Securing Consensus from Long-Range Attacks through Collaboration

Junchao Chen
Exploratory Systems Lab
University of California, Davis
jucchen@ucdavis.edu

Suyash Gupta
University of Oregon
suyash@uoregon.edu

Alberto Sonnino
Mysten Labs
University College London (UCL)
alberto@mystenlabs.com

Lefteris Kokoris-Kogias
Mysten Labs
lefteris@mystenlabs.com

Mohammad Sadoghi
Exploratory Systems Lab
University of California, Davis
msadoghi@ucdavis.edu

***Abstract*—Decentralized systems built around blockchain technology promise clients an immutable ledger. They add a transaction to the ledger after it undergoes consensus among the replicas that run a Proof-of-Stake (PoS) or Byzantine Fault-Tolerant (BFT) consensus protocol. Unfortunately, these protocols face a long-range attack where an adversary having access to the private keys of the replicas can rewrite the ledger. An existing solution to this problem forces each committed block from these protocols to undergo another consensus, Proof-of-Work (PoW) consensus; PoW protocol wastes computational resources as miners compete to solve complex puzzles. In this paper, we present the design of our Power-of-Collaboration (PoC) protocol, which guards existing PoS/BFT blockchains against long-range attacks and requires miners to collaborate rather than compete. PoC guarantees fairness and accountability and only marginally degrades the throughput of the underlying system.**

***Index Terms*—Fault Tolerance, Blockchain, Security, Long-range Attack, Consensus, Transactions, Reliability, Proof-of-Work, Proof-of-Stake, Collaborative Mining.**

I. Introduction

Decentralized systems built using blockchain technology promise their clients an immutable and verifiable ledger [1], [2], [3]. These systems receive client transactions and use state machine replication to add these transactions to the ledger. As these systems are often composed of untrusting nodes, some of which are malicious or Byzantine, establishing state machine replication requires these systems to run a *consensus* protocol that can handle malicious attacks [4], [5]. The two most widely adopted categories of these consensus protocols are: Proof-of-Stake (PoS) protocols [3], [6] and traditional Byzantine Fault-Tolerant (BFT) protocols [4], [5].

Stake-oriented consensus protocols, such as Proof-of-Stake (PoS) protocols, use a probabilistic distribution to decide which node gets to add a new block of client transactions to the ledger; often, the nodes with a higher stake (or wealth) have a higher probability of proposing a new block [6]. Communication-oriented protocols, such as traditional BFT protocols, give each node an equal opportunity (a vote) to add an entry to the ledger; agreement on the next block is reached through successive rounds of vote exchange [4], [7]. Some systems combine PoS and BFT to yield efficient consensus [3]. Despite these differences, these protocols follow the same design: each block added to the ledger includes the *digital signatures* of a quorum of participants to prove that a quorum agreed to update the ledger.

Unfortunately, any decentralized system that employs these protocols suffers from a well-known attack: a *long-range attack* where an adversary attempts to create an alternate ledger and targets clients (or new participants) that cannot distinguish between the original ledger and the adversarial ledger [8], [9], [10], [11]. An adversary can launch a long-range attack on systems running PoS/BFT consensus protocols due to the following reason: In PoS/BFT protocols, it is *computationally inexpensive* for nodes to add a new block to the ledger. An adversary with *access to the private keys* of the honest nodes can use these keys to create an alternate ledger; following are the two ways to access the private keys of others:

1) *Stealing.* An adversary can attempt to steal the keys of the nodes; stealing private keys is a widespread attack, and such attacks have resulted in losses of up to $200 million [12], [13], [14].
2) *Bribing.* An adversary can bribe honest nodes to sell their private keys, especially nodes that once participated in the system and no longer have any stake in it. This bribery attack is feasible because decentralized systems expect to run for years and cannot guarantee that the original set of participants will always run the system. Based on the Tragedy of the Commons [15], rational participants will opt to earn further incentives by selling their keys.

Once an adversary has access to these keys, it can use them to fork the original ledger at a specific block number and create an adversarial ledger with alternate blocks. If it has stolen over 50% of the private keys, the adversary can control the whole train. Alternatively, *the adversary waits for existing nodes to leave and new nodes to join the system*. Once it discovers a new node interested to join the system, the adversary presents its adversarial ledger as the authentic ledger to the new node,

which unfortunately *cannot distinguish between the two*. It is hard for new nodes to distinguish between the ledgers despite the existence of the honest nodes in the system who have access to the original ledger because the adversary has used the private keys of such honest nodes to forge their identities.

A naive solution to this problem is to ensure that honest nodes never leave the system and that no new node can join it. However, this solution is impossible to implement in a decentralized setting as nodes frequently leave/join the system [16], [17]. Other prior attempts include: (1) Using key-evolving cryptographic techniques and increasing the number of keys an adversary needs to compromise [8], [18], which typically delays the imminent long-range attack. (2) Creating state checkpoints and storing them at all the nodes, assuming that an adversary can only compromise the keys of at most one-third of nodes [19], [11] (3) Periodically appending the ledger state to the Bitcoin blockchain [20], [21]. Indeed, the third direction can guard existing decentralized systems against long-range attacks. For an adversary to present an adversarial chain to the new nodes, it also needs to rewrite the Bitcoin ledger, which is computationally infeasible. Bitcoin employs the PoW consensus protocol, which follows a *computation-oriented* model as it requires all the nodes to compete toward solving a complex puzzle. Whichever node solves the puzzle first adds a new block and receives a reward as compensation for its efforts. As PoW nodes constantly compete with each other, PoW-based solutions lead to the wastage of computational resources, as there is only one winner. [22].

The challenges existing solutions face while eliminating long-range attacks make us conclude that any solution for long-range attacks should: (1) not rely on the long-term safe-keeping of private keys, (2) reduce wastage of computational resources, and (3) be computationally expensive to rewrite the ledger.

In this paper, we introduce *Power-of-Collaboration* (PoC) protocol, which, when appended to decentralized systems running PoS/BFT consensus, helps to meet the aforementioned goals. PoC is noninvasive as it works on the output of underlying PoS/BFT consensus protocol and has minimal impact on the performance of existing decentralized systems. PoC advocates for *collaborative mining*, which, like PoW, requires miners to solve a compute-intensive puzzle, but all the miners are now working together (instead of competing) to solve the same puzzle.

The most closely related work, Bitcoin's *centralized mining pools*, also attempts to reduce the costs associated with mining [23], [24], [25]. As the name indicates, these mining pools are centralized and managed by an organization. The organization sets the rules for the mining pool, decides which node should receive a reward and how much reward, and controls which node can participate in the pool. Not only is the existence of centralized mining pools against the ethos of a decentralized system, but the managing organization charges fees for management without spending any computational resources. Further, attempts to create a decentralized mining pool have been unsuccessful due to nodes not doing designated tasks and lack of accountability: the last block added by any decentralized mining pool in Bitcoin was in 2019 [25], [23].

PoC, in essence, functions as a single decentralized mining pool where all the nodes collaborate to find a solution for the compute-intensive puzzle. Like PoW, nodes are still spending their computational resources to find the nonce, which makes it computationally expensive for the adversary to create an adversarial ledger. However, we need to ensure that, like centralized mining pools, we reduce the wastage of computational resources while also guaranteeing decentralization and *fairness*. We do so by splitting the compute-intensive puzzle into a set of unique sub-problems, and each node works on a unique subset of these sub-problems; the solution to the original compute-intensive problem is present in these subsets. We also need to provide *accountability* and deter malicious nodes from not doing work, as it can delay the discovery of the solution. PoC does so through our slice-shifting protocol, which identifies and penalizes a malicious miner and transfers its work to honest miners. To allow participants join and leave the mining group, PoC also provides a reconfiguration solution to reschedule the problem assignment.

To show that PoC is effective in practice, we append it to several decentralized systems. In our first set of experiments, we append PoC to Apache's ResilientDB (Incubating) [26] as it provides access to an open-source permissioned blockchain platform and an optimized implementation of Pbft, a BFT consensus protocol. ResilientDB's Pbft implementation adds approximately 1000 blocks per second on a system of 128 replicas, and our experiments illustrate that PoC can sustain this throughput on a system of 128 miners and requires $29\times$ less mining time than Bitcoin's PoW protocol. In our final set of experiments, we use the Diablo [27] benchmarking framework to append PoC to *four* popular blockchain systems, namely Diem [28], Algorand [3], Ethereum [2], and Quorum [29]. Our results illustrate that PoC has a minimal impact ($\approx 10\%$) on the throughput of these blockchains. Next, we list our contributions.

- We present the Power-of-Collaboration (PoC) protocol, which, when appended to existing decentralized systems running PoS and BFT protocols, makes it computationally-expensive for an adversary to launch a long-range attack.
- PoC introduces the notion of collaborative mining, which divides the mining task among all the miners.
- PoC advocates fairness and accountability: rewards are distributed among the miners in proportion to their share of work, and Byzantine behavior is quickly detected and penalized through the slice-shifting.

Outline. In §II, we discuss various types of consensus protocols, pooled mining, and long-range attacks. In §III and §IV, we present the design of our PoC protocol and discuss the impact of malicious attacks.

II. Background

We begin by presenting the necessary background.

A. PoS *and* BFT *Consensus*

PoS consensus protocols allow each node to add the next block to the blockchain in proportion to its invested stake [3], [30]. Often, the stake is equivalent to a monetary token or currency. Once a stakeholder proposes the next block, all the other nodes also sign this block, which acts like an agreement among the nodes. Similarly, BFT protocols [4], [5] designate in each round a replica as a leader, which proposes a block. Following this, all the replicas work through multiple rounds of message exchange to ensure that the proposed block has the support of a quorum of honest replicas.

B. *Long-range attack on* PoS *and* BFT

A known attack that affects both PoS [20], [21] and BFT [9], [8] protocols is the long-range attack, where an attacker attempts to create an alternate ledger. As described in §I, in PoS/BFT protocols, it is computationally inexpensive for nodes to add a new block to the ledger. Thus, an adversary needs access to the private keys of the honest nodes, which it can do either through stealing or bribing. Once an adversary has access to these keys, it can use them to fork the original ledger at a specific block number or height and create an adversarial ledger with alternate blocks (orthogonal, but in the past, blockchains have observed forks due to malicious attacks [31]). As nodes of decentralized systems frequently leave/join the system [16], [17], the adversary can use this opportunity to present its adversarial ledger as the authentic ledger to a new node (or client), which unfortunately cannot distinguish between the two. We illustrate this through the following example.

Example 1. *Assume that a decentralized system $\mathcal{S}$ has the following* PoS *blockchain ledger: $\mathfrak{B}_1, \mathfrak{B}_2, ...\mathfrak{B}_k, ...\mathfrak{B}_n$. Say malicious nodes get access to the private keys of all the honest nodes and decide to create an adversarial ledger, starting from the k-th block. Once it is the turn of malicious nodes to propose new blocks, they reveal the following adversarial ledger: $\mathfrak{B}_1, \mathfrak{B}_2, ...\mathfrak{B}'_k, ..., \mathfrak{B}'_n, \mathfrak{B}'_{n+1}$. Any new node joining $\mathcal{S}$ cannot distinguish between these two ledgers and will choose the longest chain. Similarly, some existing honest nodes, if bribed, may decide to forfeit their ledger and switch to the malicious ledger. Moreover, as time passes, with old nodes leaving the system and new nodes unable to distinguish, a hard-working adversary may be able to affirm the adversarial ledger as the original ledger.*

In practice, there are a lot of examples where an adversary has successfully stolen the private keys of honest parties [12], [13], [14]. We agree that stealing so many keys, primarily when the nodes are distributed is hard. Hence, a rational attack is where the adversary bribes the honest validators who no longer have a stake in the system. As these validators have nothing to lose, Tragedy of the Commons [15] suggests that these validators will sell their private keys in return for some incentive.

A naive solution to this problem is to fix the set of nodes. In that case, even if the adversary has access to the private keys of these nodes, it cannot convince the honest nodes to switch to the adversarial ledger as, locally, each of them has a copy of the ledger. Unfortunately, it is hard to prevent old nodes from leaving the system. New nodes will eventually fill those spots, and the adversary needs to target only these nodes.

The challenges make us conclude that any solution for long-range attacks should: (1) not rely on the long-term safe-keeping of private keys, and (2) be computationally expensive for an attacker to rewrite the ledger.

C. *Proof-of-Work Consensus*

We briefly look at the design of PoW consensus protocol, which can help in preventing long-range attacks.

In the PoW protocol, each miner $\text{M} \in \mathcal{M}$ selects some client transactions from the available pool of transactions and packs them in a block $\mathfrak{B}$. This block $\mathfrak{B}$ also includes a header, which contains: (i) hash of the previous block ($prev$), (ii) the digest of all transactions or *Merkle root* $M_{\mathfrak{B}}$, (iii) difficulty D, and (iv) the nonce η, among other fields [23], [32]. As each miner decides which transactions to include in its block, two miners may mine blocks with different transactions that *extend* the same previous block (with the hash $prev$). Computing $M_{\mathfrak{B}}$ of all transactions in the block requires a miner M to compute a pairwise hash from leaves to the root. The difficulty D, also termed as the difficulty of finding the nonce, informs the miner of the range of *desired hash* [33]. Specifically, each miner continuously selects a random nonce η till it satisfies the following equation:

$$\texttt{hash}(prev \mid\mid M_{\mathfrak{B}} \mid\mid \eta) \; < \; D \tag{1}$$

When M discovers a *valid* nonce, it adds it to its block and broadcasts this block to all the miners. When another miner M′ receives a block with a valid nonce that extends the last block added to the ledger (with hash $prev$), M′ adds the received block to its ledger and starts building/mining the next block that extends the received block. Note: once M′ has added a block to the ledger, if in the future, M′ receives any other block that includes $prev$, it ignores that block. Consequently, the miner who discovers the nonce earliest has the highest probability of adding a new block to the ledger as its block can reach a majority of miners the earliest. Clearly, PoW miners compete with each other in an attempt to find a valid nonce; PoW consensus faces the following two challenges (among many others): (1) All but one miner waste their computational resources and only the winner receives an incentive for finding the nonce. (2) More than one miner can find a valid nonce, which can temporarily fork the ledger. As there is no longer one ledger and instead multiple forks, PoW protocols define a mechanism to trim all but one fork, leading to further wastage of computational resources.

One solution to reduce the probability of forks is to increase the hardness/difficulty of finding the nonce; decentralized systems dynamically update the difficulty D to fix the rate miners add new blocks to the ledger. These systems want to ensure that miners spend at least a fixed amount of time searching for the valid nonce. The value of D is a system

parameter and D increases if miners are producing blocks at a faster rate than expected or the probability of forks is high and decreases vice versa.

III. Power-of-Collaboration

PoC aims to guard a decentralized system running PoS/BFT protocols from long-range attacks. It offers the following properties:

- G1. PoC makes it computationally expensive (solve complex puzzles) for an adversary to overwrite the original ledger,
- G2. PoC requires miners to collaborate and work on the same block; each miner has to work on a subset of the search space. Consequently, miners spend fewer resources than PoW.
- G3. PoC ensures fairness by distributing incentives among all the miners; even if there is no valid nonce in a miner's search space, it receives an incentive for its efforts.
- G4. PoC penalizes any miner that fails to find a valid nonce, if present, in its search space.

Before we describe the design of PoC, we discuss some of the possible solutions and their limitations.

Version 1. Say a decentralized system running a PoS/BFT consensus protocol employs a PoW consensus subsystem to guard itself against long-range attacks. Each batch of transactions that the consensus protocol commits is forwarded to the PoW subsystem to add to the ledger maintained by PoW miners. Each miner M follows the PoW protocol: creates a PoW block that includes one or more committed batches, a transaction that transfers incentive to its account, the hash of the previous block $prev$, and initiates the search for a valid nonce. When M finds the nonce, it broadcasts its block and the nonce to the other miners. If another miner M$'$ receives a block/nonce, it starts building/mining the next block and includes the hash of the received block/nonce as the previous block. This solution faces the following two limitations: (i) all but one miner wastes their computational resources (lack of fairness), and (ii) more than one miner can find a valid nonce, which can temporarily fork the ledger and lead to a subsequent increase in the hardness/difficulty of finding the nonce (§II-C).

Version 2. Next, we replace the PoW consensus subsystem with a centralized mining pool, where the pool operator receives the next committed block and creates sub-problems for the pool's miners to mine. This solution does not face any of the above limitations. However, this solution illustrates control by a single operator/organization, which receives fees for its services and decides the incentives/penalties for the pool's miners.

Version 3. Finally, we replace the centralized mining pool with a decentralized mining pool, where miners decide to coordinate with each other without any operator. The following are the challenges for any decentralized mining pool-based solution (i) which miner decides the content of the block, (ii) how to fairly distribute rewards among the miners, and (iii) how to detect and penalize a malicious miner that delays block mining by not performing designated tasks

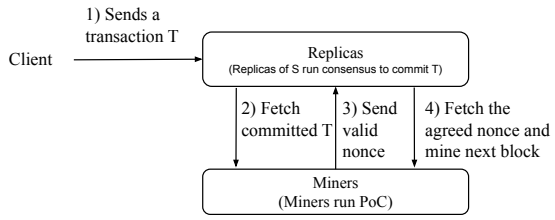

Fig. 1: Transactional flow in a system $\mathcal{S}$+PoC.

Overview. Our solution should offer the four appealing properties (G1 to G4). Consequently, we design PoC that requires no centralized organization and guards a PoS/BFT protocol from long-range attacks. In Figure 1, we illustrate the transactional flow.

(1) PoC expects that the underlying PoS/BFT protocol reaches consensus on client transactions among its replicas and forwards every committed batch of transactions to the PoC miners.

(2) Once miners are ready to mine, they select a set of ordered batches to form a block. To allow miners to collaborate and reduce computational resource wastage, PoC ensures that all the miners are mining identical blocks, and each miner searches for the valid nonce on a unique subset of the search space (property G2). It also shows that an adversary must provide more resources than half of the miners in order to create an alternative block as all the miners solve the nonce collaboratively. (property G1)

(3) Once a miner discovers a nonce, it broadcasts the nonce to everyone. To ensure that there are no forks of the ledger, PoC requires the underlying PoS/BFT consensus protocol to attest a nonce. Note: this recursive dependency helps to *quickly select* a valid nonce; the same task *can be done* by running a consensus on the nonce among the miners.

(4) Once a miner finds a valid nonce, each miner receives an incentive, which ensures fair reward distribution (property G3);

(5) Byzantine miners may not search for nonce in their subset of the search space. If the valid nonce is present in this subset, then no miner will ever discover it. PoC allows miners to independently discover such malicious attacks and switches honest miners to different subsets to facilitate the discovery of the nonce.

(6) PoC guarantees accountability by penalizing malicious miners that failed to find a valid nonce (property G4).

Next, we discuss our PoC assuming no attacks (*good case*). Later, we explain how we handle malicious attacks.

A. Client Transaction Ordering

Prior to running PoC, we expect the blockchain system $\mathcal{S}$ to reach consensus on client transactions. For PoC, the consensus run by $\mathcal{S}$ is a black box. The system $\mathcal{S}$ is free to run any consensus protocol of its choice. It needs to only provide PoC miners with *committed* blocks, a batch of transactions.

Our use of the term committed implies that each committed block has been accepted by a quorum of replicas of $\mathcal{S}$ and will persist across adversarial failures. For instance, in BFT protocols like PBFT [4] and HotStuff [5], blocks are committed when they have quorum certificates from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas.

In several other blockchain systems, a block is assumed committed if it is at a specific depth in the blockchain. A depth indicates the position of the block in the blockchain; the larger the depth of a block $\mathfrak{B}$, the greater the number of blocks that succeed $\mathfrak{B}$ and the harder it is for another fork to overtake this chain.

The assumption of POC miners working with only committed batches has two advantages: (1) It frees POC from having any knowledge on the consensus run by $\mathcal{S}$, and (2) POC miners will not waste their resources on batches that may not persist.

B. Chain Communication

Once the blockchain system $\mathcal{S}$ has committed a block, we require it to forward the committed block to POC miners to log it in the ledger. Like popular blockchain systems, Bitcoin and Ethereum, we employ the *gossip* protocol for broadcasting a block. Here, we are making a simplifying assumption that each node is connected to a sufficient number of honest nodes because, in gossip protocols, each node forwards the message to only its neighbors. Alternatively, the replicas of $\mathcal{S}$ can employ either the Information Dispersal Algorithm [34] or Byzantine Reliable Broadcast [35] if they cannot assume a uniform distribution of honest nodes. Each miner accepts a committed block once it receives it from $\mathbf{f}_{\mathcal{R}} + 1$ replicas in $\mathcal{S}$, which assures this miner that it did receive a committed block.

C. Collaborative Mining

POC introduces the notion of collaborative mining to log each committed block in the ledger.

Collaborative mining, like centralized pooled mining, should ensure that each miner works on a unique sub-problem so that miners do not waste their computational resources. Thus, we divide the POW hash computation into $\mathbf{n}_{\mathcal{M}}$ disjoint sub-problems and require each miner to work on a *distinct predetermined sub-problem*. Like existing POW systems [1], POC miners have to compute a SHA-256 hash, which is represented as a 32-byte hexadecimal value. Thus, the solution space $\mathbb{S}$ comprises of 2^{256} possible values. We divide $\mathbb{S}$ into $\mathbf{u}$ *slices*; the size of each slice is a system parameter. Given $\mathbb{S}_1, \mathbb{S}_2, ..., \mathbb{S}_{\mathbf{u}}$ slices, the following holds:

$$\mathbb{S}_1 \cap \mathbb{S}_2 \cap \cdots \cap \mathbb{S}_{\mathbf{u}} = \varnothing \quad \text{and} \quad \mathbb{S}_1 \cup \mathbb{S}_2 \cup \cdots \cup \mathbb{S}_{\mathbf{u}} = \mathbb{S}$$

POC assigns each miner one or more consecutive slices based on its stakes. We assume that each slice is assigned to a miner in $\mathcal{M}$. We discuss the slice assignment scheme in more detail in § III-E. Given the difficulty D, each miner computes a hash till it reaches the target 32-bit SHA-256 hash (refer to Equation 1). This requires each miner to find a nonce η in its set of slices.

Next, we describe the POC consensus.

D. POC *Protocol Steps*

From an outside view, our POC protocol works in rounds, and within each round, each miner attempts to find a valid nonce in its pre-determined slices. Next, we explain the POC protocol under the assumption that each miner knows the next block to mine (§III-C) and has received this block through chain communication (§ III-B). In the case that a miner does not have access to the next block to mine, it can ask the other miners about the missing blocks.

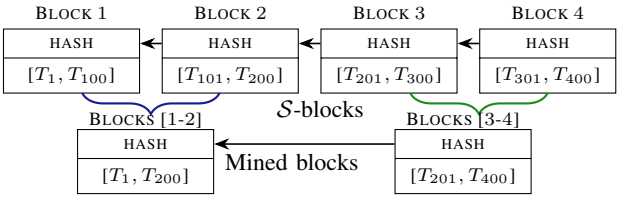

Fig. 2: Illustrating how $\sigma = 2$ contiguous $\mathcal{S}$-blocks produced by replicas of $\mathcal{S}$ are aggregated into one mined block of POC. Here, each $\mathcal{S}$-block includes 100 transactions.

Block Creation. When a POC miner receives a block from $\mathbf{f}_{\mathcal{R}}+1$ replicas, it adds that block to the *list of pending blocks*. The list of pending blocks is an ordered list of committed blocks that a miner is yet to add to the ledger; these blocks are ordered by the sequence number assigned by the POS/BFT consensus protocol running at $\mathcal{S}$. As the difficulty D of mining a block sets the rate at which blocks are added to the POC ledger, which in turn impacts the system throughput and latency, at higher difficulties, POC has a lower throughput than the POS/BFT consensus protocol. Consequently, POC miners have an ever-growing list of pending blocks, as the rate at which they add these blocks is slower than the rate at which they receive them from the POS/BFT protocol.

To reduce this gap between total blocks received and blocks mined, POC allows miners to batch a set of committed $\mathcal{S}$-blocks, thereby, each *mined block* includes $\sigma > 0$ committed $\mathcal{S}$-blocks.[1] The value of σ is a system parameter. POC requires miners to only aggregate σ consecutive $\mathcal{S}$-blocks from the pending list, which is essential for maintaining the block ordering by $\mathcal{S}$. For the sake of discussion, let us assume that each $\mathcal{S}$-block is assigned a monotonically increasing sequence number k and each mined block is assigned a sequence number b. If the last block mined by miners had sequence number $b-1$ and the sequence number of last $\mathcal{S}$-block added to the $(b-1)$-th block is $k-1$, then in the b-th block, each miner will aggregate the following blocks: $k, k+1, ..., k+\sigma$. Aggregating $\mathcal{S}$-blocks in this way is safe as these blocks are already committed by replicas of $\mathcal{S}$. We illustrate this next.

Example 2. *In Figure 2, the replicas of $\mathcal{S}$ committed four $\mathcal{S}$-blocks starting with sequence number 1. Assume $\sigma = 2$, then each* POC *miner aggregates 2 consecutive $\mathcal{S}$-blocks into a mined block.*

Each mined block includes a Merkle root of all the transactions; as each $\mathcal{S}$-block contains a Merkle root of all the transactions, a miner M generates the Merkle root for the mined block by hashing the Merkle roots of all the aggregated blocks.

Nonce Discovery. Once a miner knows the valid nonce for $(b-1)$-th block, it initiates the search for the nonce for b-th block. Assuming the miner M_i knows the set of slices it needs to mine (§III-E), which we denote as $\mathbb{S}_i$, M_i initiates nonce

[1]For disambiguation, we use $\mathcal{S}$-blocks to denote the blocks produced by replicas of $\mathcal{S}$ and mined block to denote the block produced by miners.

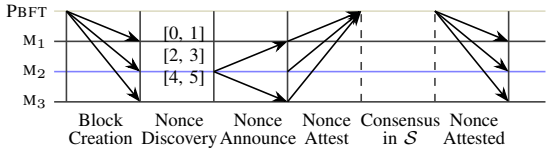

Fig. 3: PoC protocol with miners $\mathcal{M} = \{\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3\}$. The solution space $\mathbb{S} = [0, 5]$ is divided into three slices $([0, 1], [2, 3], [4, 5])$. Assume the valid nonce 2 and miner M_2 discovers it in its slice.

discovery. Specifically, M_i iterates over all the values in its slices $\mathbb{S}_i$.

Nonce Announcement. When a miner M_i finds a valid nonce η, it creates a message NONCEFIND that includes η and broadcasts this message to all the miners. When a miner M_j receives a NONCEFIND message, it terminates the process of nonce discovery if the received nonce is valid. Next, each miner that has access to a valid nonce gossips this nonce to the replicas of $\mathcal{S}$.

Nonce Attestation. Next, PoC leverages the underlying PoS/BFT consensus protocol to attest the discovered nonce. Specifically, when a replica R of $\mathcal{S}$ receives matching NONCEFIND messages from $\mathbf{f}_{\mathcal{M}} + 1$ miners, it creates a transaction that includes the received NONCEFIND message as its data. Whenever it is the turn of R to propose a new block, and if an $\mathcal{S}$-block containing the NONCEFIND message for the b-th mined block is yet to be proposed, R proposes a new block that includes this message. We expect honest replicas to prioritize nonce transactions over others to prevent delays in adding newly mined blocks to the ledger.

Chain Append. When a miner M_i receives the valid nonce for the b-th block from replicas of $\mathcal{S}$, M_i appends this block to its local ledger and marks the nonce discovery process for the b-th block as complete. Post this, M_i executes the *reward* transactions in the block to distribute the reward (§III-E). Finally, M_i begins mining the next block. We use the following example to illustrate PoC mining.

Example 3. *In Figure 3, the solution space* $\mathbb{S} = [0, 5]$ *is divided among three miners. The three slices are:* $\mathbb{S}_1 = [0, 1]$, $\mathbb{S}_2 = [2, 3]$, *and* $\mathbb{S}_3 = [4, 5]$. *Assume the valid nonce is* 2 *and it lies in the slice of miner* M_2. *Once* M_2 *discovers the nonce in its slice, it broadcasts the nonce to all the miners, following which each miner requests the replicas of* $\mathcal{S}$ *to attest this nonce.*

Multiple Nonces. Given that there may be multiple nonces/solutions for each puzzle and that miners work on these puzzles independently, the nonce attestation phase is essential for finalizing the nonce; notably, choosing any nonce would be valid. We use PoS/BFT (the consensus layer) as a tie-breaker to choose any of these valid nounces. This is the key insight that simplifies POW computation (mining layer), allowing us to solve POW collaboratively because we essentially removed reaching common grounds (i.e., consensus) from the mining process and utilize the mining proof only as a notary process to prevent long-range attacks.

Thus, we need to guarantee that all the miners select the same nonce, which is trivial for PoC as each nonce is attested by replicas of $\mathcal{S}$. If the next proposer for an $\mathcal{S}$-block receives two or more NONCEFIND messages with distinct valid nonces, it selects one of them as the solution. When, eventually, this block becomes committed, all the PoC miners will have access to the same nonce for block b. The termination process that occurs upon receiving the NONCEFIND message is designed to optimize resource utilization, as only one solution can be accepted for each block. Most importantly, in PoC , miners collaborate toward a common goal (finding the solution), whereas in PoW, each miner operates independently on the same task.

E. Staking and Rewards

PoC, like PoW, rewards its miners with incentives for their participation in the mining process; miners expend their computational resources and would only do so if they make some profit. However, unlike PoW, PoC wants to ensure fairness by rewarding each miner for collaboration even though the valid nonce was not in its slice. In PoC, we reward each miner in proportion to the number of slices in its slice set. As stated in Section III-C, in PoC, there are a total of $\mathbf{u}$ slices. We assume the following: (1) Each of these $\mathbf{u}$ slices is assigned to a unique miner. (2) Each miner is assigned a set of consecutive slices. We require a miner to invest its monetary resources in exchange for each slice it holds. Like existing PoS systems, we term this investment by a miner as *staking* as the miner no longer has access to its invested monetary resources [2], [3], [30]. If a miner could not finish the mining due to the excessive slice space, it will be regarded as an adversary. We will also discuss the miner reconfiguration in [36] to reassign the slices whenever a new miner joins or an old miner leaves the system.

The economics of converting actual currency into an online tradable commodity is a problem faced by every decentralized system, for which current literature includes several ad hoc solutions. Thus, its design is beyond the scope of this paper. For simplicity, we assume that PoC builds on top of some token Ψ, where Ψ is the cost of purchasing a slice. Each miner exchanges its currency for a set of Ψ. If a miner M_i wants e slices in its set, M_i stakes $e \times \Psi$ tokens. This information is added to the first block of the PoC ledger, which is often referred to as the genesis block. Specifically, PoC-genesis blocks stores the following information: (1) total number of miners ($\mathbf{n}_{\mathcal{M}}$), (2) number of tokens staked by each miner, (3) a public key for each miner, and (4) slice to miner mapping.

During PoC's collaborative mining, each miner refers to this genesis block to identify the slices it is responsible for mining. Once a miner receives a valid nonce from the replicas of $\mathcal{S}$, it adds a reward to the account of each miner. Like existing systems [2], [1], we assume that the reward is proportional to the fees paid by the clients; each client pays a fee to add its transaction to the ledger and this fee is divided among the miners in proportion to their number of slices. For example, if the client for transaction T pays $\Diamond$ tokens and a miner M_i has e slices in its set, M_i receives $\frac{e \times \Diamond \times \Psi}{\mathbf{u}}$ tokens

as a reward. We maintain each miner's account as a key-value pair in a NoSQL database replicated across all miners/replicas; the key field represents the public key (logged in the genesis block) of each miner, while the value field represents the token balance.

IV. Malicious Attacks

Unlike a centralized mining pool, where the pool operator oversees the activity of all the miners and rewards/penalizes them for their actions, PoC assumes a decentralized setup. Thus, PoC needs to provide protection from malicious attacks and guarantee accountability while ensuring that it makes it computationally expensive for the adversary to overwrite the ledger using long-range attacks.

First, we expect that the underlying PoS/BFT protocol guarantees safety and liveness properties [36]. Thus, it should be capable of handling attacks by a standard adversary.

Second, a standard adversary can only attack the PoC in the following ways:

A1. As multiple nonces can satisfy Equation 1 and if Byzantine miners are fortunate enough to find two such nonces for a mined block, they can equivocate by sending each nonce to only a subset of honest miners and replicas.

A2. A miner decides to not participate in the PoC's collaborative mining or if a miner discovers a nonce in its slice, it decides to not send it to other miners.

Third, an advanced adversary that has access to the keys of any replica/miner can use these keys to forge any message.

Final, each distributed system needs to also guard against denial-of-service attacks. To prevent these attacks, we follow the best practices suggested by prior works [37] and assume that the replicas/miners use one-to-one virtual communication channels, which can be disconnected if needed.

Next, we discuss how we handle attacks by a standard adversary on PoC and attacks by advanced adversaries on the entire system.

A. Standard Adversary

As replicas of $\mathcal{S}$ help in selecting a nonce, Attack A1 is trivially resolved. Each replica R selects a nonce for the mined block at height b only after it receives $\mathbf{f}_{\mathcal{M}}+1$ matching NonceFind messages. Whenever it is the turn of R to propose a new block, and if an $\mathcal{S}$-block containing the nonce for the b-th mined block is yet to be proposed, R proposes a new block that includes this nonce. Thus, only one of the two nonces will commit and honest miners will receive only one nonce. If there aren't sufficient matching messages, then no nonce will be selected and this will lead to the eventual detection of some Byzantine miners.

To resolve Attack A2, next, we present our *slice shifting* protocol, which penalizes miners for malicious behavior.

B. Slice Shifting

Each honest PoC miner should search for a nonce in its set of slices until it receives a valid nonce from another miner or it has exhausted its slices. A malicious miner may not follow this behavior; it may want to disrupt the process of nonce finding. If the malicious miners are fortunate and the nonce is present in their slices, then honest miners may never receive the nonce, and PoC will come to a halt. We aim to quickly resolve such a situation. We do so by running our *slice shifting* algorithm that switches miner slices under failures. This is done with the aim that when an honest miner has access to the slice held by a malicious miner, it can discover the nonce in that slice and broadcast it to other miners.

Next, we illustrate slice shifting protocol through an example. Post this, we will explain the protocol in detail.

Example 4. *Figure 4 illustrates the slice shifting protocol. Assume the nonce lies in slice* $[2, 3]$ *and miner* M_2 *fails to announce the nonce. Eventually, honest miners timeout while waiting to receive a valid nonce and trigger slice shifting protocol. These miners must create a certificate to prove that a majority wants to do slice shifting. They send this certificate to the replicas of* $\mathcal{S}$ *for attestation. Post this, each miner works on a new slice. Once, they discover the nonce, they initiate the process of penalizing* M_2.

Timer Initialization. PoC miners, who have exhausted their slice search space and do not have a valid nonce, need a mechanism to make progress. We follow existing BFT works and require each miner to set a *timer* before it starts mining a slice. Specifically, each miner sets a timer δ for the b-th block and stops δ when it has a valid nonce for the b-th block.

Malicious Miner. If M's timer δ expires and it does not have access to a valid nonce, M announces to all the other miners that it wishes to initiate the *slice shifting* protocol. The slice shifting protocol runs for at most $\mathbf{f}_{\mathcal{M}}$ rounds and deterministically switches slices assigned to each miner. A round of slice shifting only takes place when at least $\mathbf{f}_{\mathcal{M}}+1$ miners request to do so. Specifically, when a miner M_i timeouts, it creates a message $\textsc{Shift}(b, \mathbf{r})$ and broadcasts this message to all the other miners. Here, $\mathbf{r}$ represents the slice shifting round, which is initially set to *zero*. When M receives Shift messages from $\mathbf{f}_{\mathcal{M}}+1$ distinct miners, it requests the replicas of system $\mathcal{S}$ to help reach a consensus on slice shifting. To make this request, each miner must broadcast a certificate $\mathfrak{C}$ that includes $\mathbf{f}_{\mathcal{M}}+1$ Shift messages.

Shift Attestation. When replicas of $\mathcal{S}$ receive a signed $\mathfrak{C}$ from $\mathbf{f}_{\mathcal{M}}+1$ PoC miners, they agree to attest this slice shift. This attestation requires the replicas to run consensus on this certificate $\mathfrak{C}$; the next proposer for a $\mathcal{S}$-block includes $\mathfrak{C}$ as a transaction in its block. *Note:* consensus on $\mathfrak{C}$ is like consensus on any transaction where $\mathfrak{C}$ acts as the transactional data. Post consensus, all the replicas gossip this block to the miners. Once a miner M receives a $\mathcal{S}$-block from $\mathbf{f}_{\mathcal{R}}+1$ replicas that include a committed certificate $\mathfrak{C}$, it assumes it is time to shift its slices. Following this, it increments the shift round $\mathbf{r}$ by one and mines the next slice. If in shift round $\mathbf{r}$, M_i was responsible for mining slices $\{\mathbb{S}_i, \mathbb{S}_{i+1}, ..., \mathbb{S}_j\}$, in round $\mathbf{r}+\mathbf{1}$, M_i will mine slices $\{\mathbb{S}_{i+1}, ..., \mathbb{S}_j, \mathbb{S}_o\}$, where $o = (j+1) \mod \mathbf{u}$, and $\mathbf{u}$ is the total number of slices. Again, before mining for round $\mathbf{r}+\mathbf{1}$, M_i restarts the timer δ for the k-th block. It is possible that the timer δ again timeouts, due to more failures. In such a

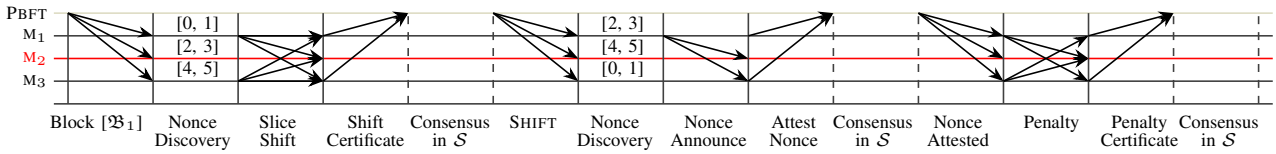

Fig. 4: Slice shifting procedure: assume 2 is the valid nonce and is present in the slice of malicious miner M_2. M_2 fails to broadcast 2, which triggers slice shifting; and with help of replicas of $\mathcal{S}$, once 2 is found, it is penalized.

case, M_i would need to initiate another round of slice shifting. However, under a standard adversary and reliable network, for each mined block, we need to run the slice shifting protocol only $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$ times.

No nonce – Merge. Although infrequent, PoC miners may encounter cases where no nonce satisfies Equation 1. This no nonce situation is scarce in PoW because miners work on different blocks, but it is possible in PoC because all the miners collaborate on the same block. We resolve this situation as follows.

If even after $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$ rounds honest miners do not have access to a valid nonce, we require the miners to terminate their search for the nonce and initiate the *merge* process, which requires miners to mine two or more consecutive mined blocks together, with the hope that mining multiple blocks together increases the probability of finding a nonce. For instance, for the b-th block, if a miner M receives a certificate $\mathfrak{C}$ from $\mathcal{S}$ replicas, which has shift round $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$, M concludes that no valid nonce exists for the b-th block. Following this, M creates a new block that merges contents of the b-th and $(b+1)$-th blocks. This merged block now serves as the b-th block and miners attempt to find the nonce for this block. The merged block includes a Merkle root, which is the hash of all the transactions in b and $(b+1)$-th blocks.

Penalty for Slice Shifting. Frequent slice shifting due to malicious miners is detrimental to the performance of PoC; it forces honest miners to do more work and wastes their resources. To discourage these attacks, we *penalize* malicious miners. We require each miner to track the number of shifts ($\mathbf{r}$) it took to find a valid nonce and to identify the miners that failed to perform this task.

In PoC, identifying malicious miners responsible for $\mathbf{r}$ shifts is a trivial task for honest miners. When a miner M_i receives a valid nonce for the b-th block in shift round $\mathbf{r}$ ($1 \leq \mathbf{r} \leq \mathbf{f}_{\mathcal{M}}$), it identifies malicious miners based on the slice containing the valid nonce. Let $\mathbb{S}_j$ be the slice, then M_i looks into the genesis block to find the initial assignment of the slice. Using this information, M_i determines the miners who held this slice in subsequent $\mathbf{r} - \mathbf{1}$ rounds and adds these miners to the set of malicious miners $\mathcal{M}_{mal}$.

To penalize these malicious miners, we again invoke the replicas of $\mathcal{S}$. Once a miner M has the knowledge of set $\mathcal{M}_{mal}$, it sends a message $\text{PENALTY}(\mathcal{M}_{mal}, b, \mathbf{r})$ to all the miners. Once M receives PENALTY messages from $\mathbf{f}_{\mathcal{M}} + 1$ miners, it creates a certificate (like it created for slice shifting) that includes these PENALTY messages and broadcasts this certificate to each replica in $\mathcal{S}$.

When the replicas of $\mathcal{S}$ receive a PENALTY certificate signed by $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners, they add it to a new block for consensus. Post consensus, honest miners penalize malicious miners by deducting their account balances.

C. Advanced Adversary: Long-Range Attacks

An advanced adversary, unlike a standard adversary, has access to the private keys of honest replicas/miners. It can use these keys to forge any message that has a digital signature. In the context of our system, the following attacks are possible: (1) Multiple committed $\mathcal{S}$-blocks with the same sequence number, (2) Forgery of nonce, shift, or penalty messages. (3) Forgery of $\mathcal{S}$-blocks, containing nonce, shift, or penalty transactions. If an adversary can compromise the keys of honest replicas/miners, PoC *can not guarantee* fairness and accountability; honest miners may get penalized. However, PoC guarantees that it is computationally expensive for the adversary to rewrite the existing ledger.

To illustrate that PoC guards against long-range attacks, we theoretically analyze its hardness. We follow Example 1 where starting from the k-th block, malicious parties want to rewrite the ledger. To do so, malicious replicas would first create an alternate set of $\mathcal{S}$-blocks using the compromised private keys of honest parties. Next, these malicious replicas would forward these $\mathcal{S}$-blocks to the malicious miners, who will attempt to forge the PoC ledger by creating new mined blocks. These malicious miners will not receive any help from honest miners as they have a local copy of the ledger.

As the combined computational power of honest miners is more than malicious miners, it is safe to assume that in any round of PoC, malicious miners control at most 50% slices and have a 50% chance of finding a valid nonce.[2] So, $\frac{1}{2}$ is the probability that malicious miners find the solution for an "alternate block" in their set of slices. The probability that malicious miners find the solution for the b consecutive alternate blocks in their set of slices is $(\frac{1}{2})^b$. If $b = 7$, the probability is 0.7%. Clearly, if Byzantine miners have to create a large number of alternate blocks, it is impossible for them to find the nonces by just mining their own set of slices. Alternatively, they can search for the nonce in all the slices (entire solution space) but this at least doubles their work.

Next, we measure the actual time required to create an alternate chain by taking into account two parameters: (1) the age of the chain α (in months) and (2) the hashing power ratio of the malicious miners (m) over the honest miners (h), $\frac{m}{h}$. Essentially, it takes $\alpha \times \frac{h}{m}$ months for the malicious miners to reconstruct an α-month-old chain given their overall hashing power ratio of $\frac{m}{h}$. For example, if the malicious miners hold

[2] Assuming that an adversary has access to the private keys of honest nodes is orthogonal to the assumption that an adversary also has more computational power than honest nodes. If an adversary controls more computational power, then it can rewrite even the PoW ledger.

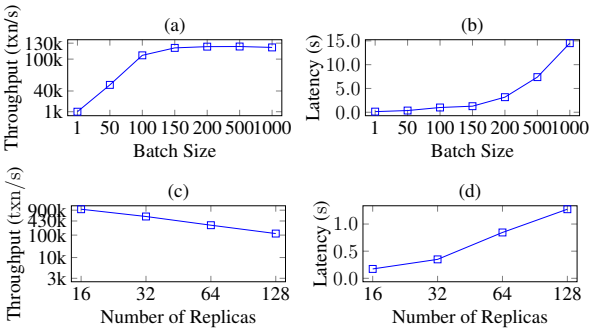

Fig. 5: Evaluation of ResilientDB architecture.

$\frac{1}{2}$ of the mining power and want to reconstruct a 1-year old chain, it would require two years' worth of computation to create an alternative chain of equal length.

During these two years, the following two things can happen: (1) The original chain will continue growing, which further increases the task of malicious miners. (2) The system is at a stall and honest miners/replicas detect that nothing is getting appended to the ledger and will leave the system. These arguments help us to demonstrate that simply compromising the private keys of honest miners/replicas is insufficient to launch a long-range attack on PoC.

Proofs. For detailed analysis and proof of correctness, please refer to our extended report [36].

V. Evaluation

Our evaluation aims to study the scalability and failure handling capability of PoC.

Implementation. We implement PoC on top of the Apache ResilientDB, written in C++ [26]; ResilientDB provides access to a scalable blockchain framework with APIs to implement and test new protocols. It provides access to an optimized implementation of the Pbft consensus protocol. Our PoC implementation has an LOC count of 1,300, while ResilientDB has 28,000 LOC.

Setup. We run experiments on AWS c5.9xlarge (36 vCPUs and 72 GiB memory), up to 128 miners and clients. *Unless explicitly stated*, we use the following setup: deploy 128 replicas in ResilientDB to run Pbft consensus on $\mathcal{S}$-blocks of size 100. In each experiment, first 20% of the time we set as warmup and results are collected over the remaining time period. We average results over *five* runs. We use *ED25519*-based DS for signing messages; Pbft makes use of *CMAC* for replica-to-replica communication. Clients issue requests in a closed-loop; a client sends its next transaction only after it receives response for its previous transaction.

Benchmark. We run two types of experiments: (1) impact of appending PoC to Pbft. (2) impact of appending PoC to real-world blockchains. For the first experiment, clients create *YCSB* [38] transactions from the Blockbench [38] framework. These single-operation transactions are key-value store operations that access a database of 600 k records. For the second experiment, we extend the Diablo benchmarking framework [27], which issues client transactions to four state-of-the-art blockchain platforms.

Protocol	Mining Time (s)			Latency (s)		
	$D=8$	$D=9$	$D=10$	$D=8$	$D=9$	$D=10$
PoW (Sequential)	696	$>2h$	$>5h$	1233	$>3h$	$>10h$
PoW (Random)	328	4108	$>5h$	652	8200	$>10h$
PoW + 30% PoC	7	165	3103	17	303	6950
PoC	3	48	738	6	84	1260

Fig. 6: Time to find a nonce by a system of 128 miners while ensuring that these mining protocols meet ResilientDB Pbft's throughput of 1k blocks per second (where each block contains 100 Txns).

$D=8$	$D=9$	$D=10$
5 possible solutions on average	2 possible solutions on average	1 solution

Fig. 7: Possible solutions on different difficulties.

A. Scalability of ResilientDB

First, we illustrate the scalability of ResilientDB fabric. This experiment serves as a baseline for the future experiments, illustrating the impact of PoC on the system throughput. In Figures 5(a) and (b), we measure the peak throughput (transactions per second) and latency for Pbft consensus protocol on varying the block size ($\mathcal{S}$-block) from 1 to 1 k on a system of 128 replicas.

We observe that although Pbft hits its peak throughput at $\mathcal{S}$-block size of 150, the optimal $\mathcal{S}$-block size is 100 as the throughput is only 11.5% less than the peak, while the latency is 3× lower.

Beyond an $\mathcal{S}$-block size of 150, there is no increase in throughput because the queues that store messages at replicas are full and can no longer process newer requests, which increases the wait time (latency) for clients.

Next, in Figures 5(c)-(d), we increase the number of replicas from 16 to 128; $\mathcal{S}$-block size is 100. Unsurprisingly, on increasing the number of replicas, there is a drop in the peak throughput (consequential increase in latency) because there is a corresponding increase in the number of messages communicated per consensus; on moving from 16 to 128 replicas, the throughput drops by 86.8%.

B. Scalability of PoC

In this set of experiments, we measure the time it takes for various mining schemes to append a block to the ledger. In Figure 6, we compare PoC against three baselines: (1) Bitcoin's PoW consensus, where each miner selects value uniformly at random and checks if it is a valid nonce. (2) PoW consensus where each miner sequentially iterates over each value in the search space (starting from 0) until it finds a valid nonce, (3) PoW with 30% miners running PoC, which allows us to approximate the impact of the largest centralized mining pool in Bitcoin [39].

In this experiment, we want to meet the following three goals: (1) The mining scheme reaches the same throughput (blocks added to the ledger per second) as ResilientDB's throughput at 128 replicas, which is approximately 100k txns/s (or 1k blocks per second). (2) Aggregate precise number of $\mathcal{S}$-blocks into a mined block so that we can observe least latency (difference between time an $\mathcal{S}$-block is received to the time it is added to the ledger). (3) PoC can address multiple solutions. For example, for PoC, the time to mine a block at $D=8$ is 3s, so we add 3k $\mathcal{S}$-blocks in each mined block. The

latency for each experiment is approximately twice the mining time because each transaction waits for a time equivalent to the mining time during the block generation phase and the mining process. We observe that sequential PoW mining requires at least $2\times$ more mining time and latency than randomized mining. In comparison, PoW with 30% mining pool does reduce the mining time and latency. However, at $D = 10$, PoC yields up to $4.2\times$ and $29\times$ less mining time than 30% PoC and PoW, respectively. In Figure 7, we also demonstrate that multiple solutions may be located in different slices when D is smaller. Like at $D = 8$, each block contains around 5 solutions. However, PoC ensures that each miner obtains the same solution to guarantee the safety of the block.

Next, in Figures 9(a) and (b), we increase the number of miners from 64 to 128 while fixing the difficulty, $D = 8$. For each setting, there is a specific size of mined block at which it sustains the throughput of PBFT and achieves least latency. We observed 3k, 5k, and 7k to be such block sizes. We know that the peak throughput of PBFT is 100k txns/s and latency at $D = 8$ is around 6s. Thus, a PoC protocol with 64 miners hits the peak performance at 7k while a PoC protocol with 96 miners hits peak performance at 5k. Thus, we can conclude that PoC is non-invasive and has minimal impact on the system throughput.

C. Resilience to Failures

Next, we illustrate the effect of failures on PoC by studying two types of failures: 1 malicious miner and No nonce (§ IV-B) in Figures 9(c) and (d). In the malicious miner experiment, we simulate a Byzantine miner, which does not broadcast the solution of the 10-th block, which causes remaining miners to timeout and perform one round of the slice-shifting protocol. In the no nonce experiment, we ensure that no nonce satisfies the 10-th block, which leads to $\mathbf{f}_{\mathcal{M}}+1$ rounds of slice shifting. These experiments cause the latency to shoot up for the 10-th block (up to $5\times$ for the no nonce case). To quickly bring latency and throughput to the steady state, we merge the blocks in the no nonce case.

D. Appending PoC to Blockchains in the Wild.

Finally, we illustrate that PoC can be integrated to state-of-the-art blockchain systems to guard them against long-range attack with minimal impact. We append PoC to *four* blockchains part of the Diablo [27] framework: Diem [28], Algorand [3], Quorum [29] and Ethereum [2]. Our primary goal is to showcase that appending PoC to these blockchains causes minimal impact on their performance.

Diablo provides access to a framework, written in Golang, for evaluating popular blockchain systems. Diablo is composed of three types of nodes: *primary*, *secondary*, and *chain* nodes. The primary node is responsible for generating transactions and delivering them to the chain nodes via secondary nodes (mimicking a mempool functionality). The secondary nodes assist the primary node in reducing resource overhead, e.g., CPU and network bandwidth, by disseminating transactions to the chain nodes and collecting results. The chain nodes run various blockchains.

To incorporate ResilientDB into Diablo framework, we developed a Go SDK that provides an interface to send transactions from Diablo to ResilientDB. To append PoC to different chains, we implement a `Go-Server` agent on each chain node that periodically fetches blocks from the local chain on the chain node through their SDKs implemented using Golang. Specifically, the Go-Server provides a unified entry for PoC to obtain the block data. Note that in Diablo clients submit requests in an open loop.

Setup. Like Diablo authors, we use AWS c5.9xlarge (36 vCPUs, 72 GiB memory) machines and deploy one primary and 10 secondaries. We deploy 10 PoC miners within PoC. We use the workloads provided by Diablo. The maximum observed throughput (in transactions per second or TPS) of these blockchains as per the original paper is: NasDAQ (168 TPS), Visa (1000 TPS), Fifa (3483 TPS), Dota 2 (13303 TPS), and Youtube (37976 TPS).

Results. Figure 8 summarizes our findings. Diem can sustain 1000 TPS and Quorum only 204 TPS; their performance drops significantly when the workload increases. Ethereum attains a consistently low throughput in part due to its default block generation period (15 seconds). Algorand exhibits a more stable performance of 500-600 TPS across workloads. All blockchains suffer from higher latency (as expected) when the load increases, in a sense, artificially over-saturating the system. We observe that ResilientDB can achieve a high throughput of 37,967 TPS, which nearly matching the injected load. When PoC is added to any system, we observe that the throughput drops at most by 10%-20% due to an increased communication and added network latency. We argue that this is a negligible cost as PoC helps these blockchains prevent long-range attacks, which continues to be a major vulnerability in their design.

VI. Related work

BFT has been studied extensively in the literature: (1) linearizing BFT consensus [4], [7], [40], [5], [41], (2) optimizing for geo-replication [42], [43], (3) concurrency and sharding [44], [45], [46], [47], [48], [49], [50], [51], [52], (4) optimizing for collaboration [53], [54], (5) leveraging security components [55], [56], and (6) framework [57], [58], [59]. Nevertheless, all of these protocols face long-range attacks [60], [21].

Alternatively, prior works have focussed on designing PoS protocols that permit the node with the highest stake to propose the next block [61], [3], [30]. However, even these protocols suffer from long-range attacks if adversary has access to the private keys.

Existing work to protect against long-range attacks includes: (1) checkpointing the state through a trusted committee [8], [62], [63], (2) key-evolving cryptographic techniques [64], [65], [3], (3) verifiable delay functions [66], and (4) appending the state to Bitcoin [11], [21].

With PoC, we show how to make it computationally expensive for an adversary to rewrite the ledger while forcing miners to collaborate and conserve their computational resources.

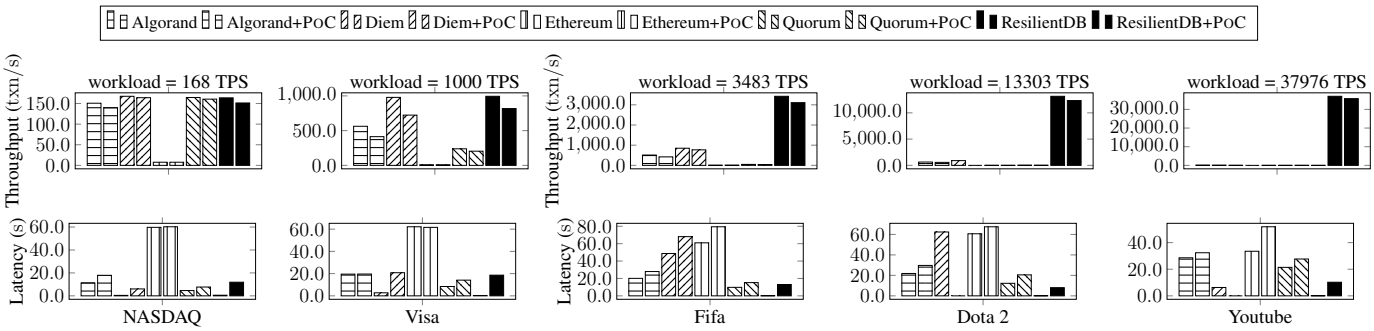

Fig. 8: Impact of appending PoC (running at difficulty $D = 8$) to different Diablo blockchains.

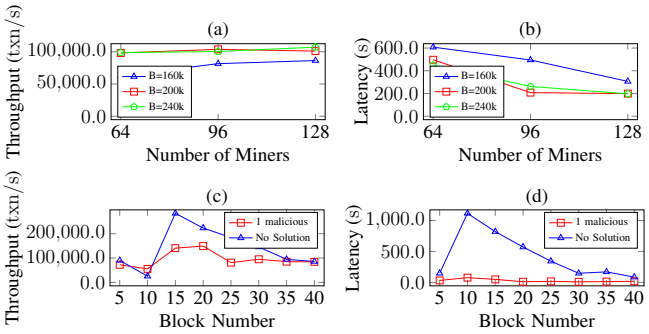

Fig. 9: Peak throughput and average latency attained by PoC under different conditions at $D = 8$.

Further, we show experimentally that collaboration helps PoC to waste fewer resources for a given difficulty.

VII. Conclusions

In this paper, we presented our novel PoC protocol, which, when appended to existing PoS/BFT protocols, guards them against long-range attacks. Like PoW, PoC makes it computationally expensive for an adversary to rewrite the ledger. However, unlike PoW, PoC introduces collaborative mining that requires miners to work with each other instead of competing.

Acknowledgment

This work is partially funded by NSF Award Number 2245373 and Mysten Labs.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2015. [Online]. Available: http://gavwood.com/paper.pdf

[3] 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*. New York, NY, USA: Association for Computing Machinery, 2017, p. 51–68.

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

[5] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356.

[6] S. King and S. Nadal, "PPCoin: Peer-to-peer crypto-currency with Proof-of-Stake," 2012. [Online]. Available: https://www.peercoin.net/whitepapers/peercoin-paper.pdf

[7] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, pp. 7:1–7:39, 2009.

[8] S. Azouvi, G. Danezis, and V. Nikolaenko, "Winkle: Foiling long-range attacks in proof-of-stake systems," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. New York, NY, USA: Association for Computing Machinery, 2020, p. 189–201.

[9] M. K. Aguilera, I. Keidar, D. Malkhi, J.-P. Martin, A. Shraer *et al.*, "Reconfiguring replicated atomic storage: A tutorial," *Bulletin of the EATCS*, no. 102, pp. 84–108, 2010.

[10] Y. Wang, J. Sun, X. Wang, Y. Wei, H. Wu, Z. Yu, and G. Chu, "Sperax: An approach to defeat long range attacks in blockchains," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 574–579.

[11] E. Tas, D. Tse, F. Gai, S. Kannan, M. Maddah-Ali, and F. Yu, "Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities," in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 126–145.

[12] Y. Yun, "Lazarus group's favorite exploit revealed — crypto hacks analysis," 2024. [Online]. Available: https://cointelegraph.com/magazine/north-korean-hackers-private-keys-flash-loan-attacks/

[13] R. Nambiampurath and K. Baird, "2,000 crypto private keys stolen from edge wallet," 2023. [Online]. Available: https://beincrypto.com/2000-crypto-private-keys-stolen-edge-wallet/

[14] E. R, "Private key breaches surge in 2024 with over $239 million stolen! is your crypto safe?" 2024. [Online]. Available: https://coinpedia.org/news/private-key-thefts-skyrocket-in-q1-2024-with-over-239-million-stolen-in-just-three-months/

[15] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, p. 34–37, dec 2014.

[16] M. Nijkerk, "Ethereum Unstaking Requests Now Face About a 17-Day Wait," 2023. [Online]. Available: https://www.coindesk.com/tech/2023/04/18/ethereum-unstaking-requests-now-face-about-a-17-day-wait/

[17] M. Sherman, "Nodes on Bitcoin's Lightning Network Double in 3 Months," 2021. [Online]. Available: https://www.coindesk.com/tech/2021/07/14/nodes-on-bitcoins-lightning-network-double-in-3-months/

[18] M. K. Franklin, "A survey of key evolving cryptosystems," *Int. J. Secur. Networks*, vol. 1, no. 1/2, pp. 46–53, 2006. [Online]. Available: https://doi.org/10.1504/IJSN.2006.010822

[19] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and casper," *CoRR*, vol. abs/2003.03052, 2020.

[20] E. N. Tas, D. Tse, F. Yu, and S. Kannan, "Babylon: Reusing bitcoin mining to enhance proof-of-stake security," *arXiv preprint arXiv:2201.07946*, 2022.

[21] S. Azouvi and M. Vukolić, "Pikachu: Securing pos blockchains from long-range attacks by checkpointing into bitcoin pow using taproot," *arXiv preprint arXiv:2208.05408*, 2022.

[22] A. de Vries, "Bitcoin's growing energy problem," *Joule*, vol. 2, no. 5, pp. 801–805, 2018.

[23] L. Luu, Y. Velner, J. Teutsch, and P. Saxena, "SmartPool: Practical decentralized pooled mining," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1409–1426.

[24] "P2Pool," 2011. [Online]. Available: http://p2pool.in/

[25] N. Dana Troutman and A. Laszka, "Poolparty: Efficient blockchain-agnostic decentralized mining pool," in *2021 The 3rd International Conference on Blockchain Technology*. New York, NY, USA: Association for Computing Machinery, 2021, p. 20–27.

[26] "Apache resilientdb (incubating)," 2024. [Online]. Available: https://resilientdb.incubator.apache.org/

[27] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo-v2: A benchmark for blockchain systems," 2012. [Online]. Available: https://infoscience.epfl.ch/record/294268

[28] Diem Association, "Diem bft," 2022. [Online]. Available: https://www.diem.com/en-us/

[29] J. Chase, "Quorum whitepaper," 2019. [Online]. Available: https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf

[30] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds. Cham: Springer International Publishing, 2018, pp. 66–98.

[31] Cryptopedia Staff, "What was the dao?" 2023. [Online]. Available: https://www.gemini.com/cryptopedia/the-dao-hack-makerdao

[32] K. Okupski, "Bitcoin developer reference," 2016. [Online]. Available: https://github.com/minium/Bitcoin-Spec

[33] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *Journal of the ACM*, vol. 71, no. 4, pp. 1–49, 2024.

[34] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM (JACM)*, vol. 36, no. 2, pp. 335–348, 1989.

[35] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case latency of byzantine broadcast: A complete categorization," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 331–341.

[36] J. Chen, S. Gupta, A. Sonnino, L. Kokoris-Kogias, and M. Sadoghi, "Securing consensus from long-range attacks through collaboration," 2025. [Online]. Available: https://arxiv.org/abs/2302.02325

[37] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2009, pp. 153–168.

[38] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1085–1100.

[39] Blockchain.com, "Hashrate Distribution: An estimation of hashrate distribution amongst the largest mining pools." 2023. [Online]. Available: https://www.blockchain.com/explorer/charts/pools

[40] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi, "Proof-of-Execution: Reaching consensus through fault-tolerant speculation," in *Proceedings of the 24th International Conference on Extending Database Technology*, 2021.

[41] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A scalable and decentralized trust infrastructure," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2019.

[42] Y. Amir, C. Danilov, J. Kirsch, J. Lane, D. Dolev, C. Nita-Rotaru, J. Olsen, and D. Zage, "Scaling byzantine fault-tolerant replication to wide area networks," in *International Conference on Dependable Systems and Networks (DSN'06)*, 2006, pp. 105–114.

[43] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "ResilientDB: Global scale resilient blockchain fabric," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 868–883, 2020.

[44] J. Hellings and M. Sadoghi, "Byshard: Sharding in a byzantine environment," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2230–2243, 2021.

[45] H. Lyu, S. Xie, J. Niu, I. Beschastnikh, Y. Zhang, M. Sadoghi, and C. Feng, "Orthrus: Accelerating multi-bft consensus through concurrent partial ordering of transactions (extended version)," 2025. [Online]. Available: https://arxiv.org/abs/2501.14732

[46] J. Chen, A. Sonnino, L. Kokoris-Kogias, and M. Sadoghi, "Thunderbolt: Concurrent smart contract execution with non-blocking reconfiguration for sharded dags," 2025. [Online]. Available: https://arxiv.org/abs/2407.09409

[47] D. Kang, S. Rahnama, J. Hellings, and M. Sadoghi, "Spotless: Concurrent rotational consensus made practical through rapid view synchronization," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 1916–1929.

[48] S. Gupta, J. Hellings, and M. Sadoghi, "RCC: resilient concurrent consensus for high-throughput secure transaction processing," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 1392–1403. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00124

[49] D. Kang, S. Gupta, D. Malkhi, and M. Sadoghi, "Hotstuff-1: Linear consensus with one-phase speculation," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–29, 2025.

[50] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, pp. 165–175.

[51] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[52] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2705–2718.

[53] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th usenix security symposium (usenix security 16)*, 2016, pp. 279–296.

[54] C. Rondanini, B. Carminati, F. Daidone, and E. Ferrari, "Blockchain-based controlled information sharing in inter-organizational workflows," in *2020 IEEE International Conference on Services Computing (SCC)*, 2020, pp. 378–385.

[55] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, *Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX*. New York, NY, USA: Association for Computing Machinery, 2020, p. 955–970.

[56] I. Messadi, M. H. Becker, K. Bleeke, L. Jehl, S. B. Mokhtar, and R. Kapitza, "Splitbft: Improving byzantine fault tolerance safety using trusted compartments," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 56–68.

[57] M.-K. Sit, M. Bravo, and Z. István, "An experimental framework for improving the performance of bft consensus for future permissioned blockchains," in *Proceedings of the 15th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 55–65.

[58] M. J. Amiri, C. Wu, D. Agrawal, A. El Abbadi, B. T. Loo, and M. Sadoghi, "The bedrock of byzantine fault tolerance: A unified platform for {BFT} protocols analysis, implementation, and experimentation," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 371–400.

[59] M. J. Amiri, C. Wu, D. Agrawal, A. E. Abbadi, B. T. Loo, and M. Sadoghi, "The bedrock of BFT: A unified platform for BFT protocol design and implementation," *CoRR*, vol. abs/2205.04534, 2022.

[60] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, "A survey on long-range attacks for proof of stake protocols," *IEEE Access*, vol. 7, pp. 28 712–28 725, 2019.

[61] M. Kohlweiss, V. Madathil, K. Nayak, and A. Scafuro, "On the anonymity guarantees of anonymous proof-of-stake protocols," in *42nd IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 1818–1833.

[62] V. Buterin, "Proof of stake: How i learned to love weak subjectivity," 2014. [Online]. Available: https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity

[63] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2019, p. 23–41.

[64] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee, "Pixel: Multi-signatures for consensus," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2093–2110.

[65] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO 2017*. Cham: Springer International Publishing, 2017, pp. 357–388.

[66] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13," 2019. [Online]. Available: https://solana.com/solana-whitepaper.pdf