility of the best adversary against Π is the same as that of the best adversary against a $\mathcal{F}$-hybrid "dummy" protocol (forwards all inputs to and outputs from the functionality $\mathcal{F}$).

Definition 1. *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack model and let Π be a protocol that realizes $\langle\mathcal{F}\rangle$. Π is attack-payoff secure in $\mathcal{M}$ if $U^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) \leq U^{\phi^{\mathcal{F}},\langle\mathcal{F}\rangle}(\mathcal{A}) + \mathsf{negl}(\kappa)$, where $\phi^{\mathcal{F}}$ is a dummy $\mathcal{F}$-hybrid protocol that forwards all inputs and outputs of $\mathcal{F}$, and $\mathsf{negl}(\kappa)$is a negligible function related to κ.*

B. Analyzing the Scores for Breaking Safety, Liveness, and Cost of Corruption

A crucial step to connect the model with practical design is to analyze the payoff of breaking the security properties of the protocol. In the blockchain context, let γ_s, γ_ℓ, and $\gamma_{\mathcal{I}}$ denote the cost of breaking safety, liveness, and cost of corruption nodes $\mathcal{I}$ for $\Pi_\lambda^{\mathsf{com}}$. Since our analysis mainly aims to support the claim that the fast path is always triggered under rational nodes, we limit ourselves to analyze the security properties of conventional BFT, assuming that $\mathcal{P}_e$ runs a conventional BFT as $\Pi_\lambda^{\mathsf{com}}$.

The safety payoff γ_s. In the context of blockchain, the payoff for breaking safety is the so-called *double spending* capability. Namely, at the same height j, if both block b_j and b'_j are output by the functionality $\langle\mathcal{F}_{ABC,e}\rangle$, the transactions in both b_j and b'_j are *a-delivered*. The values of all conflicting transactions $\boldsymbol{tx}$ of b_j and b'_j are considered the concrete payoff for γ_s. Thus, $\gamma_s > 0$. We consider that γ_s is upper bounded by some value.

The liveness payoff γ_ℓ. When liveness is broken, some blocks in the input are not included in the output. In the context of blockchain, the payoff is related to attestation incentives [65], [66] and transaction fees. Attestation incentives are available in systems such as Ethereum [67]. In Ethereum, each block includes a set of votes called attestations. After the block is *a-delivered*, the nodes (called validators) corresponding to the votes receive *rewards*. If a node does not send its vote while it is supposed to send one (i.e, its vote is not included in an *a-delivered* block for a period of time), it suffers from *penalties*. Meanwhile, almost all blockchains have the mechanism of transaction fees. Namely, if a block is *a-delivered*, the proposer of the block receives some rewards. Thus, we have the following.

- With attestation penalty, the liveness payoff $E[\gamma_l] < 0$;
- Without attestation penalty, the liveness payoff $E[\gamma_l] \approx 0$.

Thus, the payoff of breaking liveness is either zero or negative.

Note that in blockchains, other factors are also related to the liveness payoff. For instance, one may argue that once the adversary controls a sufficiently large fraction of nodes, it has full control of what transactions can be included in the output. Meanwhile, off-chain services, e.g., oracles or market manipulations, are also related to whether some transactions are included in the output. We do not consider these factors in our study, mainly because our work focuses on the BFT protocols, i.e., whether the security properties are affected under rational nodes. We consider the studies of other factors an interesting future research direction.

The costly corruption payoff $\gamma_{\mathcal{I}}$. In most blockchains, there is a cost associated with corrupting a node. In Proof-of-Work (PoW) based blockchains, the cost is the computational resources. In Proof-of-Stake (PoS) based blockchains, the cost is the *stake* the adversary pays to corrupt a node (i.e., the bribe in bribing attacks [68], [69]). Meanwhile, many protocols have designed a slashing mechanism that, once a node is caught

after equivocating, it will be removed from the system. Thus, we consider $\gamma_{\mathcal{I}} > 0$. We use $\gamma_{\$}$ to denote the cost of corrupting any node.

For $|\mathcal{I}| = t$, by simplifying the notations of γ_ℓ and $\gamma_{\mathcal{I}}$, we simplify the payoff vector as:

$$\vec{\gamma} = (\gamma_s, 0, -t \cdot \gamma_{\$}).$$

C. Feasibility of Attack-Payoff Secure Protocol

Based on the discussion above, we consider two cases: $\gamma_s \leq \lfloor \frac{\lambda}{3} \rfloor \gamma_{\$}$; $\gamma_s > \lfloor \frac{\lambda}{3} \rfloor \gamma_{\$}$.

Case 1: $\gamma_s \leq \lfloor \frac{\lambda}{3} \rfloor \gamma_{\$}$**.** In this case, the safety payoff no higher than the cost of corrupting $\lfloor \frac{\lambda}{3} \rfloor$ nodes. It is thus not too difficult to see that any provably secure BFT protocol is also attack-payoff secure. We show the following theorem and prove the correctness in Appendix B.

Theorem 2. *Assuming $\gamma_s \leq \lfloor \frac{\lambda}{3} \rfloor \gamma_{\$}$, running any BFT protocol as $\Pi_\lambda^{\mathsf{com}}$ is attack-payoff secure in the attack model $(\mathcal{F}_{ABC}, \langle \mathcal{F}_{ABC,e} \rangle, v^{\vec{\gamma}})$.*

In this case, our new definition of two-stage BFT is also attack-payoff secure under the RPD framework. In case 1, since we can run any BFT protocol as $\Pi_\lambda^{\mathsf{com}}$ to make the protocol attack-payoff secure, the universal agreement and fast finality property, and liveness can directly lead to an attack-payoff secure protocol.

Case 2: $\gamma_s > \lfloor \frac{\lambda}{3} \rfloor \gamma_{\$}$**.** The safety payoff is higher than the cost of corrupting $\lfloor \frac{\lambda}{3} \rfloor$ nodes. $\Pi_\lambda^{\mathsf{com}}$ alone is no longer attack-payoff secure. In particular, an adversary has a positive expected utility from corrupting a committee and breaking the safety of $\Pi_\lambda^{\mathsf{com}}$. We show that our two-stage design of rBFT can address this case.

Considering the fact that rational nodes are incentivized to break safety, there is no guarantee on the safety property of conventional ABC, i.e., if a correct node *a-delivers* a block b before *a-delivering* b', then no correct node *a-delivers* a block b' without first *a-delivering* b. Our *a-finalize* event, the safe finalization, and the new liveness properties capture case 2. Namely, safety of the entire system is guaranteed by safe finalization, and the liveness of the entire system is guaranteed by the new liveness property. Also, if safety of $\Pi_\lambda^{\mathsf{com}}$ is violated, e.g., a correct node *a-delivers* b and another correct node *a-delivers* b' with the same height, some correct node(s) might abort. In this case, some equivocating nodes can be identified and the safety payoff is much lower. Specifically, if one node is identified, the safety payoff becomes $\gamma_s - \gamma_{\$}$. For any $t < \lambda$, any adversary corrupting t parties can be simulated by querying the ideal functionality at most t times. Thus, the utility of the adversary is not positive anymore.

It is also not too difficult to see that under our analysis, rational nodes are identical to alive-but-corrupt nodes, mainly because they can never break the safety of the system.

VII. Implementation and Evaluation

Implementation. We implement rBFT in C++ and integrate it into the open-source blockchain platform FISCO-BCOS [70]. To ensure secure communication, we use Boost Beast as the communication library, OpenSSL TLS for authenticated channels, SHA256 for hash functions, and ECDSA for digital signatures. We use VRF [71] to elect the committee. We implement PBFT as $\Pi_\lambda^{\mathsf{com}}$ and Speeding-MVBA [63] as the MVBA protocol. Our implementation has over 7,300 LOCs. To demonstrate the practical viability of rBFT for real-world deployment, each node in our experimental setup runs the complete blockchain stack, including both consensus modules, storage, transaction execution, and synchronization modules. For our baseline comparison, we compare rBFT with running PBFT among all n nodes.

Evaluation setup. Our performance evaluation is conducted in a cloud environment using a cluster of up to 100 virtual machines (VMs). We use each instance to run five nodes so that the resource allocation is consistent across all experiments. Each VM is configured with 8 virtual CPUs, 32 GB of memory. The network bandwidth between any two VM instances is up to 500 Mbps. We run each experiment such that 10,000 new blocks are *a-broadcast* and *a-finalized* and report the result.

We vary several parameters to assess the performance. Specifically, we vary the number of nodes n with a fixed batch size of 2,000. We also vary the committee size $\lambda \in \{4, 10, 21\}$, and use the notation rBFT (λ) to distinguish between these setups. To trigger the actual transaction processing flow, the client continuously submits ERC-20 [72] "Transfer" transactions to the network. Each transaction has 254 bytes. For simulating adversarial conditions, we randomly select 10%, 20%, and 30% of the nodes as Byzantine.

Evaluation metrics. We evaluate protocol performance by throughput and latency, which are measured by varying the client's transaction submission rate from 500 tx to 20,000 tx per second. Throughput is measured as the number of non-repetitive transactions processed during the experiment. The latency is measured as the duration from when the client sends transactions until the corresponding on-chain receipt is received.

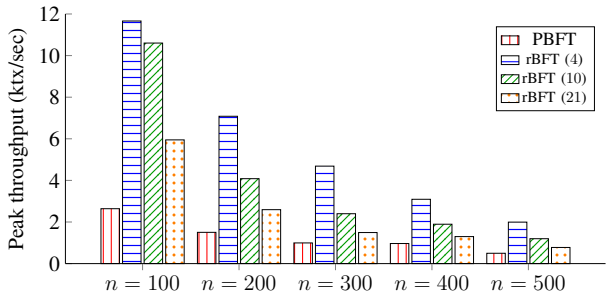
Main evaluation questions and results. Our experiments focus on the following questions.

Q1 : What is the performance (throughput and latency) when the fast path is triggered?

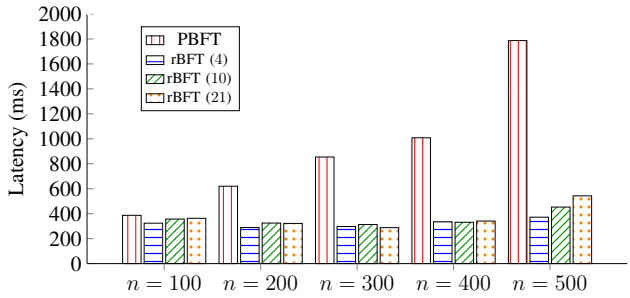
Q2 : What is the performance when all faulty nodes are Byzantine?

To answer the first question, we show that if our protocol only has the fast path (as analyzed in Sec. VI, only fast path will be triggered under rational nodes), the performance of our protocol is remarkable. Specifically, rBFT achieves 6-20x higher throughput than running PBFT among all nodes in the system. In fact, the fast path of rBFT is the same as conventional committee sampling based protocols as shown in Figure 2. Thus, this is the desirable feature of any committee sampling based approach.

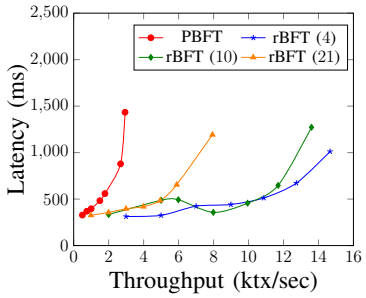
To answer the second question, we assess the performance of rBFT under failure scenarios. Specifically, we randomly select f =10%, 20%, and 30% of the nodes as Byzantine



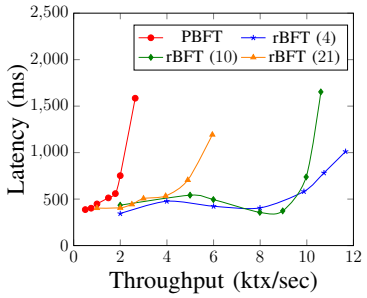
(a) Peak throughput of rBFT and PBFT [2] as n grows.



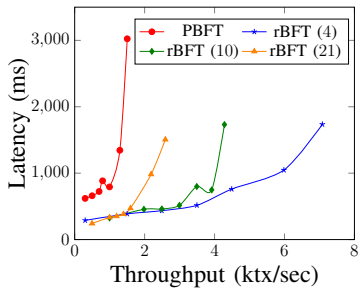
(b) Low-load latency of rBFT and PBFT [2] as n grows.



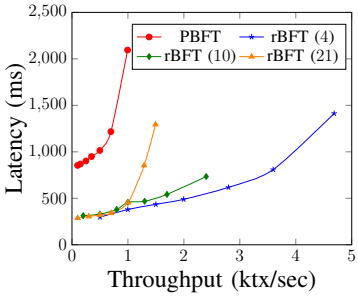
(c) Latency vs. throughput for $n = 50$.



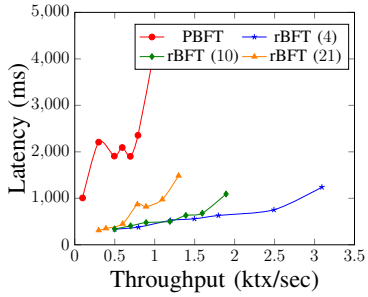
(d) Latency vs. throughput for $n = 100$.



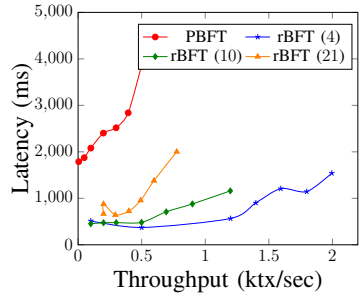
(e) Latency vs. throughput for $n = 200$.



(f) Latency vs. throughput for $n = 300$.



(g) Latency vs. throughput for $n = 400$.



(h) Latency vs. throughput for $n = 500$.

Fig. 5: Performance of the protocols.

in the entire system. When any of the Byzantine node is selected as the committee member, it will equivocate and try to make correct committee members *a-deliver* inconsistent blocks. We still assess the performance of rBFT and show the performance degradation. When approaching the capability limit, the throughput drops by up to 35%, as the transactions processed by the byzantine committee are invalid. However, under normal load, the degradation in throughput does not exceed 1%. This indicates that the overhead of MVBA instances is quite low and has little impact on system performance.

Q1(a): Performance with only fast path. We evaluate the performance of rBFT scales by varying n from 100 to 500. As shown in Figure 5a, the peak throughput of rBFT degrades as the network grows. This is expected, since all nodes in the system need to synchronize the output of $\Pi_{\lambda}^{\mathsf{com}}$, they will consume more network resources. The degradation fraction of throughput ranges from 20% to 80% as the number of nodes increases to 500. Although there is a significant degradation, the throughput of rBFT (4) still remaines at 4x that of PBFT protocol.

We also compare the latency of the protocols, as shown in Figure 5b. An interesting fact is that the latency of rBFT does not increase significantly as n grows. This is mainly because the synchronization of agreement result only involves one step of communication between the committee members and all nodes in the system. Even with 500 nodes, the latency of rBFT (21) is only 543 ms, showing that rBFT is highly practical. In contrast, the latency of PBFT grows significantly as n grows.

Q1(b): rBFT with different committee sizes vs. the baseline protocol. We assess latency vs. throughput for rBFT for $n = 50$ to 500 by varying the committee size as 4, 10 and 21. As shown in Figures 5c-5h, for the same n, the performance of rBFT is higher when the committee size is smaller. This is expected as having a small committee size will lower both communication and computational costs. For example, the peak throughput of rBFT (4) is 2x that of rBFT (10) and 3x

that of rBFT (21). In fact, as we show in our work, rational nodes will not try to break the security properties of $\Pi_{\lambda}^{\mathsf{com}}$. Thus, we can actually use a quite small committee size to achieve high performance. For all the cases, rBFT consistently outperforms PBFT. This is expected, as in the fast path, only committee members run the PBFT protocol.

	n	# Byzantine nodes					
		$f = 10\%n$		$f = 20\%n$		$f = 30\%n$	
		full	normal	full	normal	full	normal
rBFT (4)	200	5.08%	0.08%	17.31%	0.29%	34.51%	0.57%
	500	4.17%	0.18%	19.68%	0.84%	34.98%	1.49%
rBFT (10)	200	0.96%	0.01%	12.16%	0.18%	29.47%	0.43%
	500	1.70%	0.06%	11.36%	0.40%	33.89%	1.19%
rBFT (21)	200	0.00%	0.00%	3.59%	0.05%	28.88%	0.43%
	500	0.00%	0.00%	5.14%	0.15%	28.61%	0.84%

TABLE II: Throughput and degradation of rBFT for various n and f under Byzantine scenario. Full load refers to the maximum throughput capacity of the system. Normal load represents typical operating conditions where the transaction submission rate is set at 500 tx/s.

Q2(a): Throughput degradation under Byzantine failures. To evaluate the performance impact of Byzantine behavior, we run different scenarios by failing different fractions of nodes in the system. Specifically, we implement the following adversarial behavior when each Byzantine node is selected as any committee member: (1) when selected as the leader of the committee, it sends conflicting proposals to different correct nodes (equivocation); (2) it withholds messages or sends invalid signatures; and (3) it votes for conflicting proposals. We fail different fractions of faulty nodes and conduct experiments where each configuration runs for 10,000 blocks, and we measure the system's throughput and latency under these adversarial conditions. When the number of Byzantine nodes in the committee exceeds $1/3$, MVBA instances will be triggered at the end of the epoch, and all the blocks generated within that epoch will be rolled back.

As shown in Table II, the adversary ratio f is the most significant factor contributing to performance degradation. 30% adversary ratio can cause a throughput degradation of 28%-35% when the system is operating at full load. Here, *full load* refers to the setup when peak throughput can be achieved. Under full load, the system's resources are fully saturated, and any disruption (e.g., Byzantine behavior triggering MVBA) directly translates to performance degradation. The degradation is expected, as a faulty committee will make all nodes fail to reach an agreement in stage 2 of our protocol. In contrast, the *normal load* represents typical operating conditions where the transaction submission rate is set at 500 tx/s. Under normal load, our results show that the performance degradation is less than 1.5% for all configurations.

Q2(b): Latency breakdown of the slow path. We also have the MVBA triggered at stage 2 of each epoch under n=100 to 500 scenarios to assess the latency breakdown of rBFT according to three phases: stage 1, stage 2 (sync), stage 2 (MVBA). Namely, stage 1 and stage 2 (sync) together assess the latency of the fast path, and all three phases assess the latency of the slow path.

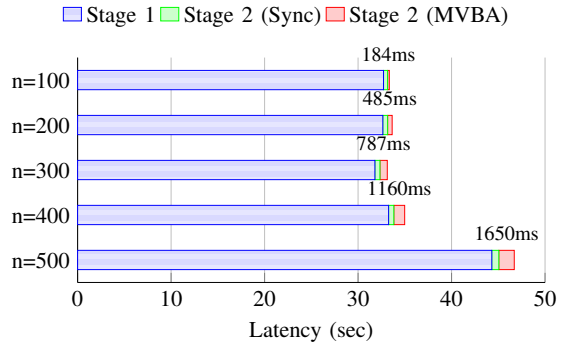


Fig. 6: Latency breakdown of rBFT for different n with 10 committee members. The slow path is forcibly triggered in every epoch and the result is the latency of 100 epochs.

As shown in Figure 6, the latency of stage 2 (MVBA) is quite low, which accounts for only 0.11%-3.36% of the total epoch time. So even if the adversary frequently acts maliciously, causing the MVBA to be triggered, all nodes can still quickly synchronize and enter the block generation process for the next epoch. To conclude, even in the slow path, our rBFT protocol is still highly practical and suffers from low performance degradation.

protocols	**rBFT(4)**		**Pando(0.2)**		**Algorand**	
n	100	500	100	500	100	500
Peak TPS	122,909	82,166	362,738	214,333	9,014	1,434
Latency (ms)	259	365	2,771	23,305	2,723	2,812

TABLE III: Performance of rBFT (4), Pando (0.2) [40] and Algorand [39] for $n = 100$ and $n = 500$.

Comparison with Pando and Algorand. Pando [40] represents a state-of-the-art BFT consensus protocol from committee sampling assuming an adaptive adversary and Algorand [39] is a committee sampling-based BFT protocol used in the leading blockchain Algorand with over one billion market capitalization. Under identical deployment configurations, we compare Pando (0.2) (20% of the nodes belong to the committee), Algorand and our rBFT (4). As shown in Table III, rBFT outperforms Algorand in both latency and peak throughput. In particular, our protocol has 13.6-57.2× higher peak throughput and 10.5× faster latency compared with Algorand. Additionally, rBFT demonstrates a lower throughput due to a smaller committee size while Pando has better peak throughput. Notably, rBFT achieves a latency of 259ms at $n = 100$, which is approximately 10× faster than Pando's 2,771ms and Algorand's 2,723ms.

VIII. Conclusion

We present rBFT, a new Byzantine fault-tolerant protocol from delegated committee via a revamped two-stage design. We show that the paradigm adopted by current industrial

systems has overly strong assumptions and the real instances such as takeover attack to Delegated Proof-of-Stake (DPoS) systems also support our claim. We provide a refined notion of BFT into a new *two-stage BFT* definition that preserves the safety and liveness properties of conventional BFT but additionally introduces new properties to model the two-stage design. Our construction, rBFT protocol, instantiates two-stage BFT via a fast path that is extremely fast and a slow path that achieves decent performance.

REFERENCES

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

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

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

[4] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

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

[6] "Tron," https://tron.network, 2025.

[7] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[8] BNB, "Bnb smart chain," https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md, 2025.

[9] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone *et al.*, "PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain," in *CEUR Workshop Proceedings*, vol. 2058. CEUR-WS, 2018.

[10] Clique, "Clique in Go-Ethereum," *https://github.com/ethereum/go-ethereum/blob/master/consensus/clique/clique.go*, 2021.

[11] C. Li, B. Palanisamy, R. Xu, L. Duan, J. Liu, and W. Wang, "How hard is takeover in dpos blockchains? understanding the security of coin-based voting governance," in *CCS*, 2023, pp. 150–164.

[12] 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.

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

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

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

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

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

[18] A. Bhangale, C.-D. Liu-Zhang, J. Loss, and K. Nayak, "Efficient adaptively-secure byzantine agreement for long messages," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 504–525.

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

[20] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of ACM*, vol. 32, no. 2, pp. 288–323, 1988.

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

[22] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "Bft protocol forensics," in *CCS*, 2021.

[23] S. Sridhar, D. Zindros, and D. Tse, "Better safe than sorry: Recovering after adversarial majority," *arXiv preprint arXiv:2310.06338*, 2023.

[24] D. Malkhi, K. Nayak, and L. Ren, "Flexible byzantine fault tolerance," in *CCS*, 2019, pp. 1041–1053.

[25] J. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Rational protocol design: Cryptography against incentive-driven adversaries," in *FOCS*. IEEE, 2013, pp. 648–657.

[26] C. Badertscher, J. Garay, U. Maurer, D. Tschudi, and V. Zikas, "But why does it work? a rational protocol design treatment of bitcoin," in *EUROCRYPT*. Springer, 2018, pp. 34–65.

[27] J. Neu, E. N. Tas, and D. Tse, "The availability-accountability dilemma and its resolution via accountability gadgets," in *FC*, 2022.

[28] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic, "As easy as abc: Optimal (a) ccountable (b) yzantine (c) onsensus is easy!" *Journal of Parallel and Distributed Computing*, 2023.

[29] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," *ACM SIGOPS operating systems review*, 2007.

[30] A. R. Yumerefendi and J. S. Chase, "Trust but verify: accountability for network services," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, 2004.

[31] ——, "The role of accountability in dependable distributed systems," in *Proceedings of HotDep*, 2005.

[32] P. Civit, S. Gilbert, and V. Gramoli, "Polygraph: Accountable byzantine agreement," in *ICDCS*, 2021.

[33] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *SP*, 2021.

[34] A. Stewart and E. Kokoris-Kogia, "Grandpa: a byzantine finality gadget," *arXiv preprint arXiv:2007.01560*, 2020.

[35] T. Gong, G. F. Camilo, K. Nayak, A. Lewis-Pye, and A. Kate, "Recover from excessive faults in partially-synchronous bft smr," in *USENIX Security*, 2025.

[36] J. D. Leshno, E. Shi, and R. Pass, "On the viability of open-source financial rails: Economic security of permissionless consensus," *arXiv preprint arXiv:2409.08951*, 2024.

[37] A. Lewis-Pye, J. Neu, T. Roughgarden, and L. Zanolini, "Accountable liveness," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 3431–3445.

[38] D. Collins, S. Duan, J. Loss, C. Papamanthou, G. Tsimos, and H. Wang, "Towards optimal parallel broadcast under a dishonest majority," in *FC*, 2025.

[39] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, p. 51–68.

[40] X. Wang, H. Wang, H. Zhang, and S. Duan, "Pando: Extremely scalable bft based on committee sampling," in *NDSS*, 2026.

[41] J. Halpern and V. Teague, "Rational secret sharing and multiparty computation," in *STOC*, 2004, pp. 623–632.

[42] S. D. Gordon and J. Katz, "Rational secret sharing, revisited," in *International conference on security and cryptography for networks*. Springer, 2006, pp. 229–241.

[43] I. Abraham, D. Dolev, R. Gonen, and J. Halpern, "Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation," in *PODC*, 2006, pp. 53–62.

[44] G. Kol and M. Naor, "Cryptography and game theory: Designing protocols for exchanging information," in *Theory of cryptography conference*. Springer, 2008, pp. 320–339.

[45] G. Fuchsbauer, J. Katz, and D. Naccache, "Efficient rational secret sharing in standard communication networks," in *TCC*. Springer, 2010, pp. 419–436.

[46] G. Asharov, R. Canetti, and C. Hazay, "Toward a game theoretic view of secure computation," *Journal of Cryptology*, vol. 29, no. 4, pp. 879–926, 2016.

[47] A. Groce and J. Katz, "Fair computation with rational players," in *Eurocrypt*. Springer, 2012, pp. 81–98.

[48] A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas, "Byzantine agreement with a rational adversary," in *ICALP*. Springer, 2012, pp. 561–572.

[49] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *OSDI*, 2006, pp. 191–204.

[50] A. Clement, H. Li, J. Napper, J.-P. Martin, L. Alvisi, and M. Dahlin, "Bar primer," in *DSN*. IEEE, 2008, pp. 287–296.

[51] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *SOSP*, 2005, pp. 45–58.

[52] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *Journal of the ACM*, 2015.

[53] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol with chains of variable difficulty," in *CRYPTO*, 2017.

[54] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.
[55] S. Duan, X. Wang, and H. Zhang, "Practical signature-free asynchronous common subset in constant time," in *CCS*, 2023.
[56] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited," in *PODC*, 2020.
[57] S. Das, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup, "Asynchronous consensus without trusted setup or public-key cryptography," in *CCS*, 2024.
[58] T. S. Ferguson, *A course in game theory*. World Scientific, 2020.
[59] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault tolerant services," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 253–267, 2003.
[60] X. Sui, S. Duan, and H. Zhang, "Marlin: Two-phase BFT with linearity," *DSN*, 2022.
[61] 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.
[62] 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.
[63] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Speeding dumbo: Pushing asynchronous bft closer to practice," *NDSS*, 2022.
[64] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, pp. 143–202, 2000.
[65] M. Zhang, R. Li, X. Lu, and S. Duan, "Available attestation: Towards a reorg-resilient solution for ethereum proof-of-stake," in *USENIX Security*, 2025.
[66] M. Zhang, R. Li, and S. Duan, "Max attestation matters: Making honest parties lose their incentives in ethereum pos," in *USENIX Security*, 2024.
[67] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," *arXiv preprint arXiv:2003.03052*, 2020.
[68] H. Sun, N. Ruan, and C. Su, "How to model the bribery attack: A practical quantification method in blockchain," in *Computer Security–ESORICS*, 2020.
[69] D. Karakostas, A. Kiayias, and T. Zacharias, "Blockchain bribing attacks and the efficacy of counterincentives," in *CCS*, 2024.
[70] H. Li, Y. Chen, X. Shi, X. Bai, N. Mo, W. Li, R. Guo, Z. Wang, and Y. Sun, "Fisco-bcos: An enterprise-grade permissioned blockchain system with high-performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23, 2023.
[71] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.
[72] "Erc-20," https://eips.ethereum.org/EIPS/eip-20, 2015.

Appendix A
An Attempt

Figure 7 achieves optimal resilience by making Π_n^{all} also a BFT protocol. Instead of agreeing on the order of blocks, Π_n^{all} agrees on the agreement results, e.g., $\mathsf{C}_{e,i}$. By running a fully-fledged BFT protocol and letting all nodes in the system wait for a short period (i.e., Δ), all nodes in the system agree on the output of $\Pi_\lambda^{\mathsf{com}}$. By running an MVBA protocol, nodes either agree on one chain C_e and extend their canonical chain, or enter the next epoch without by discarding the agreement results in epoch e. The predicate of MVBA of an input value v is defined as $Q(v) \equiv 1$ if and only if v is $\perp$ or a valid chain C in this epoch, which is signed by at least $\mathcal{RQ}(\mathcal{P}_e)$ committee nodes.

The protocol is both safe and live, even with $\epsilon = 0$. This is primarily because the consistency property of MVBA ensures that correct nodes always enter the new epoch with a consistent result of the previous epoch. After GST, there exists some epoch e' such that $\mathcal{P}_{e'}$ is correct with an overwhelming probability. This correct committee would output a correct chain containing all unconfirmed transactions.

An attempt

- **Initialization**: epoch $e \leftarrow 0$, parties $\mathcal{P}$, canonical chain $\mathsf{C} \leftarrow \perp$
- $\Pi_\lambda^{\mathsf{com}}$ is required to be verifiable
- The predicate of MVBA of an input value V is defined as $Q(V) \equiv 1$ if and only if V is $\perp$ or a valid chain C in this epoch, which is signed by enough committee nodes.

Epoch e

- Select a committee of λ nodes $\mathcal{P}_e$
- **Stage 1.** If $p_i \in \mathcal{P}_e$, execute $\Pi_\lambda^{\mathsf{com}}$ and output $\mathsf{C}_{e,i}$
- **Stage 2.** After epoch e times out, execute Π_n^{all}, which is instantiated as follows
  - If $p_i \in \mathcal{P}_e$, send $\mathsf{C}_{e,i}$ to all nodes in $\mathcal{P}$
  - For any p_i, starts a timer Δ
  - Upon receiving a valid $\mathsf{C}_{e,j}$ from p_j, $S \leftarrow S \cup \mathsf{C}_{e,j}$
  - After Δ times out
    - If $|S| = 0$, *mvba-propose* $\perp$ to MVBA_e
    - If $|S| \geq 1$, randomly select a $\mathsf{C}_{e,j}$ from S and *mvba-propose* $\mathsf{C}_{e,j}$ to MVBA_e
  - After MVBA_e outputs V
    - If $V = \perp$, do not update C
    - If $V = \mathsf{C}_{e,j}$, discard S and set $C \leftarrow C || \mathsf{C}_{e,j}$

Fig. 7: An attempt. The protocol achieves the security goals of BFT where $\epsilon = 0$. Code for p_i.

Analysis. The protocol in Figure 7 is a secure ABC protocol. In fact, it directly achieves optimal resilience, i.e., $f < n/3$. This is mainly because all nodes in the system agree on the output of $\Pi_\lambda^{\mathsf{com}}$. However, the protocol suffers from two limitations. First, transaction finality is delayed since each node only *a-delivers* a block after MVBA outputs some value and MVBA is triggered once every epoch. Second, MVBA involves all n nodes. Since it is triggered in every epoch, the communication overhead can be high.

Appendix B
Proof of Theorem 2

Proof. We describe the construction of the simulator $\mathcal{S}$ that emulates the real-world adversary's behavior in different cases. The simulator $\mathcal{S}$ works in an ideal-world where it operates with the idealized functionality $\langle \mathcal{F}_{ABC,e} \rangle$ and outputs the simulated execution $\mathrm{EXEC}_{\langle \mathcal{F}_{ABC,e} \rangle, \mathcal{S}, \mathcal{Z}}$ to the environment $\mathcal{Z}$. The adversary $\mathcal{A}$ is invoked by simulator $\mathcal{S}$ as a black box.

In the real-world, the adversary and parties interact with $\Pi_\lambda^{\mathsf{com}}$, the execution $\mathrm{EXEC}_{\Pi_\lambda^{\mathsf{com}}, \mathcal{A}, \mathcal{Z}}$ contains the input from $\mathcal{Z}$ and the output to all parties. If no party is corrupted, $\Pi_\lambda^{\mathsf{com}}$ will

be perfectly executed by all parties. Each party would receive the same output from $\Pi_\lambda^{\mathsf{com}}$, which is consistent with the input to $\Pi_\lambda^{\mathsf{com}}$. In that case, the simulator only needs to simulate the interaction process of the parties in strict accordance with $\Pi_\lambda^{\mathsf{com}}$, and finally input the sequence of blocks from $\mathcal{Z}$ to $\langle \mathcal{F}_{ABC,e} \rangle$. This simulated execution is indistinguishable from the real-world. Since E_s, E_ℓ and $E_\mathcal{I}$ do not happen, the utility of adversary is 0.

We next prove that, for any $0 < t < \mathcal{P}_e$, the simulator $\mathcal{S}$ with $\langle \mathcal{F}_{ABC,e} \rangle$ could simulate the adversary $\mathcal{A}$ with $\Pi_\lambda^{\mathsf{com}}$ and the utility is 0 for all adversary's attack strategy. There are two cases:

- **Case 1:** $0 < t < \frac{\lambda}{3}$**.** By the safety and liveness of $\Pi_\lambda^{\mathsf{com}}$, the output to each parties that is not corrupted is a identical sequence of blocks $\boldsymbol{b}'_e$, which is a permutation of input $\boldsymbol{b}_e$ from $\mathcal{Z}$. The simulator $\mathcal{S}$ first inputs the sequence of blocks from $\mathcal{Z}$ to $\mathcal{A}$ and learn about the corrupted parties from $\mathcal{A}$. Then it simulates the messages of other parties and sends them to $\mathcal{A}$, obtaining the execution of the corrupted parties. By simulating the interaction between parties that strictly implement protocol $\Pi_\lambda^{\mathsf{com}}$ and corrupted parties, $\mathcal{S}$ finally generates the output result $\boldsymbol{b}'_e$. To spread the result to all ideal-world parties, $\mathcal{S}$ first invokes $\langle \mathcal{F}_{ABC,e} \rangle$ with input $\boldsymbol{b}_e$. For $\mathcal{Z}$, the input and output from either real-world or ideal-world parties have the same distribution. Hence, $\mathcal{S}$ can simulate the adversary $\mathcal{A}$ without triggering event E_s or E_ℓ, but triggers $E_\mathcal{I}$, so the utility is $-t\gamma_\$ < 0$.
- **Case 2:** $\frac{\lambda}{3} \leq t < n$**.** In the real-world, each party p_i initiates with $\boldsymbol{b}_e$ and finally receives a sequence of blocks $\boldsymbol{b}^*_{e,i}$ from $\Pi_\lambda^{\mathsf{com}}$. For any $i \neq j \in [\mathcal{P}_e]$, there can be $\boldsymbol{b}^*_{e,i} \neq \boldsymbol{b}^*_{e,j} \neq \boldsymbol{b}_e$ since the adversary $\mathcal{A}$ can arbitrarily destroy the safety and liveness. For the ideal-world, the simulation of honest parties and corrupted parties is identical to case 1. In the last step, when $\langle \mathcal{F}_{ABC,e} \rangle$ is invoked, $\mathcal{S}$ judges each party p_i based on the result of simulated $\Pi_\lambda^{\mathsf{com}}$: (1) if there is at least one $\boldsymbol{b}^*_{e,i} \neq \boldsymbol{b}_e$, then $\mathcal{S}$ triggers event E_s and input $\boldsymbol{b}^*_e$; (2) if there is at least one $|\boldsymbol{b}^*_{e,i}| < |\boldsymbol{b}_e|$, then $\mathcal{S}$ triggers event E_ℓ and input $\boldsymbol{b}^*_e$; By triggering the two events, $\mathcal{S}$ could simulate any result of real-world attack and $\mathcal{Z}$ cannot distinguish them. The maximum utility of the adversary is $\gamma_s + \gamma_\ell - t\gamma_\$ < 0$.

Hence, in any case, if the adversary corrupts at least one node, its utility will be negative. The optimal strategy for the rational adversary is to corrupt no parties, i.e., be a "dummy" adversary, where the utility is 0. □

APPENDIX C
PROOF OF RBFT

Without loss of generality, our proofs assume $n = 3f + 1$. We also follow prior works [12], [13], [14], [15], [16], [17], [18] and consider λ as a security parameter with size $O(\kappa)$.

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

Proof. Correctness follows [40, Lemma 1]. □

Corollary 4. *Let α^* be the fraction of correct nodes in the system that hold some value v. If we sample a committee of $\frac{3\alpha}{\epsilon^2} \ln \frac{1}{\mu} = O(\kappa)$ size, $\alpha^*\kappa$ committee members hold value v with probability $1 - \mathsf{negl}(\kappa)$.*

Corollary 5. *If we sample a committee of $\frac{3\alpha}{\epsilon^2} \ln \frac{1}{\mu} = O(\kappa)$ size, and $\alpha^*\kappa$ committee members hold some value v, then with probability $1 - \mathsf{negl}(\kappa)$, $\alpha^* n$ correct nodes in the system that holds value v.*

Theorem 6 (Universal agreement)**.** *If a committee $\mathcal{P}_e$ is correct and a correct node a-delivers a block b before a-delivering b', then no correct node a-delivers a block b' without first a-delivering b.*

Proof. Correctness follows the safety of the underlying BFT protocol used in $\Pi_\lambda^{\mathsf{com}}$. □

Theorem 7 (Safe finalization)**.** *If a correct node a-finalizes a block b before a-finalizing b', then no correct node a-finalizes b' without first a-finalizing b.*

Proof. We assume that a correct node p_i *a-finalizes* block b in configuration c_1 where b is tagged with epoch e_1; p_i also *a-finalizes* block b' in configuration c_2 where b' is tagged with epoch e_2; $e_2 > e_1$; and $c_2 \geq c_1$. We also assume that a correct node p_j *a-finalizes* block b in configuration c_3 where b is tagged with epoch e_3; p_j also *a-finalizes* block b' in configuration c_4 where b' is tagged with epoch e_4; $e_3 > e_4$; and $c_3 \geq c_4$.

There are two cases: $c_1 = c_3$; $c_1 < c_3$ (the case where $c_1 > c_3$ is the same as $c_1 < c_3$).

Case 1: $c_1 = c_3$**.** Without loss of generality, we assume $e_1 < e_3$, i.e., p_j has not *a-finalize* block b where b is tagged with epoch e_1, but *a-finalize* b later. If p_i *a-finalizes* b, the chain led by b (denoted as C_{e_1}) has been extended to $\mathsf{C}_{e'_1}$ such that $e'_1 - e_1 = \kappa$. In our protocol, p_i has already entered epoch e'_1. Similarly, if p_j *a-finalizes* b in e_3, the chain led by b (denoted as C_{e_3}) has been extended to $\mathsf{C}_{e'_3}$ such that $e'_3 - e_3 = \kappa$. In our protocol, p_i has entered epoch e'_3. Additionally, since p_j *a-finalizes* b' before b, C_{e_1} and C_{e_3} are conflicting.

Since $c_1 = c_3$, p_i and p_j share common committees between epochs e_1 to e'_3, denoted as $\mathcal{P}_{e_1}, \cdots, \mathcal{P}_{e'_3}$. According to Lemma 3, with the probability of $1 - \mathsf{negl}(\kappa)$, at least one committee member in $\mathcal{P}_e$ is correct where $e_3 \leq e \leq e'_3$. We also consider another correct committee $\mathcal{P}_{e'}$ where $e_1 \leq e' \leq e'_1$. We assume $\mathcal{P}_e$ produces a chain that extends C_{e_3} and $\mathcal{P}_{e'}$ produces a chain that extends C_{e_1}. According to our protocol, each correct node votes for a block if the block is consistent with its canonical chain. Thus, $\mathcal{RQ}(\mathcal{P}_e)$ previously maintained a canonical chain that is consistent with C_{e_3} and $\mathcal{RQ}(\mathcal{P}_{e'})$ previously maintained a canonical chain that is consistent with C_{e_1}. Since both $\mathcal{P}_e$ and $\mathcal{P}_{e'}$ are correct, $\mathcal{WQ}(\mathcal{P}_e)$ correct nodes vote for C_{e_3} and $\mathcal{WQ}(\mathcal{P}_{e'})$ vote for C_{e_1}. According to

Corollary 5, with probability $1-\mathsf{negl}(\kappa)$, $f+1$ correct nodes in the system maintain a canonical chain consistent with C_{e_3} and $f+1$ correct nodes in the system maintain a canonical chain consistent with C_{e_1}. Since there are $2f+1$ correct nodes in total, at least one correct node maintains both C_{e_1} and C_{e_3} as its canonical chain, a contradiction.

Case 2: $c_1 < c_3$**.** If $c_1 < c_3$, nodes already enter new configurations after p_i *a-finalizes* b by assumption. According to the protocol, if p_i *a-finalizes* b, the chain led by b (denoted as C_{e_1}) has been extended to $\mathsf{C}_{e_1'}$ such that $e_1' - e_1 = \kappa$. According to Lemma 3, with probability of $1-\mathsf{negl}(\kappa)$, at least one of the committees $\mathcal{P}_e$ is correct where $e_1 \leq e \leq e_1'$. Following a similar argument as that in case 1, at least $\mathcal{WQ}(\mathcal{P}_e)$ correct nodes have set their canonical chain as C_e such that C_e is an extension of C_{e_1}. According to Corollary 5, with probability $1-\mathsf{negl}(\kappa)$, $f+1$ correct nodes in the system set their canonical chain as C_{e_1} or an extension of C_{e_1}. When nodes move to configuration c_1+1, these $f+1$ correct nodes send an $(\text{ABORT}, c_1, h, \delta)$ to all nodes, where h is the hash of $b = \mathsf{head}(\mathsf{C}_{e_1})$ or the hash of an extended block of b. As each node collects $n-f$ (ABORT) messages before starting MVBA_{c_1}, it is not too difficult to see that all correct nodes set their $lock$ as a pair (h', δ') such that h' is the hash of a block b'' where b'' is either b or an extension of b. (Note that the case where a correct node maintains (h'', δ'') such that the block corresponding to h'' is conflicting with the block corresponding to h' is impossible according to case 1). According to the external validity property of MVBA, the output of MVBA $(*, *, h, \delta)$ at any correct node satisfies that h denotes a block that is either b or an extension of b.

By induction on the configuration number c, it is not difficult to see that when correct nodes enter c_3, they all set their canonical chain as C_{e_1} or an extension of C_{e_1}. However, p_j *a-finalizes* b in configuration c_3, its canonical chain extends b', which is conflicting with C_{e_1}. Accordingly, either the consistency property of MVBA is violated or p_j is a faulty node. Correctness of the theorem thus follows.

□

Corollary 8. *In the same configuration, if a correct node p_i a-finalizes block b and another correct node p_j a-finalizes block b' in the same epoch, $b = b'$.*

Theorem 9 (Identifiable abort)**.** *If one correct node aborts in some configuration, all correct nodes eventually agree upon the identity of at least one Byzantine node.*

Proof. We first prove that if one correct node aborts in some configuration, all correct nodes will also abort in the configuration. If one correct node p_i aborts in configuration c, it holds an equivocation proof σ_ℓ. According to the protocol, p_i sends σ_ℓ to all nodes in the system. Eventually, all correct nodes receive σ_ℓ. If any correct node has not sent an (ABORT) message yet, it will send an (ABORT) message. Thus, all correct nodes eventually receive $n-f$ (ABORT) messages and then set their $abort$ status as $true$. If a correct node sends an $(\text{ABORT}, c, h, \delta)$ message, another correct node sends an $(\text{ABORT}, c, h', \delta')$, the block b corresponding to h' is either the block b' corresponding to h or an extension of b. This is because Corollary 8 will be violated otherwise. After receiving $n-f$ (ABORT) messages, all correct nodes will start MVBA_c. According to the termination property of MVBA, all correct nodes will terminate MVBA_c and then enter configuration $c+1$.

We now show that all correct nodes agree on the identity of at least one Byzantine node. As the predicate of MVBA requires an input tuple to be true if a valid equivocation proof is included, at least one Byzantine node (e.g. the leader of some epoch) will be identified. □

Corollary 10. *If a correct node aborts in configuration c, all correct nodes eventually enter configuration $c+1$.*

Theorem 11 (Liveness)**.** *If a correct node a-broadcasts a block b, then all correct nodes eventually a-finalize b.*

Proof. According to our protocol, all nodes act as learners for $\Pi_\lambda^{\mathsf{com}}$. If a valid block b is *a-broadcasted* in an epoch but is not *a-delivered*, all correct nodes will locally cache b and then *a-broadcast* b when they are selected as committee members. We now show that: 1) all correct nodes eventually share the same configuration number; 2) after GST and under a correct committee $\mathcal{P}_e$, any *a-broadcasted* block b will be *a-delivered* and *a-finalized*.

According to Corollary 10, if a correct node aborts in configuration c, all correct nodes eventually abort in c and enter $c+1$. After GST, it is not difficult to see that all correct nodes always share the same configuration number.

After GST and after all correct nodes share the same configuration number, in each epoch e, the same committee $\mathcal{P}_e$ is selected. With probability $1-\mathsf{negl}(\kappa)$, $\mathcal{P}_e$ is a correct committee and the leader in $\mathcal{P}_e$ is correct. If a correct node *a-broadcasts* a block b, the liveness property of any underlying protocol of $\Pi_\lambda^{\mathsf{com}}$ ensures that correct nodes will *a-deliver* b. After epoch e ends, all correct nodes in $\mathcal{P}_e$ send chain C_e to all nodes in the system where $b \in \mathsf{C}_e$. No node is able to trigger configuration changes. Thus, all correct nodes set their canonical chain as $\mathsf{C}||\mathsf{C}_e$. In any epochs greater than e, all correct nodes in the committee only vote for a proposed block if the block is consistent with its canonical chain. It is then not difficult to see that eventually after $O(\kappa)$ epochs, all correct nodes *a-finalize* b. □