Resilient Consensus Sustained Collaboratively

Junchao Chen, Suyash Gupta[†], Alberto Sonnino[‡], Lefteris Kokoris-Kogias[§‡], Mohammad Sadoghi
Exploratory Systems Lab
University of California, Davis
[†]University of California, Berkeley [‡] MystenLabs [§] IST Austria

ABSTRACT

The recent growth of blockchain technology has accelerated research on decentralized platforms. Initial such platforms decide on what should be added to the ledger based on the Proof-of-Work (PoW) consensus protocol. PoW protocol requires its participants to perform massive computations and leads to massive energy wastage. Existing solutions to replace PoW protocol make use of Proof-of-Stake (PoS) protocol or classical fault-tolerant consensus protocols. However, the safety of ledger created by these protocols is at the mercy of the long-term safe-keeping of private keys of participants subject to long-range attacks. To ameliorate this situation, we present the design of our novel HybridChain architecture, which requires each client transaction to undergo two consensus protocols: a fault-tolerant consensus followed by our novel Power-of-Collaboration (PoC) protocol. Despite this, we observe that our HybridChain system outperforms state-of-the-art blockchain systems yielding up to 2000× higher throughput and 10^5 times less energy costs.

PVLDB Reference Format:
Junchao Chen, Suyash Gupta[†], Alberto Sonnino[‡], Lefteris Kokoris-Kogias[§‡], Mohammad Sadoghi. Resilient Consensus Sustained Collaboratively. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/msadoghi/poc.

1 INTRODUCTION

The key goal of any blockchain system is to offer its clients a transparent and accountable ledger [67]. This ledger is maintained by multiple untrusting parties (servers) that add each client transaction to the ledger in an ordered manner by participating in a malicious fault-tolerant (MFT[1]) consensus protocol. Initial blockchain systems such as Bitcoin [54] and Ethereum [76] employed the *Proof-of-Work* (PoW) protocol [54, 76], which follows a *computation-oriented* consensus model. PoW protocol requires all its participants to compete toward solving a complex puzzle. Whichever participant solves the puzzle first, gets to add a new entry (*block*) to the ledger. However, PoW protocol is infamous for massive energy wastage [21, 75].

This motivated the blockchain community to adopt two other categories of protocols: (1) Proof-of-Stake (PoS) protocols and (2) traditional MFT consensus protocols. PoS protocols advocate for a *stake-oriented* model where the node with the highest stake (or wealth) gets to add a new block to the ledger [20, 44]. Traditional MFT protocols advocate for an authenticated *communication-oriented* architecture, where each node gets an equal chance to add an entry to the ledger; agreement on the next block is reached through successive rounds of vote exchange [13, 48]. However, both of these types of protocols suffer from *long-range attacks* [7, 22]. These long-range attacks pose unprecedented dangers; a malicious party can re-write the *full history* of the ledger.

A key reason why PoS and MFT protocols suffer from long-range attacks is that, in these protocols, adding a new block to the ledger is *computationally inexpensive*. Each block added to the ledger is only digitally signed by the participants. To perform a long-range attack, the adversary needs to compromise the private keys of the honest participants. Unsurprisingly, such attacks are common [7, 22] and using these private keys, the adversary can create its own ledger.

In this paper, we resolve these challenges through our novel HybridChain design, which offers efficient and tamper-free logging of client transactions by requiring them to undergo two distinct *consensus* protocols: a traditional MFT protocol like PBFT, followed by our novel *Power-of-Collaboration* (PoC) protocol.[2] Although this may seem redundant at first, it allows HybridChain to prevent long-range attacks while ensuring high throughput, low latency, and substantially reduced energy consumption in comparison to prior attempts [8, 29, 74]. In our HybridChain system, each client transaction is first ordered with the help of a traditional MFT protocol, which results in fast *commitment* and *low-latency response* for the clients. Next, this ordered transaction is forwarded to a set of miners, which run our novel PoC protocol to add the transaction to the tamper-proof ledger. We refer to this PoC ordering as lazy *settlement* because it does not delay the response to the client.

To guarantee a tamper-proof ledger that prevents long-range attacks, PoC (like PoW) requires miners to solve compute-intensive puzzles. However, PoC avoids being an energy guzzler by ensuring that no miner's work goes to waste. It does so by requiring all the miners to *collaborate* to solve the compute-intensive puzzle. This collaboration also ensures that any malicious miner that wants to re-write the ledger needs to have more computational power than the combined power of all the honest parties. Of course, malicious miners may avoid collaboration, which may momentarily waste

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[1] In this paper, we intentionally refrain from the commonly accepted terminology of Byzantine behavior that carries a negative connotation as noted in [56], We further adhere to ACM's D&I policy, and instead of using the traditional phrase. Byzantine Fault-Tolerant (BFT), we use the term Malicious Fault-Tolerant (MFT) to refer to protocols that can handle malicious attacks.

[2] The sketch of our collaborative consensus idea was presented as an extended abstract in [14].

the resources of honest miners. But, we can identify such miners and penalize them for their malicious behavior.

In PoW-based systems, as miners constantly compete with each other to propose the next block, forks of the ledger (and selfish mining attacks) are common [21, 27]. The frequency of these forks increases as new miners join the system. As a result, these PoW systems need to constantly increase the difficulty of solving the puzzle, which also increases energy consumption. In our HybridChain system, the transaction ordering by mft protocol ensures no forking of the ledger. Further, PoC leverages the mft consensus to *select and commit* a solution for the puzzle, which guarantees *fixed energy consumption* (except in the case of a technological hardware change) per transaction.

To prove that our HybridChain system is effective in practice, we implement it on the NexRes version of ResilientDB [35–39, 61, 62] and compare its performance against *five* state-of-the-art blockchain systems: Hyperledger Fabric [3], Cardano [20], DiemBFT [23], Avalanche [63], and Ethereum [76]; these blockchain systems hold a combined market cap of nearly 200 billion dollars.[3] We deploy up to 120 replicas and 120 miners, and our results illustrate that HybridChain achieves up to 2000× higher throughput and consumes up to 10^5 less energy than the aforementioned systems. Next, we list our contributions.

- We present the HybridChain architecture that prevents long-range attacks on the ledger and guarantees fast responses to the clients.
- We present the PoC protocol, which prevents an honest miner's work to go waste and ensures that to re-write the ledger, a malicious miner needs more resources than the combined power of honest miners.
- We make novel use of mft consensus protocols to guarantee high-throughput and low-latency commitment, to prevent forks of the ledger and to guarantee fixed energy consumption for PoC.

2 PRELIMINARIES

We adopt the standard communication and failure model adopted by most mft protocols [13, 32, 48]. We consider a service $\mathcal{S}$ of the form $\mathcal{S} = \{\mathcal{R}, \mathcal{M}, \mathcal{C}\}$. The set $\mathcal{R}$ consists of $\mathbf{n}_{\mathcal{R}}$ replicas of which at most $\mathbf{f}_{\mathcal{R}}$ can behave arbitrarily. The remaining $\mathbf{n}_{\mathcal{R}} - \mathbf{f}_{\mathcal{R}}$ are honest: they will follow the protocol and remain live. Similarly, the set $\mathcal{M}$ consists of $\mathbf{n}_{\mathcal{M}}$ miners of which at most $\mathbf{f}_{\mathcal{M}}$ can act maliciously. We assign each miner and replica a unique identifier, which can be obtained by a call to the function id(). The range of these identifiers are $[0, |\mathcal{R}|]$ for replicas and $[0, |\mathcal{M}|]$ for miners. We further consider the existence of a finite set of clients C of which arbitrarily many can be malicious. **Authenticated communication**: replicas/miners employ standard cryptographic primitives such as Mac and digital signatures (DS) to sign messages. We denote a message m signed by a replica r using DS as $\langle m \rangle_{\mathbf{R}}$. We employ a *collision-resistant* hash function $\mathsf{hash}(\cdot)$ to map an arbitrary value v to a constant-sized digest $\mathsf{hash}(v)$ [43]. Each replica/miner only accepts a message if it is ***well-formed***.

Anonymity. Existing permissionless blockchains like Bitcoin support *psuedo-anonymity*; miners are identified only through their public keys, which they may hold many. However, permissioned blockchains (e.g. DiemBFT), require the identities of replicas to be known and verified before consensus. HybridChain requires the identities of replicas to be known and provides miners with same pseudo-anonymity as Bitcoin (more discussion in § 6).

Guarantees. We adopt the same partial synchrony model adopted in most consensus systems: both mft consensus and PoC mining guarantees safety in an asynchronous environment where messages can get lost, delayed or duplicated. However, liveness is only guaranteed during the periods of synchrony [13, 32, 77]. Additionally, we need to incentivize PoC miners, which requires synchrony to ensure fairness.

Safety. If two honest replicas r1 and r2 order a transaction T at sequence numbers k and k', then $k = k'$.

Liveness. If a client sends a transaction T, then it will eventually receive a response for T.

Incentive-compatibility. No honest miner is penalized if it solves its part of the PoW puzzle and the network is undergoing a period of synchrony.

3 BACKGROUND

Next, we present some necessary background concepts.

3.1 Proof-of-Work Consensus

Initial blockchain applications, such as Bitcoin and Ethereum employ the PoW consensus to add transactions to the ledger. Prior to running the PoW protocol, each miner[4] $\text{m} \in \mathcal{M}$ selects some client transactions and packs them in a block. This block includes a header, which contains: (i) the *hash* of the previous block, (ii) the *Merkle root* of all transactions, (iii) D, which determines the difficulty of the puzzle, and (iv) *nonce*, the solution of the puzzle, among other fields [40, 57].

Computing the Merkle root of all transactions requires a miner m to compute a pairwise hash from the leaf to the root. This Merkle root helps to verify if a transaction was included to create the Merkle tree. The main challenge for any PoW miner is to determine the nonce that solves the complex PoW puzzle, which is essentially a *desired hash* of the block (having a specific number of leading zero bits). For this purpose, the miners iterate through different *nonce* values and rehash the block until they reach the desired hash. Whichever miner discovers a *valid* nonce first, it gets to propose the next block to be added to the chain. The difficulty of finding the nonce is controlled through the system parameter D.

PoW Challenges. In blockchain applications running PoW consensus, *forks* are a common occurrence; multiple miners may propose the next block with valid nonces at approximately the same time. In such a case, the protocol states that each miner would only accept the first block it receives. This could lead to temporary branches or *forks*, all of which have the same previous hash. However, these applications assume that this condition would resolve as time passes because the honest miners would stick to the *longest chain*—the one with the largest number of blocks. Eventually, all the shorter forks are discarded, and only the longest chain survives. Consequently, to reduce the probability of forks, PoW-based applications periodically increase the difficulty in finding the nonce. As

[3] https://www.coingecko.com/

[4] In PoW, machines that participate in consensus are referred to as miners.

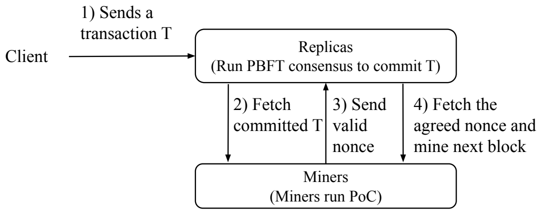

Figure 1: Transactional flow in the HybridChain system.

a result, the PoW protocol is often viewed as an energy guzzler: (1) miners need to perform massive computations to find the nonce, and (2) only one block is added to the chain (the rest are discarded).

Incentives. Considering energy costs, PoW-based systems award their miners incentives to act honestly and follow the protocol. Generally, there are two main sources for the awards: *winner's reward* and *transaction fees*. The winner's reward is given to the miner who successfully adds a new block to the chain, whereas each client pays a fee for its transaction to be added to the block.

3.2 Proof-of-Stake Consensus

Due to the limitations of the PoW protocol, several recent blockchain applications have advocated a switch to the PoS consensus protocol [20, 31]. In the PoS protocols, participants earn the opportunity to add the next block to the blockchain based on their invested stake. Often, the stake is equivalent to a monetary token or currency. As a result, the higher the stake a miner invests, the greater are the opportunities it receives to add a block. Once a stakeholder proposes the next block, all the other participants also sign this block, which acts like an agreement among the participants.

3.3 Long-range attack: PoS and mft Challenge

Among many other challenges, the key challenge for any PoS-based application is long-range attacks like *posterior corruption* [7, 30]. This attack aims to create an alternate history by re-writing the ledger. To do so, a malicious party tries to compromise the private keys of the honest parties. Once an adversary has access to the private keys of honest stakeholders, it can create desirable blocks and sign them with the compromised keys. In the following example, we illustrate a long-range attack.

Example 3.1. Assume a PoS blockchain: $\mathfrak{B}_1, \mathfrak{B}_2, ...\mathfrak{B}_k, ..., \mathfrak{B}_n$. Say malicious stakeholders have access to the private keys of all the honest stakeholders and decide to create an alternate ledger, starting from the k-th block. Further, these malicious parties can introduce newer blocks. So, the blockchain at malicious stakeholders looks as follows: $\mathfrak{B}_1, \mathfrak{B}_2, ...\mathfrak{B}'_k, ..., \mathfrak{B}'_n, \mathfrak{B}'_{n+1}$. Any new stakeholder joining the system cannot distinguish between these two chains, and will select the longest chain. Similarly, the existing honest stakeholders will forfeit their chain and switch to the malicious chain.

In hindsight, if the private keys of honest parties are compromised, all the bets are off. A common assumption made by existing mft and PoS protocols is that *malicious parties cannot impersonate honest parties*. But, recent security breaches [5, 41] illustrate that hackers can easily compromise millions of accounts, which in turn allows malicious parties to impersonate and forge messages or blocks. Hence, the security of any ledger should not solely rely on the security of private keys of its administrators.

This highlights an unprecedented fragility in the design of existing mft and PoS systems: the only proof attached to each block is the signatures of a majority of participants. As the consensus mechanism of these protocols is computationally cheap, any malicious party that has access to the private keys of honest parties can trivially create an alternate ledger or *history*. This is where PoW-based systems differ; to rewrite the ledger, a malicious party has to solve computationally expensive puzzles. Longer the blockchain, the harder to replace existing blocks with desired blocks; not to forget that new blocks are being proposed continuously.

Existing attempts to resolve long-range attacks are either unscalable or unsecure [7, 25]. We discuss these in § 9.

4 HYBRIDCHAIN ARCHITECTURE

Owing to the long-range attacks on PoS and mft blockchains and energy wastage by PoW blockchains, we present our HybridChain design, which requires each client transaction to undergo two consensuses before it is written to the ledger. The two consensus protocols are run in a pipelined manner, which facilitates processing multiple client transactions at the same time.

We use Figure 1 to illustrate the transactional flow in our HybridChain system. Each client sends its transaction to the replicas participating in a mft protocol like Pbft, PoE [35], and HotStuff [77]. Once the replicas receive a client transaction, they run a mft consensus protocol to order this request (*commitment phase*). Periodically, miners fetch the next committed transaction from the replicas. Following this, the miners collaboratively run our PoC consensus protocol to add this transaction to the next block in the ledger. This requires finding a valid nonce, which they forward to the replicas. Once replicas have agreed on the nonce, the miners fetch the nonce and add the block to the ledger (*settlement phase*). This design offers following appealing properties:

(G1) **Tamper-proof ledger.** HybridChain protects the ledger against long-range attacks like posterior corruption.

(G2) **High throughput and low latency.** Like mft and PoS protocols, HybridChain offers clients high throughput and low latency processing; clients receive responses to their transactions once they are committed by mft replicas.

(G3) **No forks and selfish mining attacks.** HybridChain prevents forking of the ledger and selfish mining by malicious miners; miners run PoC on committed blocks.

(G4) **Minimal energy costs.** Unlike PoW, HybridChain expends minimal energy and wastes fewer resource as PoC miners collaborate to solve the compute-intensive puzzle. Further, mft consensus stabilizes the difficulty of the PoC computations.

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

4.1 Client Request and Transaction Ordering

The first step in our HybridChain architecture is to *order* incoming client transactions. We term this step as commitment phase because it gives the client a guarantee that *eventually* its transaction will be written to the ledger. To drive the commitment phase, we ensure that each transaction undergoes a traditional mft consensus. We

```
Client-role (used by client $c$) :
event $c$ wants to process a transaction $T$ do
    Sends $\langle T \rangle_c$ to the primary P.
event $c$ receives $q := \textsc{Response}(\langle T \rangle_c, k, r)$ messages from $\mathbf{f}_{\mathcal{R}} + 1$ replicas such that:
        (1) each message $q$ is well-formed and is sent by a distinct replica R $\in \mathcal{R}$.
        (2) all the messages are identical.
do
    Considers $T$ executed, with result $r$, as the $k$-th transaction.
```

Figure 2: HybridChain client protocol.

rely on traditional MFT protocols because these protocols : (1) generate a single order for all the transactions, and (2) do not require replicas to compete, which prevents the forking of the chain.

For the commitment phase, **our HybridChain system can adopt any MFT protocol**. In this paper, we employ the Pbft [13] protocol as all the other MFT protocols follow the consensus dictated by Pbft [32, 48, 77]. Pbft follows the primary-backup model where one replica is designated as the *primary* while other replicas act as backups. Each consensus is led by the primary of the current *view*. In the case the primary is malicious, *view-change* takes place to replace the primary. When the primary is honest, Pbft requires three phases to reach consensus on a client transaction. We term this as *civil execution* and we explain these phases next.

Client Request. A client c that wants to process a transaction T creates a request $\langle T \rangle_c$ and sends it to the primary replica of the view v. The client c uses DS to sign this message and adds a monotonically increasing timestamp to this message. In Figure 2, we present the client algorithm.

Pre-prepare. When the primary replica P receives a well-formed client request $m := \langle T \rangle_c$, it assigns m a sequence number k and sends a Preprepare message to all the replicas (refer to Figure 3 for algorithm run by replicas). This Preprepare message also includes a digest $\mathsf{hash}(m)$ of m, which is used in future communication to save space. During this phase, it is sufficient for the primary to sign the messages using Mac. When a replica R $\in \mathcal{R}$ receives a well-formed Preprepare message from the primary P of view v, it agrees to support the order k for m if it has not agreed to order another request at sequence number k. The replica R shows its support by broadcasting a Prepare message.

Prepare. When a replica R receives identical Prepare messages from $2\mathbf{f}_{\mathcal{R}} + 1$ distinct replicas (can count its own message), it marks the request m as *prepared* and broadcasts a Commit message. In HybridChain, we require each replica R to use DS to sign the Commit message.

Commit. When R receives identical Commit messages from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas, it marks m as *committed*. If R has executed all requests with a sequence number less than k, it executes m and sends a Response message to the client, which includes the result of execution r. The client c marks $\langle T \rangle_c$ as *committed* when it receives identical Response messages from at least $\mathbf{f}_{\mathcal{R}} + 1$ replicas.

4.2 Chain Communication: IDA

Post Pbft consensus on m, each replica R runs the Information Dispersal Algorithm (IDA) [60] to encode the committed transaction. We use the IDA algorithm for the following two reasons:

(1) To reliably transmit the committed message between the replicas and miners despite up to $\mathbf{f}_{\mathcal{R}}$ malicious replicas.

(2) As IDA splits a message into multiple parts, and we require each replica to communicate only one of these parts per miner, the communication complexity of this phase becomes linear (*conserves the bandwidth*); though IDA is computationally expensive.

```
    Primary-role (running at the primary node P) :
 1: event P receives
            • $m := \langle T \rangle_c$ from client $c$, or
            • $m := \langle \textsc{NonceFind}(b, \eta) \rangle_{\mathrm{M}}$ from miner M.
    do
 2:     Calculate digest $\Delta := \mathsf{hash}(m)$.
 3:     Broadcast $\textsc{Preprepare}(\langle T \rangle_c, \Delta, k)$ to all replicas (order at sequence $k$).

 4: event P receives a certificate signed by $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners
            • $m := \langle \textsc{Penalty}(\mathcal{M}_{mal}, b, \mathbf{r}, \mathfrak{C}) \rangle_{\mathrm{M}}$ messages, or
            • $m := \langle \textsc{Shift}(b, \mathbf{r}, \mathfrak{C}) \rangle_{\mathrm{M}}$ messages.
 5: Verify the certificate $\mathfrak{C}$. do
 6:     Follow Line 2.

    Non-Primary role (running at a replicas R $\in \mathcal{R}$) :
 7: event R receives $\textsc{Preprepare}(\langle T \rangle_c, \Delta, k)$ from P such that:
            (1) message is well-formed, and R did not accept a $k$-th proposal from P.
    do
 8:     Broadcast $\textsc{Prepare}(\Delta, k)$ to all nodes in $\mathcal{R}$.

    All replicas role (running at each replica R (primary or non-primary)) :
 9: event R receives $\textsc{Prepare}(\Delta, k)$ messages from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas such that:
            (1) each message is well-formed and is sent by a distinct node, R$*$ $\in \mathcal{R}$.
    do
10:     Broadcast $\langle \textsc{Commit}(\Delta, k) \rangle_{\mathrm{R}}$ to all nodes in $\mathcal{R}$.

11: event R receives $\langle \textsc{Commit}(\Delta, k) \rangle_{\mathrm{R}}$ messages from $2\mathbf{f}_{\mathcal{R}} + 1$ replicas such that:
            (1) each message is well-formed and is sent by a distinct node, R$*$ $\in \mathcal{R}$.
    do
12:     if R has executed transaction with sequence number $k - 1 \land k > 0$ then
13:         Execute $T$ as the $k$-th transaction.
14:         Let $r$ be the result of execution of $T$ (if there is any result).
15:         Send $m' = \textsc{Response}(\langle T \rangle_c, k, r)$ to $c$.
16:         $m'_i$ := Run function IDA-split($m'$, $i$), where $i$ is this replica's identifier.
17:     else
18:         Place $T$ in queue for execution.

19: function IDA-split (message: $m'$, identifier: $i$)
20:     Run IDA scheme $(2\mathbf{f}_{\mathcal{M}} + 1, \mathbf{f}_{\mathcal{M}} + 1)$ on $m'$.
21:     Let the resulting parts be $m'_1, m'_2, ..., m'_q, ..., m'_{2\mathbf{f}_{\mathcal{M}}+1}$.
22:     Return $m'_q$ such that $q = i \bmod \mathbf{n}_{\mathcal{M}}$.
```

Figure 3: HybridChain replica protocol.

For this work, we use the IDA scheme $(2\mathbf{f}_{\mathcal{M}} + 1, \mathbf{f}_{\mathcal{M}} + 1)$. Assume that a replica R represents the committed transaction as $m' := \textsc{Committed}(k, m, r)$. Each replica R runs the IDA algorithm to split m' into $2\mathbf{f}_{\mathcal{M}} + 1$ parts (encodings). Let us denote these parts as $m'_1, m'_2, ..., m'_q, ..., m'_{2\mathbf{f}_{\mathcal{M}}+1}$. We require each i-th replica Ri to be responsible for the q-th part m'_q ($q = i \bmod \mathbf{n}_{\mathcal{M}}$). We illustrate this process in Figure 3, Lines 19 to 22.

Periodically, each miner M in set $\mathcal{M}$ sends a message to all the replicas in $\mathcal{R}$ to send their respective parts. Once M receives any of the distinct $\mathbf{f}_{\mathcal{M}} + 1$ parts, it can use them to reconstruct the message m'. Hence, the IDA scheme $(2\mathbf{f}_{\mathcal{M}} + 1, \mathbf{f}_{\mathcal{M}} + 1)$ requires each miner to wait for only $\mathbf{f}_{\mathcal{M}} + 1$ parts to recover the message.

4.3 Collaborative Mining

As stated previously, the miners in set $\mathcal{M}$ periodically fetch the next committed transactions from the Pbft replicas. They do so, to reliably append these committed transactions to the ledger; without securely appending the transactions, they can be subject to long-range attacks and rollbacked.

In our HybridChain system, to add transactions to the ledger, we require miners to participate in our PoC consensus protocol. Like PoW, our PoC protocol expects computational proofs to be associated with every transaction added to the ledger. However, unlike PoW, miners participating in the PoC consensus do not compete with each other because they work on transactions that

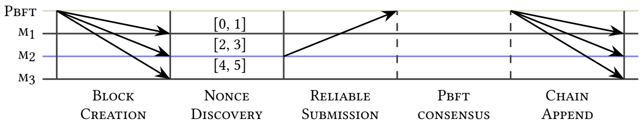

Figure 4: PoC protocol with miners $\mathcal{M} = \{\text{M}_1, \text{M}_2, \text{M}_3\}$. The solution space is $\mathcal{S} = [0, 5]$ and is divided into four slices ($[0, 1]$, $[2, 3]$, $[4, 5]$). Miners fetch transactions from the PBFT replicas. Post creating a block, each miner tries to discover a nonce in its slice. Assume the valid nonce is 2, which M_2 discovers and broadcasts to PBFT for commitment. Eventually, all miners fetch the nonce.

have been been ordered by MFT consensus. Instead, they collaborate, which significantly reduces energy consumption. PoC supports collaboration by dividing the PoW hash computation into $\mathbf{n}_{\mathcal{M}}$ disjoint sub-problems and requires each miner to work on a distinct predetermined sub-problem. Assume $\mathcal{S}$ is the solution space for a PoW hash computation; the set of numbers a miner has to try to find a valid nonce. In our PoC protocol, without the loss of generality, we divide the solution space $\mathcal{S}$ into $\mathbf{n}_{\mathcal{M}}$ equal slices, $\{\mathcal{S}_1, \mathcal{S}_2, \cdots \mathcal{S}_{\mathbf{n}_{\mathcal{M}}}\}$, such that:

$$\mathcal{S}_1 \cap \mathcal{S}_2 \cap \cdots \cap \mathcal{S}_{\mathbf{n}_{\mathcal{M}}} = \varnothing \quad \text{and} \quad \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_{\mathbf{n}_{\mathcal{M}}} = \mathcal{S}$$

Our PoC protocol assigns slice $\mathcal{S}_1$ to miner M_1, $\mathcal{S}_2$ to M_2, and $\mathcal{S}_i$ to M_i, $i \in [1, \mathbf{n}_{\mathcal{M}}]$. As each miner is working on a reduced solution space, so if a PoW miner takes time τ to find a valid nonce on the solution space $\mathcal{S}$, then in PoC, if all the miners are honest, the time required to find the nonce is $O(\frac{\tau}{\mathbf{n}_{\mathcal{M}}})$. Consequently, PoC leads to reduced energy consumption.

4.4 PoC Protocol Steps

From an outside view, it seems that our PoC protocol works in rounds, and within each round, each miner attempts to find a valid nonce in its slice. In the rest of this section, we assume that the solution space $\mathcal{S}$ can be deterministically divided into $\mathbf{n}_{\mathcal{M}}$ disjoint equal slices by each miner. For example, in Figure 4, the solution space $\mathcal{S} = [0, 5]$ is divided into $\mathbf{n}_{\mathcal{M}} = 3$ slices; the slices are: $\mathcal{S}_1 = [0, 1]$, $\mathcal{S}_2 = [2, 3]$, and $\mathcal{S}_3 = [4, 5]$.

Transaction Communication. As stated in Section 4.2, periodically, each miner in $\mathcal{M}$ queries the PBFT replicas for the next committed transactions. Prior works have illustrated that to reduce consensus costs, MFT protocols **batch** a set of transactions and run consensus on this batch. In our HYBRIDCHAIN system, we also employ batching and assume that in each round of PBFT consensus, each replica commits a batch of transactions (the number of transactions in a batch is a fixed system parameter). Similarly, each committed batch has a monotonically increasing sequence number: $k, k + 1, k + 2, ...$ Each PBFT replica runs the IDA scheme on a committed batch. Each miner $\text{M} \in \mathcal{M}$ asks the PBFT replicas to send their parts for the next committed batch; if $(k - 1)$-th batch was the last batch that M received, then M asks PBFT replicas for k-th batch (refer to Figure 6, Lines 1 to 3).

Block Creation. When a miner M has a valid nonce (η) for the block ordered at sequence $k - 1$ (Figure 6, Line 15), it initiates creation of the k-th block by picking up the next σ committed batches it received from PBFT replicas (Figure 6, Line 22). Specifically, in PoC, each **mined block** includes $\sigma > 0$ committed **batches**. The value of σ is a protocol parameter. Permitting each mined block to include σ batches helps to reduce the mining costs. Moreover, as each miner receives committed batches, aggregating consecutive batches together in a single block does not affect safety. In Figure 5, we illustrate the difference between PBFT batches and PoC blocks.

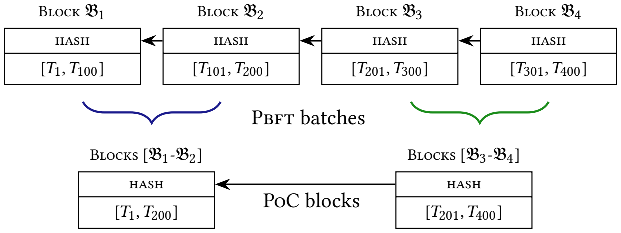

Figure 5: Batches and blocks in HYBRIDCHAIN system. PBFT replicas can run consensus on batches of transaction (say each batch has 100 transactions). PoC miners aggregate σ committed batches in a single block (say $\sigma = 2$) to mine.

As each mined block includes the Merkle root of all the transactions, PoC miners generate the Merkle root of σ batches. This task is easy to compute as we expect each committed batch to include the Merkle root of its transactions. So, the Merkle root of a PoC block is generated after hashing the Merkle roots of σ batches.

The only thing remaining is slice discovery. We assume that each miner is deterministically assigned a slice (§6.1). Specifically, the i-th miner M_i knows that a total of $\mathbf{n}_{\mathcal{M}}$ miners are participating in the PoC protocol. It uses this information to creates $\mathbf{n}_{\mathcal{M}}$ slices and assigns itself the i-th slice $\mathcal{S}_i$ (Figure 6, Line 43).

Nonce Discovery. Once a miner knows its slice, it starts nonce discovery (Figure 6, Line 44). We assume that each miner M knows the characteristics of the expected hash (the number of leading zeroes). The miner M uses this hash information to search in its slice for any value that yields the desirable hash. This process may require M to iterate over all the possible values in its slice.

Reliable Submission. Once a miner M computes the expected hash, it creates a message NONCEFIND, which includes the discovered nonce, and broadcasts this message to all the miners (Figure 6, Line 47). When a miner receives a NONCEFIND message, it stops its timers and broadcast to all the replicas in PBFT (Figure 6, Lines 51-53). When the primary replica P receives NONCEFIND message it creates a transaction that assumes a NONCEFIND message as its data (Figure 3, Line 1). Next, P initiates PBFT consensus on this *nonce transaction.* This nonce transaction is like any other transaction, and once committed, miners acquire it using IDA.

Chain Append. When a miner M_i receives the valid nonce for the σ-th block from the PBFT replicas, M_i appends this block to its local blockchain and terminates nonce discovery for the σ-th block (Figure 6, Lines 14-16). A note-worthy observation of our PoC protocol is that if the miner with the slice containing the valid nonce is honest, then our PoC protocol finishes after a single round of nonce discovery. Post nonce discovery, each miner starts working on the next block to be added to the chain.

Multiple Nonces. It is possible that two or more miners may find valid nonces that compute the expected hash; miners M_i and M_j may both discover a valid nonce in their respective slices. In such a case, we ensure that all the miners select the same nonce. This is trivially

```
    Miner-role (running at the miner M with identifier $i$) :
1:  event M wants to fetch next committed batches do
2:      if $k - 1$ was sequence of the last fetched transaction (or batch) from replicas then
3:          Ask replicas in $\mathcal{R}$ to send their parts for next $\sigma$ committed batches starting from $k$.

4:  event M receives $\mathbf{f}_{\mathcal{R}} + 1$ parts for $k$-th committed batch do
5:      Regenerate $m'$ using IDA recovery scheme $(2\mathbf{f}_{\mathcal{M}} + 1, \mathbf{f}_{\mathcal{M}} + 1)$ on $m'$.
6:      Run function PenaltyCheck($m'$).
7:      Run function NonceCheck($m'$).
8:      Run function SliceCheck($m'$).

9:  function PenaltyCheck (message: $m'$)
10:     if $m'$ includes committed set of miners ($\mathcal{M}_{mal}$) to be penalized then
11:         Calculate penalty based on shift round $\mathbf{r}$.
12:         Deduct tokens from the accounts of miners in $\mathcal{M}_{mal}$.

13: function NonceCheck (message: $m'$)
14:     if $m'$ includes a committed nonce $\eta$ for block $b$ then
15:         if $\eta$ is valid then
16:             Add $\eta$ to block $b$ and terminate PoC mining for $b$.
17:             if Shift round $\mathbf{r} > \mathbf{0}$ then
18:                 $\mathcal{S}_i$ := Slice which contains $\eta$.
19:                 $\mathcal{M}_{mal} = \{\text{M}_i\}$, where $i = (j + l - 1) \bmod \mathbf{n}_{\mathcal{M}}$, $l \in [1, \mathbf{r}]$
20:                 Broadcast $\langle \text{PENALTY}(\mathcal{M}_{mal}, b, \mathbf{r}) \rangle_{\text{M}}$ to all miners.
21:             Add rewards to accounts of all miners.
22:             Run function NewMine().
23:         else
24:             Add miner who sent this invalid nonce to $\mathcal{M}_{mal}$.
25:             Broadcast $\langle \text{PENALTY}(\mathcal{M}_{mal}, b, \mathbf{r}) \rangle_{\text{M}}$ to all miners.

26: function SliceCheck (message: $m'$)
27:     if $m'$ includes a committed certificate $\mathfrak{C}$ for $b$-th block then
28:         if Shift round $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$ then
29:             Terminate PoC mining for $b$-th block.
30:             Run function NewMine().
31:         else
32:             $\mathbf{r}++$.
33:             Reset timer $\delta$.
34:             if Last mined slice $\mathcal{S}_i$ then
35:                 $i = (i + 1) \bmod \mathbf{n}_{\mathcal{M}}$.
36:                 Run function FindNonce($\mathcal{S}_i$).

37: function NewMine ()
38:     if $k - 1$ was the sequence of last committed batch in the previously mined block $b - 1$ then
39:         if M has $\sigma$ committed batches starting from sequence $k$ then
40:             Create block $b$ such that it includes these $\sigma$ batches.
41:             If no nonce found for block $b - 1$, then create a merged block.
42:             Set shift round $\mathbf{r} = \mathbf{0}$ and start a timer $\delta$.
43:             $\mathcal{S}_i$ := Initial slice assigned to M.
44:             Run function FindNonce($\mathcal{S}_i$).

45: function FindNonce (slice: $\mathcal{S}_i$)
46:     Miner M tried to find a valid nonce $\eta$ in slice $\mathcal{S}_i$.
47:     if M finds $\eta$ then
48:         M broadcast $\langle \text{NONCEFIND}(b, \eta) \rangle_{\text{M}}$ to all miners.

49: event Timer $\delta$ expires while finding nonce for block $b$ do
50:     M sends $\langle \text{SHIFT}(b, \mathbf{r}) \rangle_{\text{M}}$ to all miners.

51: event M receives $\langle \text{NONCEFIND}(b, \eta) \rangle_{\text{M}}$ from a miners do
52:     Verify the nonce and stop the Timer $\delta$.
53:     M sends $\langle \text{NONCEFIND}(b, \eta) \rangle_{\text{M}}$ to all replicas in $\mathcal{R}$.

54: event M receives $\langle \text{SHIFT}(b, \mathbf{r}) \rangle_{\text{M}}$ from $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners do
55:     Create the certificate $\mathfrak{C}$ that includes $\mathbf{f}_{\mathcal{M}} + 1$ SHIFT messages.
56:     M sends $\langle \text{SHIFT}(b, \mathbf{r}, \mathfrak{C}) \rangle_{\text{M}}$ to all replicas in $\mathcal{R}$.

57: event M receives $\langle \text{PENALTY}(\mathcal{M}_{mal}, b, \mathbf{r}) \rangle_{\text{M}}$ from $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners do
58:     Create the certificate $\mathfrak{C}$ that includes $\mathbf{f}_{\mathcal{M}} + 1$ PENALTY messages.
59:     M sends $\langle \text{PENALTY}(b, \mathbf{r}, \mathfrak{C}) \rangle_{\text{M}}$ to all replicas in $\mathcal{R}$.
```

Figure 6: HybridChain miner protocol.

achieved in our HybridChain system as miners only receive the committed nonce transaction from Pbft replicas. Specifically, the primary P will choose a nonce and initiate consensus on that nonce; other nonces will be ignored. Note: only one nonce per block will get successfully committed by $2\mathbf{f}_{\mathcal{R}} + 1$ replicas.

4.5 Rewards

As PoW-based systems need to incentivize their miners for their computational resources, we also need to reward the PoC miners. However, in PoW systems, miners compete with each other to propose the next block. As a result, only the first miner to propose the next block receives the rewards while resources spent by other miners go to waste without any compensation. In contrast, our PoC protocol avoids this resource wastage by requiring miners to collaborate. Further, PoC ensures that each honest miner gets rewarded for its efforts; rewards are distributed among miners.

If $\diamond$ is the reward for a miner to find a valid nonce in PoW protocol, then in our PoC protocol, each i-th miner M_i receives a reward $\diamond_i$ proportional to the size of its slice $\mathcal{S}_i$.

$$\diamond_i = \frac{|\mathcal{S}_i|}{|\mathcal{S}|} * \diamond$$

Like PoW, in PoC, at the time of block creation, each miner in $\mathcal{M}$ adds $\mathbf{n}_{\mathcal{M}}$ transactions to the block (includes them in the Merkle root). These $\mathbf{n}_{\mathcal{M}}$ transactions help to deterministically divide the reward among the miners; i-th transaction adds the reward $\diamond_i$ to the account of miner M_i. Hence, when the miner M_i has access to the valid nonce, its block already includes all the transaction. Including the reward transactions in the block does not imply their execution. Once a miner has access to the valid nonce, it executes the $\mathbf{n}_{\mathcal{M}}$ transactions to update all accounts (Figure 6, Line 21).

Reward Economics. In Section 3.1, we briefly discussed the source of miner rewards. Specifically, miner rewards are composed of transaction fees and winner's reward. Existing blockchain applications like Bitcoin [54] advocate the generation of a token (cryptocurrency) that acts as the winner's reward. We skip diving into the crypto-economics of token generation. However, the general idea behind this token is that it is a monetary compensation for the computational resources spent by the winning miner. Hence, we focus on the transaction fees, which are paid by each client that wants its transactions to be processed.[5]

5 MALICIOUS ATTACKS

Like any MFT system, our HybridChain system also aims to thwart attacks from malicious miners and replicas. Following is the main list of possible malicious attacks on our system:

(A1) Primary prevents consensus on a client transaction.
(A2) Primary equivocates transactions or nonces.
(A3) Miner does not participate in the PoC protocol.
(A4) Miner sends invalid nonce to Pbft replicas.
(A5) Long-range attacks.

To resolve Attacks (A1) and (A2), we employ the *view change* protocol provided by the Pbft protocol [13]. To resolve Attack (A3), we use our novel *slice shifting* protocol and prevent Attack (A4) by severely penalizing the malicious miner. We also show that HybridChain implicitly guards against Attack (A5). To prevent flooding attacks, we follow prior works [15, 16] and assume that the replicas and miners use one-to-one virtual communication channels, which can be disconnected.

5.1 Slice Shifting

The task of PoC miners is to reliably append the transactions committed by Pbft replicas to the ledger. As each miner is trying to find a valid nonce in its slice, an honest miner cannot make progress

[5] Systems like Ethereum [76] set the maximum allowed transaction fees.

until it either finds a valid nonce or receives the valid nonce from Pbft replicas. Such a case may arise due to the following reasons:

(1) A malicious miner may choose not to participate in the PoC protocol, and coincidently, it may be assigned the slice containing a valid nonce. Even if a malicious miner finds the valid nonce, it may not broadcast that nonce to other miners.

(2) Although a significantly low probability, it is possible that none of the miners find a valid nonce. This may be possible if no value in any slice yields the expected hash.

As these cases restrict PoC miners from making progress, we provide novel solutions to alleviate these situations.

Timer Initialization. Prior to searching the nonce for the k-th block, we require each miner M to set a timer δ (Figure 6, Line 42). If the miner M either discovers a valid nonce or receives a valid k-th block from another miner, it stops the timer δ for the k-th block.

Malicious Miner. If M's timer δ expires and it does not have access to a valid nonce, it initiates our *slice shifting* protocol (Figure 6, Line 49). The slice shifting protocol runs for at most $\mathbf{f}_{\mathcal{M}}$ rounds and aims to resolve Attack (A3) by a malicious miner. It does so by switching the slices assigned to different miners. A round of slice shifting only takes place when at least $\mathbf{f}_{\mathcal{M}} + 1$ miners request to do so. When a miner M_i timeouts, it creates a message 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 Pbft to help reach a consensus on slice shifting (Figure 6, Line 54). To do so, each miner creates a certificate $\mathfrak{C}$ that includes $\mathbf{f}_{\mathcal{M}} + 1$ Shift messages and broadcasts this certificate to a Pbft replica.

Once P receives $\mathfrak{C}$ signed by $\mathbf{f}_{\mathcal{M}} + 1$ miners, it initiates consensus on $\mathfrak{C}$. *Note:* consensus on $\mathfrak{C}$ is like consensus on any transaction where $\mathfrak{C}$ acts as the transactional data. PoC miners will fetch this committed transaction when they run the IDA algorithm to fetch the next committed batch of transactions. Once a miner receives the committed certificate $\mathfrak{C}$ from Pbft replicas, it assumes it is time to run the next round $\mathbf{r} + \mathbf{1}$ of the slice shifting protocol (Figure 6, Lines 32-36). Following this, M updates its slice $\mathcal{S}_i$ to slice $\mathcal{S}_j$, where $j = (i+1) \mod \mathbf{n}_{\mathcal{M}}$. Next, M restarts the timer δ for the k-th block and initiates nonce discovery using its new slice $\mathcal{S}_j$.

If M has access to a valid nonce, prior to the expiry of its timer δ, it follows the steps in Section 4.4. Otherwise, M increases $\mathbf{r}$ and re-starts the slice shifting protocol. If M is unable to receive a valid nonce due to malicious miners, it will receive the nonce within $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$ rounds (with high probability). This is the case because at most $\mathbf{f}_{\mathcal{M}}$ miners may act maliciously.

No nonce – Merge. It is possible that even after $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$ rounds, honest miners do not have access to a valid nonce. Such a case takes place if no value in any slice can yield the expected hash. To resolve this case, we require the miners to terminate their search for the nonce and initiate the merge process. The aim of merging blocks is to increase the probability of reaching the expected hash.

Specifically, for the k-th block, if a miner M receives a certificate $\mathfrak{C}$ from Pbft replicas, which has shift round $\mathbf{r} = \mathbf{f}_{\mathcal{M}}$, then M concludes that no valid nonce exists for the k-th block (Figure 6, Line 28). Following this, M creates a new block that merges contents of the k-th and $(k+1)$-th blocks (Figure 6, Line 41). This new merged block acts as the k-th block and miners attempt to find the nonce for this new block. The merged block includes a Merkle root, which is the hash of all the transactions in k and $(k+1)$-th blocks.

Penalty for Slice Shifting. Frequent slice shifting due to malicious miners will be detrimental to the performance of our PoC protocol; it forces honest miners to do more work and wastes their resources. To discourage malicious attacks, we heavily *penalize* malicious miners. Specifically, 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 find the valid nonce.

In our HybridChain system, identifying malicious miners responsible for $\mathbf{r}$ shifts is a trivial task for honest miners. When a miner M receives a valid nonce for the k-th block in shift round $\mathbf{r}$ ($1 \le \mathbf{r} \le \mathbf{f}_{\mathcal{M}}$), it identifies malicious miners on the basis of the slice containing the valid nonce. Let $\mathcal{S}_j$ be the slice, then:

$$\mathcal{M}_{mal} = \{\text{M}_i\}, \text{ where } i = (j + l - 1) \bmod \mathbf{n}_{\mathcal{M}}, \ l \in [1, \mathbf{r}]$$

Here $\mathcal{M}_{mal}$ is the set of malicious miners who caused $\mathbf{r}$ rounds of slice shifting. To penalize these malicious miners, we again invoke the Pbft protocol. Specifically, once a miner M has the knowledge of set $\mathcal{M}_{mal}$, it sends a message Penalty$(\mathcal{M}_{mal}, b, \mathbf{r})$ to all the miners (Figure 6, Lines 17-20). Once M receives Penalty messages from $\mathbf{f}_{\mathcal{M}} + 1$ miners, it creates a certificate that includes these Penalty messages and broadcasts this certificate to each replica in $\mathcal{R}$ (Figure 6, Line 57).

When the Pbft primary receives a Penalty certificate signed by $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners, it creates a new batch of transactions that penalize the malicious miners and initiates Pbft consensus on this batch. Post consensus, each replica executes these transactions, which deducts the stake of each malicious miner (§ 6.1).

Invalid Nonce Attack. A malicious miner can always send an invalid nonce to the Pbft replicas. As Pbft replicas do not verify the correctness of the nonce, such invalid nonces force them to do *wasteful consensus*. However, honest miners can always validate whether a nonce is valid or not. To thwart malicious miners from flooding the Pbft replicas with invalid nonces, we require honest miners to detect an invalid nonce and initiate the process to penalize the responsible miners (Figure 6, Line 23). The process of penalizing the malicious miners is the same as before.

5.2 Malicious Primary – View Change

Pbft replicas play an important role in guaranteeing safety and liveness to our HybridChain system. Specifically, Pbft replicas help in the following tasks:

(1) Ordering client transactions.
(2) Selecting a nonce.
(3) Facilitating slice shifting protocol.
(4) Penalizing malicious miners.

All of these tasks require the primary replica P to ensure successful consensus. However, if P is malicious and prevents any of these tasks, we require honest replicas to replace P. To do so, we follow Pbft's *view change protocol.* Specifically, each consensus is led by the primary of current view v, and the view change protocol moves replicas from view v to view $v + 1$.

Timer Initialization. At the start of each consensus, each replica starts a timer τ. Each replica also initiates a timer, when it receives (i) NonceFind messages, (ii) Shift certificate signed by

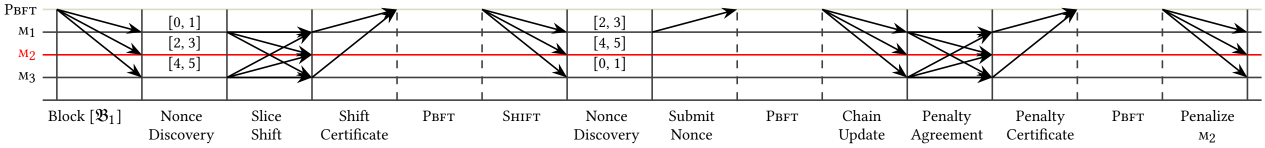

Figure 7: Slice shifting procedure: assume 2 is the valid nonce and is present in the slice of malicious miner M_2. M_2 fails to broadcast 2, which triggers slice shifting. Miners agree on slice shifting with help of PBFT. Once 2 is found, M_2 is penalized.

$\mathbf{f}_{\mathcal{M}} + 1$ distinct miners, (iii) PENALTY certificate signed by $\mathbf{f}_{\mathcal{M}} + 1$ distinct miners, and (iv) an uncommitted client request from a client.

In all of these cases, each replica R waits for P to initiate consensus on respective transactions. If its timer τ timeouts, R requests view change by broadcasting a VIEWCHANGE message to all the replicas. When the replica P′ designated as the primary for view $v+1$ receives VIEWCHANGE message from $2\mathbf{f}_{\mathcal{R}} + 1$ distinct replicas, it starts view $v + 1$ by broadcasting NEWVIEW messages. Following this, P′ tries to complete consensus on pending requests.

5.3 Long Range Attacks

In existing PoS and MFT systems that experience frequent validator churn, it is not unthinkable that a malicious party gains access to the old keys of validators who have exited the system. If this happens, either through bribery or simply mishandling of key material, the adversary can trivially rewrite the complete ledger. This is an extremely precarious situation as even-aged ledgers–5, 10, or 50 years old–are easy to forge. In these systems, a malicious party needs to create new blocks to replace the original blocks and digitally sign them using the keys of honest parties. As stated earlier, performing such an attack on a PoW system though theoretically possible is infeasible in practice due to computational resources needed to mine each block. We illustrate that the same guarantees hold for our HYBRIDCHAIN system.

In our HYBRIDCHAIN system, if malicious parties have access to private-keys of honest parties, they can attempt the following: (1) malicious replicas can commit multiple batches at the same sequence number, (2) malicious miners may change the set $\mathcal{M}_{mal}$.

Malicious replicas may attempt to commit multiple batches at the same sequence number if they want to rewrite the ledger. Assuming that the ledger has $|\mathfrak{B}|$ blocks and starting from the i-th block $\mathfrak{B}_i$, $i < |\mathfrak{B}|$, the malicious parties want to rewrite the ledger. To do so, malicious miners need to create a new block to replace each block in the range i to $|\mathfrak{B}|$. However, malicious miners will not receive help from honest miners to create these blocks as each honest miner has a copy of the ledger. As at least 51% miners are honest (hold at least 51% compute power), so each malicious miner has to solve at least two slices to find a nonce for the replacement block. This is the case because the probability of successfully finding the solution for each block in the initial slice decreases exponentially.

Let, $\frac{1}{2}$ be the probability that malicious miners find the solution for the *first* new block in the first slice. So, the probability that malicious miners find the solution for the b consecutive blocks in the first slice is $(\frac{1}{2})^b$. If $b \geq 7$, the probability is 0.7%. This implies that malicious miners will spend twice the amount of resources/time to mine each new block. This task becomes infeasible if the number of blocks to mine is greater than 7 and in parallel, the ledger at the honest miners may keep growing.

Alternatively, malicious parties may decide to delay honest miners by giving them different batches or no batches. Such an attack will be detected by honest miners and anyone observing the network as there will be a sudden drop in the rate of block production.

The insight here is that the malicious parties can neither rewrite the ledger fast like in PoS and MFT systems, as finding nonce is computationally expensive, nor be fast enough to produce new blocks at the expected latency of the PoC network.

6 DISCUSSION

We now present a discussion on how the miners and replicas join our HYBRIDCHAIN system, how are the accounts of miners managed, and how is the difficulty of PoC mining controlled.

6.1 Staking and Account Database

Like existing PoW and PoS blockchain systems [20, 23, 76], our HYBRIDCHAIN system also rewards and penalizes miners participating in the PoC consensus. To do so, we assume our HYBRIDCHAIN system also includes some **native token**. For the sake of discussion, let us refer to this native token as Ψ. Without delving into token economics, we assume that initially Ψ is proportional to a percentage of dollar currency.

At the genesis of our HYBRIDCHAIN system, any miner that wishes to participate in the PoC mining needs to exchange their dollars for Ψ. Each miner M **stakes** a portion of its tokens Ψ_s to be eligible for mining. For simplicity, we assume that each miner has to stake at least $e \cdot \Psi_s$, where $e > 0$ and e is a system parameter. Staking implies that $e \cdot \Psi_s$ of the miner M are locked prior to its participation in PoC mining; M cannot access its $e \cdot \Psi_s$ till it is participating in PoC mining. For this purpose, we assume each miner interested in participating in our PoC mining has two accounts: *stake account* and *mining account*.

In our HYBRIDCHAIN system, these accounts are managed as a **NoSQL** database. Each account is represented as a *key-value pair*; *key* is the public-key of a miner and *value* stores the number of Ψ_s held by the miner. We assume that each miner has a local copy of this NoSQL database, and it updates the rewards/penalties for each miner as per the protocols described in Sections 4 and 5.

Whenever PoC miners successfully append a new block to the ledger, they are rewarded with Ψ_r. These Ψ_r are added to their mining account. To penalize a malicious miner, Ψ_p are subtracted from its staking and mining accounts.

6.2 Bootstrapping and Discovery

We follow the practices dictated by existing PoS or MFT systems to bootstrap our HYBRIDCHAIN system [20, 23, 63]. Each of these systems assumed that the initial members of their network were honest; these members were vetted before being added to the network. Further, these systems assume that all times at least 2/3-rd members are honest. For example, Ethereum Foundation [76], Avalanche [63], and Cardano [20], control large stakes in their respective networks to prevent malicious attacks.

Prior to any consensus, the PBFT replicas of our HYBRIDCHAIN system help create the first block of the ledger, *genesis block*. This genesis block includes the following information: (1) total number of miners ($\mathbf{n}_{\mathcal{M}}$), (2) total number of replicas ($\mathbf{n}_{\mathcal{M}}$), (3) keys of staking and mining accounts of each miner, (4) value of staking account of each miner, (5) slice identifier for each miner, and (6) identifier for each replica. This genesis block is created and proposed by the primary, and all the replicas participate in PBFT consensus to create to commit this block. Post PBFT consensus, all the PoC miners run the first round of PoC mining to add this block to the ledger.

During the execution of our HybridChain system, replicas and miners may want to join or leave the system. To allow dynamic participation, we make a simplifying assumption that any replica or miner leaves at the boundary of consensus; no miner or replica leaves during an ongoing consensus. This assumption is not surprising as all the MFT systems make a similar assumption [23, 77]. Even PoS systems that use a committee of stakeholders to propose the next set of blocks require the committee members to stay until the next committee is selected [20, 31].

If a miner M wants to join our HybridChain system, it creates a message $\langle \text{JoinMiner}(pk, \Psi)\rangle_{\text{M}}$ and broadcasts this message to all the PBFT replicas. Similarly, a replica R broadcasts a message $\langle \text{JoinReplica}(pk)\rangle_{\text{R}}$. In these messages, pk refers to the public-key; in the case of replicas, public keys must be registered with a public-key certificate authority. The primary replica P, on receiving a join request, creates a transaction that includes this request and initiates PBFT consensus on this transaction. Once the transaction is successfully committed, it is fetched by the miners through IDA. When the miners have added this transaction to the next block, the new node is allowed to join the system. In the case a new miner wants to join the system, once the committed transaction is added to the block, each existing miner creates staking and mining accounts for the new miner in its database.

Similarly, when a miner M (or a replica R) is going to leave the HybridChain system, it will broadcast a message $\langle \text{LeaveMiner}(pk)\rangle_{\text{M}}$ ($\langle \text{LeaveReplica}(pk)\rangle_{\text{R}}$) to all the PBFT replicas. Again, a PBFT consensus is initiated by the primary. Once the leave request is successfully added to the ledger and *all the blocks*, in which the miner (or replica) participated have been added to the ledger, they can leave the system. At this point, it is safe to release the Ψs of the miner.

6.3 Difficulty

Difficulty is a measure of how quickly a block can be generated. It determines the computational resources required to find a valid nonce. Specifically, difficulty states the characteristics of the expected hash–number of leading zeroes. The higher the difficulty, the larger is the amount of time it takes a miner to find the valid nonce.

In existing PoW-based blockchain systems, the difficulty frequently changes (mostly increasing) in an attempt to keep the ledger safe. Specifically, when these systems have lower difficulty, there is a higher probability of two or more miners reaching the valid nonce, which leads to forks in the blockchain. Hence, by keeping high difficulty, these systems aim to reduce the probability of forks. For example, Bitcoin changes its difficulty every 2016 blocks [10, 12].

Unsurprisingly, our PoC has a stable difficulty, which is stated in the genesis block. This stable difficulty helps to substantially save computation resources and energy. *Why does PoC has a stable difficulty?* Every valid nonce is committed by PBFT replicas, which ensures that a malicious miner cannot produce a random nonce. However, as the hardware technology improves with time, some miners may have access to better hardware. In such a case, we would have to increase the difficulty to prevent blocks from being recomputed by malicious miners with better hardware. This decision to increase the difficulty will be taken by PBFT replicas, and all the miners will be informed via committed transactions.

7 CORRECTNESS ARGUMENTS

We argue the security properties of HybridChain presented in Section 2. These properties hold under the standard MFT assumption stated in Section 2. That is, the set of $3\mathbf{f}_{\mathcal{R}} + 1$ replicas contains at most $\mathbf{f}_{\mathcal{R}}$ malicious replicas. Similarly, the set of $2\mathbf{f}_{\mathcal{M}} + 1$ miners contains at most $\mathbf{f}_{\mathcal{M}}$ malicious miners.

7.1 Safety Argument

HybridChain is composed of two subsystems: a MFT consensus and PoC protocol. We argue the safety of HybridChain by relying on the agreement property of the MFT consensus sub-system.

Theorem 7.1 (Safety). *No two conflicting blocks are settled by the PoC protocol. That is, two correct miners m and m' do not settle different blocks $b \neq b'$ with the same sequence number k.*

Proof. Let's assume a correct miner m settles a block b with sequence k and another correct miner m' settles a block b' ($b \neq b'$) with the same sequence k. Correct miners settle a block b with sequence number k only if their MFT consensus sub-system committed b with sequence k. This implies that the MFT consensus sub-system of miner m committed b with sequence number k, and that the MFT consensus sub-system of miner m' committed b' with sequence number k. This implies that the MFT consensus sub-system of two correct miners sequenced different blocks at the same sequence number. This is however a direct contradiction of the agreement property of the MFT consensus sub-system. □

7.2 Liveness Argument

Lemma 7.2 (Commit Availability). *A correct miner eventually obtains the k-th message m committed by a correct replica.*

Proof. We argue this lemma by induction over the serialized reconstruction of committed messages. Assuming a history of $k - 1$ committed messages for which this property holds we consider the k-th committed message m. When a correct replica commits message m it calls the function **IDA-split** (Line 19 of Figure 3) to split m into $2\mathbf{f}_{\mathcal{M}} + 1$ parts. It then makes its part available to the miners, whenever they request for the same. It is thus guaranteed that $2\mathbf{f}_{\mathcal{R}} + 1$ correct replicas eventually hold their part of m. A correct miner can then query any $\mathbf{f}_{\mathcal{R}} + 1$ correct replica for their parts of m. Finally, the correctness property of the IDA protocol ensures that the correct miner can regenerate m from these parts (Line 4 of Figure 6). The inductive base case involves assuming that all replicas are initialized with a committed genesis ($k = 1$) message, which we can ensure axiomatically. □

Corollary 7.3 (Batch Availability). *A correct miner eventually obtains the k-th batch σ committed by a correct replica.*

Proof. The proof follows from the application of Lemma 7.2 with message $m := \text{Committed}(k, \sigma)$. □

Lemma 7.4 (Nonce Search). *The first time a correct miner obtains the k-th committed batch σ, it searches for a valid nonce η in slice $\mathcal{S}_i$.*

Proof. Upon receiving the k-th committed batch σ for the first time, correct miners call the function **PenaltyCheck** (Line 6 of Figure 6) that eventually returns (i.e., it does not contain any loop). They then call the function **NewMine** either through function **NonceCheck** (Line 7 of Figure 6) or **SliceCheck** (Line 8 of Figure 6). The function **NewMine** (Line 37 of Figure 6) checks that σ is the k-th committed batch (which is the case by definition), aggregates it into a block b (Line 40) and calls **FindNonce** (Line 45) to search for a nonce in slice $\mathcal{S}_i$. □

Lemma 7.5 (Shift Liveness). *If a correct miner does not find a valid nonce η in slice $\mathcal{S}_i$ to settle block b within time δ, another miner eventually tries it.*

Proof. When timer δ expires correct replicas submit a message SignMessage to at least one correct replica (Line 49 of Figure 6). The liveness property of the mft consensus sub-system ensures that these messages are eventually committed, and Lemma 7.2 ensures that correct miners are eventually notified of the commit. When $\mathbf{f}_{\mathcal{M}} + 1$ miners timeout, they create a shift certificate (Line 56 of Figure 6). This certificate is then used to reset the timer (Line 32 of Figure 6) and restart the nonce finding process in slice $\mathcal{S}_j$ (with $j = (i+1) \mod \mathbf{n}_{\mathcal{M}}$ (Line 36 of Figure 6). As a result, correct miners keep shifting and searching for each other nonces until they are all found. □

Lemma 7.6 (Block Settlement). *All correct miners settle block b if a valid nonce η for block b exists.*

Proof. When a correct miner m finds a nonce it submits it to the mft consensus sub-system (Line 48 of Figure 6). The liveness property of the mft consensus sub-system ensures that this nonce is eventually committed by all correct validators. Lemma 7.2 Then ensures that all correct miners can obtain the corresponding certificate. We conclude the proof by noting that if the certificate contains a valid nonce η, correct miners settle b (Line 16 of Figure 6). □

Theorem 7.7 (HybridChain Liveness). *Any valid transaction $\langle T \rangle_c$ of a correct client c is eventually settled.*

Proof. The correct client c disseminates its transaction $\langle T \rangle_c$ to at least one correct replica. The liveness property of the mft consensus sub-system ensures that the transaction $\langle T \rangle_c$ is eventually committed as part of a batch σ by at least $2\mathbf{f}_{\mathcal{R}} + 1$ correct replicas. Corollary 7.3 ensures that miners eventually obtain the committed batch σ (Line 4 of Figure 6). Lemma 7.4 then ensures that the miner searches for a valid nonce η to settle σ as part of a block b. Finally, an honest miner can find a nonce in its slice $\mathcal{S}_i$ within time δ with non-zero probability. If it doesn't, Lemma 7.5 ensures that other correct miners will try it again until they succeed. As a result, correct miners eventually find a nonce η for block b in their slice within time δ. Lemma 7.6 then ensures that all correct miners use η to settle block b and thus the transaction $\langle T \rangle_c$ it contains. □

Incentive-Compatibility Argument

Lemma 7.8 (Penaly Certificate). *There cannot be a penalty certificate $\langle \textit{Penalty}(b, \mathbf{r}, \mathfrak{C}) \rangle_M$ unless the timer δ of at least 1 correct miners expires.*

Proof. Assume there exists a penalty certificate $\langle \text{Penalty}(b, \mathbf{r}, \mathfrak{C}) \rangle_M$ while none of the timers of the correct miners expired. Correct miners do not sign penalty certificates when their timer is not expired. Since penalty certificates are composed of $\mathbf{f}_{\mathcal{M}} + 1$ signatures, it follows the system contains $\mathbf{f}_{\mathcal{M}} + 1 > \mathbf{f}_{\mathcal{M}}$ corrupt miners, hence a contradiction. □

Theorem 7.9 (HybridChain Incentive-Compatibility). *No correct miner receives a penalty if (i) it can find a nonce η within time δ and (ii) the network is experiencing a period of synchrony.*

Proof. Let's assume a correct miner m receives a penalty based on shift round r (Line 11 of Figure 6). This implies the existence of a penalty certificate (Line 9 of Figure 6) including miner m in its list of miners to penalize. Lemma 7.8 states that this certificate can only exist if the timer δ of at least 1 correct miners expires. However, since the $\mathbf{f}_{\mathcal{M}} + 1$ miners are correct, they only try to penalize m if they did not receive its nonce before δ. This implies that either miner m did not find its nonce before δ (which is a direct contradiction of assumption (i)), or that its nonce did not reach the $\mathbf{f}_{\mathcal{M}} + 1$ correct miners before their timer expire (which is a direct contradiction of assumption (ii)). □

8 EVALUATION

We implement our HybridChain system on top of our high-throughput yielding, open-sourced ResilientDB fabric [35, 37, 38]. Like ResilientDB, HybridChain is written in C++. Further, ResilientDB provides access to Pbft consensus, so we only had to implement our PoC protocol. In the rest of this section, we refer to ResilientDB's Pbft protocol as **R-Pbft**. The simplicity of our PoC implementation is evident from the fact that it has 2,400 LOC, while ResilientDB has 32,215 LOC. Our evaluation aims to answer the following:

(1) Batching and scaling of R-Pbft? (§8.3-§8.4)
(2) Batching and scaling of PoC? (§8.5-§8.6)
(3) Failures in PoC? (§8.7)
(4) HybridChain versus other blockchains? (§8.8)
(5) Energy consumed by different blockchains? (§8.9)

8.1 Baselines. We evaluate our HybridChain system against *five* state-of-the-art blockchain systems: DiemBFT [23], Hyperledger Fabric [3] (abbreviated as Hyperledger in rest of this section), Avalanche [63], Ethereum [76], and Cardano [20]. We choose to compare against these systems: (1) Ethereum, Avalanche and Cardano are among the top ten cryptocurrencies by market capital; these permissionless blockchain systems hold a combined market capital of 185 billion dollars as of writing this paper. (2) Ethereum version 1.1 allows us to compare against a PoW-based system, while Avalanche and Cardano permit comparison against PoS-based system. (3) Hyperledger and DiemBFT are two popular permissioned blockchain systems using raft [58] and HotStuff [77] consensus algorithms, respectively.

8.2 Setup. We run experiments on Oracle Cloud Infra. (OCI); each replica/miner/node uses *VM.Standard2.8* architecture (16 cores, 8.2 Gbps bandwidth, 120 GB Memory). For each baseline, we present their peak throughput and least latency on a variety of cluster sizes: 16, 32, 64, and 120 nodes (OCI limited us from reaching 128 nodes).

Unless explicitly stated, we set the number of replicas and miners for HybridChain system to 120. We require R-Pbft replicas to run consensus on a batch 100 transactions. Each experiment we run for 30 minutes where first 20 min are for warmup and results are collected over last 10 minutes. We average results over *three* runs to remove noise. For HybridChain, we require clients to sign their messages using *ED25519*-based DS while replicas use *CMAC*. In all the systems, clients send the next transaction after receiving confirmation that their previous block has been committed.

We serve *YCSB* [17, 24] transactions from Blockbench [24] framework to R-Pbft replicas. These transactions are key-value store operations that access a database of 600 k records.

Note: In PoW-based systems, difficulty (D) is expressed as the number of leading zeroes in expected hash [54, 76]. Larger the difficulty, greater the amount of resources needed to find a nonce. As Bitcoin started with difficulty $D = 8$, so in this work, we experiment $D = 8$ and $D = 9$. With our available compute power, larger values of D requires 20+ hours to find a valid nonce, which made the experiment infeasible. Like Bitcoin, we set the number of nonce bits to 42, which leads to a total solution space of 2^{42}. In all our experiments, we reach the peak network bandwidth for R-Pbft, which implies that the network can no longer send any more messages.

Avalanche presents a leaderless protocol called Snow to achieve mft consensus [63, 64]. For our experiments, we employ Avalanche's *C-Chain* design as it exposes a web API to access the blockchain. For Avalanche experiments, we could deploy only 1 k clients, which write to 10 default accounts. Clients send their transactions to proposers, which verify the transaction and propose the next block. Avalanche requires each block to have a size of 8 Mb.

Hyperledger is a permissioned blockchain fabric. We run the latest version, `release-2.1`, where it only supports Raft [58] consensus. In Hyperledger, clients submit their transactions as simple smart-contract, which update key-value pairs, to validators. These validators verify the transaction and reply to the client. Once the client has a sufficient number of verifications, it asks the orderers to run Raft. For our experiments, we deploy an equal number of validators and orderers and set up 100 clients, which continuously submit contracts with different key-value pairs.

Cardano employs PoS consensus to reach mft agreement. For our experiments, we use Cardano's Allegra Era version. We deploy 100 clients, which create smart contracts and send them to a proposer. Cardano randomly selects the proposer with the highest stake to propose the next block to be added to the chain.

Ethereum `version1.1` makes uses of the PoW protocol. When the cluster size is less than 64, each miner is connected to other miners; for 64 and above, Ethereum suggests connecting each miner to only 50 other miners. For Ethereum, we set the minimum difficulty at 131072.

DiemBFT optimizes the HotStuff [77] protocol. Like Pbft, HotStuff also has the *Preprepare, prepare*, and *commit* phases but it requires additional two phases as it switches primary at the end of each consensus. We deploy the research implementation of DiemBFT [72] as it includes all the recent optimizations, and run both with or without mempool versions. Mempool optimization removes the task of disseminating batches from the primary [18].

8.3 Impact of Batching on R-Pbft. First, we illustrate the effect of batching transactions on R-Pbft; in Figures 8(a) and (b), we vary the batch size from 1 to 1 k. We observe that R-Pbft hits the peak throughput when batch size is 150. Beyond this, we observe saturation of peak throughput, while there is a significant increase in latency. This is the case because the queues at replicas are all full and can no longer process any newer requests, which increases the wait time for client requests.

8.4 Impact of Scaling on R-Pbft. Next, in Figures 8(c) and (d), we increase the number of replicas from 16 to 120 and compare the scalability of R-Pbft against DiemBFT's *linear* consensus. As expected, 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.

In the case of DiemBFT, the no-mempool version has much lower throughput as the implementation bottlenecks at the primary replica, which is forced to broadcast all requests.[6] DiemBFT's mempool version requires replicas to wait for 100 ms or to receive 500 Kb of transactions (whichever comes first) before propagating the batch. As a result, DiemBFT's mempool variant trades latency for higher throughput. For R-Pbft, we observe a linear decrease in throughput when scaling from 16 to 120 replicas while sustaining well over 100,000 txn/s with nearly 1 second latency even at 120 replicas, which even outperforms state-of-the-art DiemBFT protocol that is further enhanced with mempool.

8.5 Impact of Blocking on PoC. In Figures 8(e) and (f), we vary the block size for PoC. Within each block, we increase the number of transactions committed by Pbft from 120 k to 280 k. We require the miners to find nonce at $D = 8$ and $D = 9$. As $D = 8$ is easy to solve, PoC miners are able to find the nonce quickly and hit the peak throughput, which is also the maximum throughput achieved by R-Pbft at 120 replicas. For $D = 9$, we observe an increase in throughput with an increase in block size until it hits peak throughput of R-Pbft. This makes us conclude that our PoC protocol does not bottleneck our HybridChain system.

8.6 Impact of Scaling on PoC. Next, in Figures 8(g) and (h), we vary the number of PoC miners from 64 to 120; we test on three different block batch sizes. We observe that by increasing the degree of collaboration (number of miners), there is an improvement in system performance. Further, larger batch sizes help to quickly hit the peak throughput.

8.7 Impact of Failures. We now study two types of failures in our PoC protocol: malicious miner and No nonce (§ 5.1). In both of these experiments (Figures 8(i) and (j)), we issue the failure at the 10-th block, which causes the honest miners to time out and start slice shifting protocol. For the malicious miner experiment, we run PoC among 120 miners, while for the no nonce case, we run PoC among 16 miners. As a result, we also select the highest block size necessary to hit R-Pbft's throughput. Despite this, both

[6] We talked to DiemBFT's authors and they said that the no-mempool version of DiemBFT is unstable and has not been tested at more than 30 replicas.

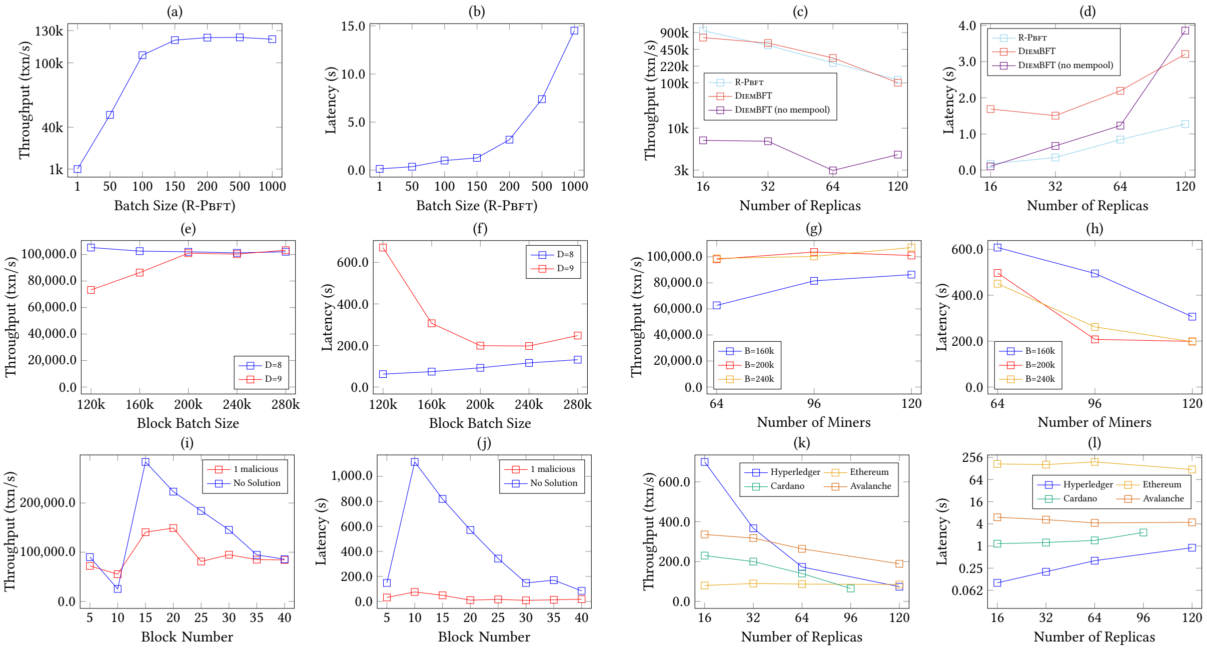

Figure 8: Peak throughput and average latency attained by different components of our HybridChain architecture.

Blockchain	Cost Per 1000 Transactions
R-Pbft	$0.00007
HybridChain	$0.000136
DiemBFT	$0.00019
Hyperledger	$0.08
Avalanche	$0.155
Cardano	$0.239
Ethereum	$14.66

Figure 9: Cost of running different blockchains.

experiments have a similar observable trend. There is a drop in throughput at block size 10 as miners timeout. Due to this, there is a large stack of pending transactions, which allows miners to aggregate more transactions in their block and leads to momentary higher throughputs. However, this period soon fades away and the system reaches stable throughput.

8.8 Performance of Popular Blockchains. We use Figures 8(k) and (l) to analyze the throughput and latency attained by four state-of-the-art blockchain systems. Despite, fine-tuning these systems, we could not reproduce their *claimed* results. Prior works have also recorded such a performance gap [33, 42]. For example, we observe that Avalanche only reaches a peak throughput of 336 txn/s at 16 nodes, while Hyperledger attains 700 txn/s at 16 nodes but drops to 70 txn/s at 120 nodes. Similarly, Cardano attains up to 236 txn/s. However, despite yielding the least throughput, Ethereum's throughput remains stable.

8.9 Energy Cost. We use Figure 9 to illustrate the cost of running different blockchains. To do so, we use the following formula: $(CostPerHour \times Latency)/(Throughput)$. At OCI, each of our deployed node costs $0.2 per hour [59] and we use the throughput and latency measurements at 120 nodes as reference. We use this information to show that running our HybridChain system is more energy efficient than other blockchain systems as it costs the least.

9 RELATED WORK

mft has been studied extensively in the literature [4, 6, 13, 36, 48, 50–52, 56, 65, 66, 70, 73, 79]. A sequence of efforts [11, 13, 34, 46, 49, 53, 68, 69, 71, 77, 78] have been made to reduce the communication cost of the mftprotocols: (1) linearizing mft consensus [32, 35, 77], (2) optimizing for geo-replication [1, 38], and (3) sharding [2, 19, 26, 62]. Nevertheless, all of these protocols face long-range attacks [22].

Alternatively, prior works have focussed on designing PoS protocols that permit the node with the highest stake to propose the next block [20, 31, 45, 47] However, even these protocols suffer from long-range attacks. Long-range attack [22] is a known attack against PoS-based blockchain system that cannot protect from compromised keys of old validators. Existing proposals to protect against such attacks include checkpointing [29], use of key-evolving cryptography [25, 28], or strict chain density statistics. However, recent studies show that these systems are still unsafe. As a result, recent works [8, 74] have proposed anchoring PoS chains on a PoW chain and more concretely Bitcoin [54, 55]. However, these protocols are designed ad-hoc assuming an external chain for security, and additionally, lack rigorous performance evaluations. Combined with the fact that as we show in our experiments classic PoS protocols are orders of magnitude slower than permissioned ones such as Pbft [13] and DiemBFT [9] it seems that the state-of-the-art is either performant or secure against long-range and stake-bleeding attacks. With HybridChain, we show how to get the best of both words, introducing a secure and performant blockchains.

10 CONCLUSIONS

In this paper, we present the design of our HybridChain system, which ensures that each client transaction undergoes a Pbft consensus followed by our novel PoC mining. On the one hand, PoC mining prevents long-range attacks as each block added to the ledger includes a solution to a compute-intensive puzzle; PoC miners do not waste their resources as they collaborate to solve this puzzle. On the other hand, Pbft consensus guarantees low latency response time to clients and fixed difficulty for PoC puzzles. This combination allows our HybridChain system to yield higher throughput and lower latency and energy costs in comparison to state-of-the-art blockchain systems.

ACKNOWLEDGMENTS

This work was supported in part by (1) *Oracle Cloud Credits* and related resources provided by the Oracle for Research program and (2) the *NSF STTR* under *Award Number* 2112345 provided to *Moka Blox LLC*.

REFERENCES

[1] Yair Amir, Claudiu Danilov, Jonathan Kirsch, John Lane, Danny Dolev, Cristina Nita-Rotaru, Josh Olsen, and David Zage. 2006. Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks. In *International Conference on Dependable Systems and Networks (DSN'06)*. 105–114. https://doi.org/10.1109/DSN.2006.63

[2] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. *SharPer: Sharding Permissioned Blockchains Over Network Clusters.* Association for Computing Machinery, 76–88.

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 30:1–30:15. https://doi.org/10.1145/3190508.3190538

[4] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. 2021. Leaderless Consensus. In *41st IEEE International Conference on Distributed Computing Systems*. IEEE, 392–402. https://doi.org/10.1109/ICDCS51616.2021.00045

[5] John Armstrong. 2017. AWS Encryption Keys Compromised in OneLogin Data Breach. https://www.zettaset.com/blog/aws-encryption-keys-compromised-onelogin-data-breach/

[6] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. 2015. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.* 32, 4 (2015), 12:1–12:45. https://doi.org/10.1145/2658994

[7] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. 2020. Winkle: Foiling Long-Range Attacks in Proof-of-Stake Systems. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. Association for Computing Machinery, New York, NY, USA, 189–201. https://doi.org/10.1145/3419614.3423260

[8] Sarah Azouvi and Marko Vukolić. 2022. Pikachu: Securing PoS Blockchains from Long-Range Attacks by Checkpointing into Bitcoin PoW using Taproot. *arXiv preprint arXiv:2208.05408* (2022).

[9] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. 2019. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep* (2019).

[10] Bitcoin Wiki. [n.d.]. Difficulty. https://en.bitcoin.it/wiki/Difficulty

[11] Erik-Oliver Blass and Florian Kerschbaum. 2020. BOREALIS: Building Block for Sealed Bid Auctions on Blockchains. In *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security*. ACM, 558–571. https://doi.org/10.1145/3320269.3384752

[12] Blockchain.com. [n.d.]. Assets Explorer. https://www.blockchain.com/explorer/assets/btc

[13] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461. https://doi.org/10.1145/571637.571640

[14] Junchao Chen, Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. 2022. Power-of-Collaboration: A Sustainable Resilient Ledger Built Democratically. *IEEE Data Eng. Bull.* 45, 2 (2022), 25–36. http://sites.computer.org/debull/A22june/p25.pdf

[15] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright Cluster Services. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, 277–290. https://doi.org/10.1145/1629575.1629602

[16] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 153–168.

[17] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, 143–154. https://doi.org/10.1145/1807128.1807152

[18] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. Association for Computing Machinery, New York, NY, USA, 34–50. https://doi.org/10.1145/3492321.3519594

[19] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 123–140. https://doi.org/10.1145/3299869.3319889

[20] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *Advances in Cryptology – EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 66–98.

[21] Alex de Vries. 2018. Bitcoin's Growing Energy Problem. *Joule* 2, 5 (2018), 801–805. https://doi.org/10.1016/j.joule.2018.04.016

[22] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. 2019. A survey on long-range attacks for proof of stake protocols. *IEEE Access* 7 (2019), 28712–28725.

[23] Diem Association. 2022. Diem BFT. https://www.diem.com/en-us/

[24] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1085–1100. https://doi.org/10.1145/3035918.3064033

[25] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. 2020. Pixel: Multi-signatures for Consensus. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2093–2110.

[26] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: A shared database on blockchains. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1597–1609.

[27] Ittay Eyal and Emin Gün Sirer. 2018. Majority is Not Enough: Bitcoin Mining is Vulnerable. *Commun. ACM* 61, 7 (jun 2018). https://doi.org/10.1145/3212998

[28] Matt Franklin. 2006. A survey of key evolving cryptosystems. *International Journal of Security and Networks* 1, 1-2 (2006), 46–53.

[29] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-bleeding attacks on proof-of-stake blockchains. In *2018 Crypto Valley conference on Blockchain technology (CVCBT)*. IEEE, 85–92.

[30] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-Bleeding Attacks on Proof-of-Stake Blockchains. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 85–92. https://doi.org/10.1109/CVCBT.2018.00015

[31] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. Association for Computing Machinery, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[32] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A Scalable and Decentralized Trust Infrastructure. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. https://doi.org/10.1109/DSN.2019.00063

[33] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. 2022. Diablo-v2: A Benchmark for Blockchain Systems. (2022), 14. http://infoscience.epfl.ch/record/294268

[34] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 568–580.

[35] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2021. Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation. In *Proceedings of the 24th International Conference on Extending Database Technology*.

[36] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. *Fault-Tolerant Distributed Transactions on Blockchain.* Morgan & Claypool Publishers. https://doi.org/10.2200/S01068ED1V01Y202012DTM065

[37] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. 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, 1392–1403. https://doi.org/10.1109/ICDE51399.2021.00124

[38] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proc. VLDB Endow.* 13, 6 (2020), 868–883. https://doi.org/10.14778/3380750.3380757

[39] Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Permissioned Blockchain Through the Looking Glass: Architectural and Implementation Lessons Learned. In *Proceedings of the 40th IEEE International Conference on Distributed Computing Systems*.

[40] Suyash Gupta and Mohammad Sadoghi. 2019. Blockchain Transaction Processing. In *Encyclopedia of Big Data Technologies*. Springer, 1–11. https://doi.org/10.1007/978-3-319-63962-8_333-1

[41] Aaron Holmes. 2021. 533 million Facebook users' phone numbers and personal data have been leaked online. https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4

[42] Vincent iGramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. 2012. Diablo-v2: A Benchmark for Blockchain Systems. https:

//infoscience.epfl.ch/record/294268

[43] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography* (2nd ed.). Chapman and Hall/CRC.

[44] Sunny King and Scott Nadal. 2012. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. https://www.peercoin.net/whitepapers/peercoin-paper.pdf

[45] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19, 1 (2012).

[46] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th usenix security symposium (usenix security 16)*. 279–296.

[47] Markulf Kohlweiss, Varun Madathil, Kartik Nayak, and Alessandra Scafuro. 2021. On the Anonymity Guarantees of Anonymous Proof-of-Stake Protocols. In *42nd IEEE Symposium on Security and Privacy*. IEEE, 1818–1833. https://doi.org/10.1109/SP40001.2021.00107

[48] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2009. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Trans. Comput. Syst.* 27, 4 (2009), 7:1–7:39. https://doi.org/10.1145/1658357.1658358

[49] Lucas Kuhring, Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2021. StreamChain: Building a Low-Latency Permissioned Blockchain For Enterprise Use-Cases. In *2021 IEEE International Conference on Blockchain (Blockchain)*. 130–139. https://doi.org/10.1109/Blockchain53845.2021.00027

[50] Ahmed Lekssays, Giorgia Sirigu, Barbara Carminati, and Elena Ferrari. 2022. MalRec: A Blockchain-Based Malware Recovery Framework for Internet of Things. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '22)*. Association for Computing Machinery, New York, NY, USA, Article 99, 8 pages. https://doi.org/10.1145/3538969.3544446

[51] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. 2016. XFT: Practical Fault Tolerance beyond Crashes. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, USA, 485–500.

[52] Dumitrel Loghin, Tien Tuan Anh Dinh, Aung Maw, Chen Gang, Yong Meng Teo, and Beng Chin Ooi. 2022. Blockchain Goes Green? Part II: Characterizing the Performance and Cost of Blockchains on the Cloud and at the Edge. *arXiv preprint arXiv:2205.06941* (2022).

[53] Mads Frederik Madsen, Mikkel Gaub, Malthe Ettrup Kirkbro, and Søren Debois. 2019. Transforming Byzantine Faults using a Trusted Execution Environment. In *15th European Dependable Computing Conference*. 63–70. https://doi.org/10.1109/EDCC.2019.00022

[54] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[55] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. 2019. Erlay: Efficient Transaction Relay for Bitcoin. Association for Computing Machinery, New York, NY, USA, 817–831. https://doi.org/10.1145/3319535.3354237

[56] Faisal Nawab and Mohammad Sadoghi. 2023. Consensus in Data Management: From Distributed Commit to Blockchain. *Found. Trends Databases* (2023).

[57] Krzysztof Okupski. 2016. Bitcoin Developer Reference. https://github.com/minium/Bitcoin-Spec

[58] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX, 305–320.

[59] Oracle. [n.d.]. Oracle Cloud Infrastructure Blog. https://blogs.oracle.com/cloud-infrastructure/post/vcpu-and-ocpu-pricing-information

[60] Michael O Rabin. 1989. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)* 36, 2 (1989), 335–348.

[61] Sajjad Rahnama, Suyash Gupta, Thamir Qadah, Jelle Hellings, and Mohammad Sadoghi. 2020. Scalable, Resilient and Configurable Permissioned Blockchain Fabric. *Proc. VLDB Endow.* 13, 12 (2020), 2893–2896. https://doi.org/10.14778/3415478.3415502

[62] Sajjad Rahnama, Suyash Gupta, Rohan Sogani, Dhruv Krishnan, and Mohammad Sadoghi. 2022. RingBFT: Resilient Consensus over Sharded Ring Topology. In *Proceedings of the 25th International Conference on Extending Database Technology*. OpenProceedings.org, 2:298–2:311. https://doi.org/10.48786/edbt.2022.17

[63] Team Rocket. 2018. *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*. Technical Report. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV

[64] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and probabilistic leaderless BFT consensus through metastability. *arXiv preprint arXiv:1906.08936* (2019).

[65] Christian Rondanini, Barbara Carminati, Federico Daidone, and Elena Ferrari. 2020. Blockchain-based controlled information sharing in inter-organizational workflows. In *2020 IEEE International Conference on Services Computing (SCC)*. 378–385. https://doi.org/10.1109/SCC49832.2020.00056

[66] Pingcheng Ruan, Tien Tuan Anh Dinh, Qian Lin, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2021. LineageChain: a fine-grained, secure and efficient data provenance system for blockchains. *VLDB J.* 30, 1 (2021), 3–24. https://doi.org/10.1007/s00778-020-00646-1

[67] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, and Gang Chen. 2022. *Blockchains: Decentralized and Verifiable Data Systems*. Springer. https://doi.org/10.1007/978-3-031-13979-6

[68] Vasily A. Sartakov, Stefan Brenner, Sonia Ben Mokhtar, Sara Bouchenak, Gaël Thomas, and Rüdiger Kapitza. 2018. EActors: Fast and flexible trusted computing using SGX. In *Proceedings of the 19th International Middleware Conference*, Paulo Ferreira and Liuba Shrira (Eds.). ACM, 187–200. https://doi.org/10.1145/3274808.3274823

[69] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. *Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX*. Association for Computing Machinery, New York, NY, USA, 955–970.

[70] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. 2021. BFT Protocol Forensics. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1722–1743. https://doi.org/10.1145/3460120.3484566

[71] Man-Kit Sit, Manuel Bravo, and Zsolt István. 2021. 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* (Virtual Event, Italy) *(DEBS '21)*. Association for Computing Machinery, New York, NY, USA, 55–65. https://doi.org/10.1145/3465480.3466922

[72] Alberto Sonnino. 2022. DiemBFT Research. https://github.com/asonnino/hotstuff

[73] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*. Association for Computing Machinery, New York, NY, USA, 17–33. https://doi.org/10.1145/3492321.3519579

[74] Ertem Nusret Tas, David Tse, Fisher Yu, and Sreeram Kannan. 2022. Babylon: Reusing Bitcoin Mining to Enhance Proof-of-Stake Security. *arXiv preprint arXiv:2201.07946* (2022).

[75] Harald Vranken. 2017. Sustainability of bitcoin and blockchains. *Current Opinion in Environmental Sustainability* 28 (2017), 1–9. https://doi.org/10.1016/j.cosust.2017.04.011

[76] Gavin Wood. 2015. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf

[77] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 347–356. https://doi.org/10.1145/3293611.3331591

[78] Rui Yuan, Yubin Xia, Haibo Chen, Binyu Zang, and Jan Xie. 2018. ShadowEth: Private Smart Contract on Public Blockchain. *J. Comput. Sci. Technol.* 33, 3 (2018), 542–556. https://doi.org/10.1007/s11390-018-1839-y

[79] Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi. 2019. GEM^2-Tree: A Gas-Efficient Structure for Authenticated Range Queries in Blockchain. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 842–853. https://doi.org/10.1109/ICDE.2019.00080