Scaling Blockchain using Sharding
A Survey

Seminar Report (CS694)

Master of Technology
in
Computer Science Engineering

by

Debdoot Roy Chowdhury
(23M0765)

Under the Supervision of

Prof. Vinay J. Ribeiro

Department of Computer Science Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
Mumbai - 400076, India

May, 2024

Abstract

The Blockchain, renowned for its decentralized and immutable characteristics, has garnered widespread adoption across finance and industry sectors. However, its limited scalability presents a hurdle to broader implementation, given the complexity of simultaneously maintaining decentralization, security, and scalability. Various remedies have been proposed, encompassing alterations to on-chain data structures, consensus algorithms, and the incorporation of off-chain technologies. Sharding emerges as a pragmatic approach to horizontal scalability, achieved by segmenting the network into multiple shards, thereby alleviating the burden on full nodes. This study conducts a methodical examination of existing blockchain sharding mechanisms and furnishing comprehensive comparisons. It illuminates the characteristics and limitations of current solutions, as well as the theoretical maximum throughput for each method. Moreover, it delves into remaining challenges and outlines prospective research avenues in the domain of blockchain sharding.

Contents

1 Introduction

1.1 Scalability Issue and The Blockchain Trilemma

The challenge of scalability in blockchain arises from its inability to efficiently manage a large volume of transactions without compromising decentralization and security. Established blockchains like Bitcoin [27] and Ethereum [10] encounter limitations as every node is required to process and retain all transactions, resulting in network congestion and elevated fees during peak usage periods. Nevertheless, the pursuit of scalability while upholding decentralization and security remains a persistent concern in blockchain advancement.

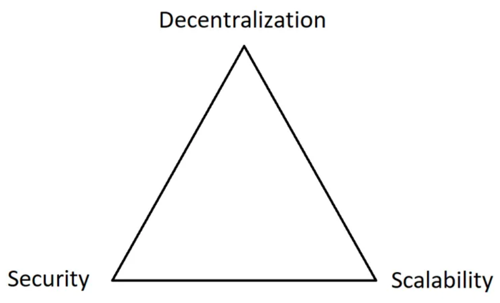


Figure 1: The Trilemma

Coined by Vitalik Buterin, the Blockchain Trilemma [32], also known as the Scalability Trilemma (Figure 1), posits that traditional blockchains can only optimize two out of three properties: scalability, decentralization, and security.

Scalability refers to the blockchain's capacity to handle a greater influx of transactions, while security pertains to safeguarding the data stored on the blockchain against various attacks, including protection against double-spending. Decentralization ensures that control over the network is distributed among multiple entities, preventing domination by a small number of participants.

A network that is both scalable and decentralized necessitates incentivizing a large number of active participants to ensure robust security. However, enhancing scalability and security often entails increasing the operational costs of running a node, which can compromise decentralization. Conversely, prioritizing decentralization and security tends to maintain low node requirements and high costs for potential attacks but can lead to bottlenecks in scalability.

Various solutions, such as sharding, layer-two protocols like the Lightning Network [29], and Sidechains [3], have been proposed to address these challenges. Nonetheless, achieving scalability without sacrificing decentralization and security remains a significant ongoing concern in blockchain development.

1.2 Permissioned and Permissionless Network

Blockchain systems fall into two main categories: permissionless and permissioned [24]. In a permissionless blockchain, any node can freely join or exit the network without requiring authorization. To participate in a permissionless blockchain network like Bitcoin, a node simply needs a pair of public and private keys for message signing and the creation of a receiving address for coins. This open accessibility allows users to join anonymously, potentially enabling adversaries to create multiple identities to bolster their influence in the system—an exploit known as a Sybil attack. To mitigate such threats, permissionless blockchains employ various consensus mechanisms like Proof-of-Work [27] and Proof-of-Stake [21], where nodes contribute computational power or stake amounts to validate transactions. Despite their public and decentralized nature, these consensus protocols lack efficiency in scaling due to their unrestricted participant count.

In contrast, permissioned blockchains regulate participant inclusion, making them suitable for scenarios where entry requires permission and participants must disclose their identities, even if mutual trust isn't a necessity. By restricting access to a predefined group, permissioned blockchains can handle more intricate challenges with greater throughput and lower latency. They find particular utility in enterprises where access to blockchain data requires authorization, as exemplified by Hyperledger Fabric [11] — a permissioned blockchain protocol often adopted by private companies as a foundational technology. In Hyperledger Fabric [11], every node must possess a recognized identity and obtain permission to access network data. In sensitive sectors like finance and healthcare, where data sharing and access control are paramount, leveraging permissioned blockchains ensures efficient regulation of data storage, permissions, and distribution.

1.3 Network State

The effectiveness of a consensus protocol in distributed systems heavily depends on the underlying communication network. Communication networks are typically classified into three categories: strongly synchronous, partially synchronous, and asynchronous [25].

In a strongly synchronous network, a definite bound δ is present, guaranteeing that every message moves from one node to another within a recognized maximum time frame δ.

In a partially synchronous network, there exists a predetermined bound δ on network delay, but with some variations: either δ is consistently enforced but its exact value is unknown, or δ is known but only becomes applicable at an unspecified time.

On the other hand, an asynchronous network does not have a predetermined upper limit on network delay. It is important to acknowledge that communication network models can also differ based on adversarial models, including adversarial network scheduling and oblivious adversarial models.

1.4 Sharding in Blockchain

Sharding, first introduced in database optimization, involves splitting a big database into smaller pieces that are dispersed over several servers [34] [37]. By reducing the load on a centralized server, this decentralization enhances search efficiency and increases the database system's total storage capacity.

The core idea behind sharding technology resembles the "divide and conquer" strategy. When implemented in blockchain, it entails partitioning the network into smaller segments, termed shards, each housing a subset of nodes. Transactions traverse these shards for processing, enabling each node to manage a smaller portion of incoming transactions. This parallel processing capability heightens transaction concurrency and verification, thereby elevating overall network throughput. However, ensuring the decentralization and security of the system during shard partitioning is paramount. Critical considerations include: (a) Establishing consensus within each shard to mitigate common risks like the 51% attack and double-spending. (b) Effectively managing cross-shard transactions while upholding transaction consistency.

2 Fundamentals in Sharding

2.1 Steps in Sharding

The sharding process typically unfolds in four key stages: identity establishment, shard assignment, consensus determination, and shard reconfiguration.

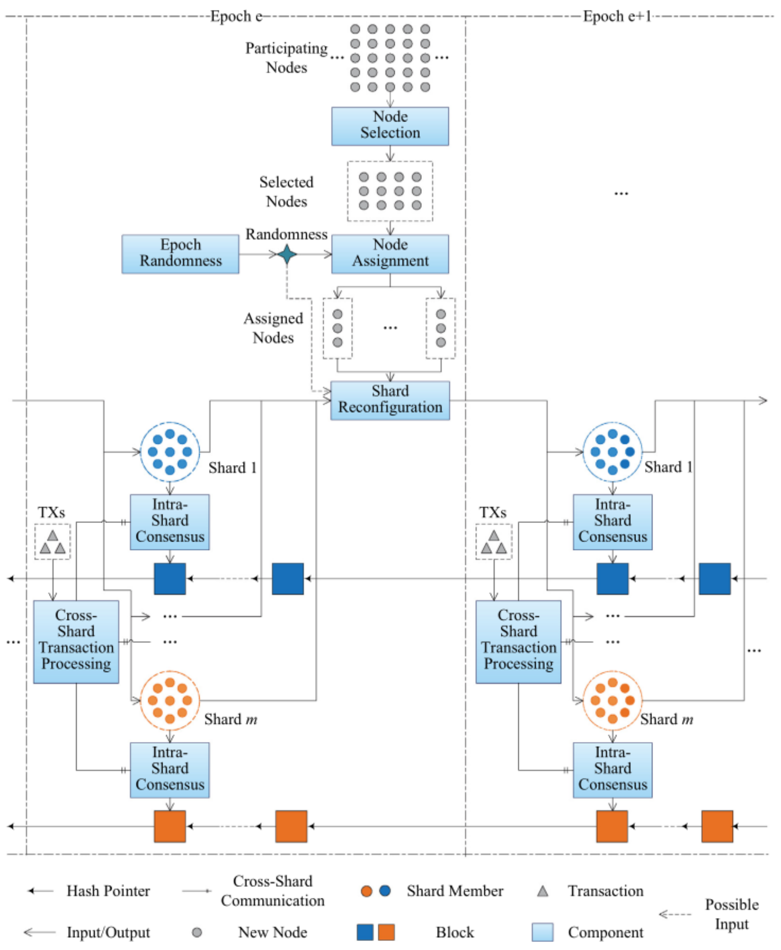


Figure 2: Steps in Sharding

In permissionless blockchains, nodes establish identities like public keys and IP addresses during identity establishment. Following this, in the shard assignment phase, nodes and transactions are randomly allocated across different shards. Within each shard, nodes exchange identities and establish connections without constraints. Intra-shard consensus occurs among nodes within the same shard to reach agreement on transaction sets, while inter-shard consensus involves collaboration among nodes from different shards to achieve global consensus. Sharding systems undergo shard reconfiguration, redistributing nodes among shards after each epoch, to enhance system security.

2.2 Sharding Techniques

1. **Network Sharding** - All nodes in a blockchain network are randomly divided into different shards using a fundamental sharding approach. Two ways are usually used in this method for allocating nodes: functional allocation and non-functional allocation. Nodes are distributed among shards by non-functional allocation, which uses random number generators or their identities.

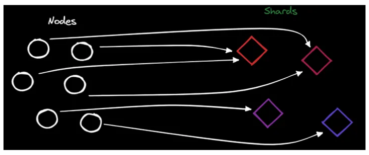


Figure 3: Network Sharding

2. **Transaction Sharding** - By splitting up transactions into distinct shards, each node keeps a copy of the complete blockchain and handles transactions in parallel. This technique is known as transaction sharding. Transactions with shared inputs can be routed to the same shard in order to prevent double-spending. An alternative is to distribute transactions randomly among shards according to their addresses; this will necessitate cross-shard communication in order to synchronize transactions with identical inputs across various shards and avoid double-spending.

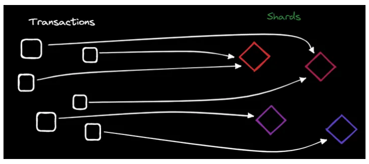


Figure 4: Transaction Sharding

3. **State Sharding** - This method divides the entirety of a blockchain network's state into distinct shards. Unlike transaction sharding, which involves nodes storing solely transaction data, state sharding entails storing a segment of the entire ledger, thereby reducing storage requirements. Preserving the comprehensive global state necessitates additional backups to forestall corruption within shards.

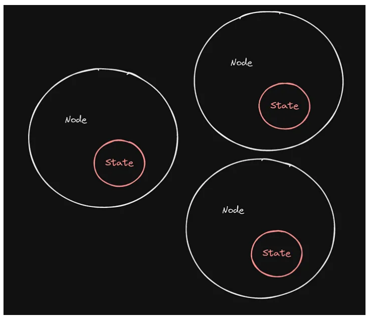


Figure 5: State Sharding

2.3 Challenges in Sharding

The challenges confronted by sharding-based blockchain systems encompass scalability, consensus, performance, and security. These challenges are addressed in various research papers as follows:

1. **Reduction in Security**: Sharding can compromise the security of the blockchain by introducing vulnerabilities such as increased attack surface and potential for collusion among shard validators.

2. **Cross-shard Communication**: Efficient communication between shards is crucial for maintaining consistency and atomicity of cross-shard transactions. However, achieving low-latency and reliable cross-shard communication poses technical challenges.

3. **Effect on Decentralization**: Sharding may impact the decentralization of the blockchain network by concentrating power among a subset of nodes responsible for managing shards. Maintaining a decentralized network while implementing sharding is a delicate balance.

4. **Finite Scalability**: While sharding improves scalability by dividing the network into smaller partitions, there is a limit to the number of shards that can be added before diminishing returns or increased complexity outweigh the benefits.

5. **Imbalanced Workloads on Shards**: Uneven distribution of transactions among shards can lead to imbalanced workloads, causing some shards to become overloaded while others remain underutilized. Balancing transaction distribution across shards is essential for optimal performance.

6. **Dynamic Shard Management**: Adapting to changes in network conditions, such as fluctuations in transaction volume or node availability, requires dynamic shard management mechanisms. Ensuring efficient shard reconfiguration and load balancing in response to changing conditions is a significant challenge.

2.4 Evaluation Metrics

1. **Scalability**: This metric evaluates how effectively the sharding approach can expand the blockchain's capacity to handle a growing number of transactions and users. It involves analyzing the system's ability to maintain performance levels as the network size increases, without compromising on security or decentralization.

2. **Throughput**: Throughput measurement focuses on quantifying the rate at which transactions are processed and added to the blockchain within a given timeframe. It provides insights into the efficiency of the sharding mechanism in terms of transaction processing speed and overall system performance.

3. **Latency**: Latency assessment involves examining the time taken for transactions to be confirmed and included in the blockchain. Low latency is desirable as it ensures quick transaction finality and enhances the user experience by reducing the time required for transaction confirmation.

4. **Security**: Security evaluation entails analyzing the resilience of the sharding solution against various types of attacks, including 51% attacks, double-spending, and Byzantine faults. It examines the integrity and confidentiality of transactions and ensures that the system remains secure even in the presence of malicious actors.

5. **Decentralization**: Decentralization metrics assess the distribution of power and control among nodes and shards within the blockchain network. It examines whether the sharding approach maintains the principles of decentralization, censorship resistance, and equal participation among network participants.

6. **Cross-Shard Communication Overhead**: This metric measures the efficiency of communication between shards and evaluates its impact on overall system performance. It examines the overhead associated with cross-shard transactions and assesses strategies to minimize communication latency and maximize throughput.

7. **Fault Tolerance**: Fault tolerance evaluation focuses on the system's ability to maintain functionality and consensus even in the presence of faulty or malicious nodes. It examines the resilience of the sharding approach to node failures, network partitions, and other disruptive events.

8. **Load Balancing**: Load balancing analysis examines the distribution of workload among shards to ensure optimal resource utilization and prevent performance bottlenecks. It assesses mechanisms for dynamically redistributing workload to maintain consistent performance across shards.

9. **Atomicity of Cross-Shard Transactions**: This metric verifies that transactions involving multiple shards are executed atomically, ensuring consistency and integrity across the blockchain. It examines the mechanisms for coordinating and synchronizing cross-shard transactions to prevent double-spending and maintain transaction atomicity.

10. **Adaptability and Flexibility**: Assessment of adaptability and flexibility involves evaluating the sharding approach's ability to accommodate changes in network conditions, technology advancements, and evolving user requirements. It examines the scalability of the solution and its capacity to integrate new features or enhancements while maintaining compatibility with existing systems.

3 Literature Review

3.1 Elastico

Elastico [26] represents the inaugural secure sharding protocol capable of accommodating Byzantine adversaries. Its core concept involves segmenting the network into smaller committees, each responsible for processing distinct sets of transactions referred to as 'shards'. Elastico [26] operates under the assumption that adversaries control at most 1/4 of the hashing power, with this fraction selected as an arbitrary constant set apart from 1/3 to ensure reasonable parameter values.

The protocol operates in epochs, with each epoch comprising five sequential steps: 1) Identity establishment and Committee Formation 2) Overlay setup, 3) Intra-committee consensus, 4) Final consensus broadcast, and 5) Epoch randomness generation.

3.1.1 Identity Establishment and Committee Formation

Nodes are not allowed to autonomously select which committee to join because it would let enemies to take control of particular shards. A node must first solve a Proof-of-Work challenge in order to prove its identity and join the network. Every node searches for a nonce, in a manner akin to that of Bitcoin miners.

$$\text{ID} = \text{H}(\text{EpochRandomness, IP, Public key, Nonce}) < \text{D},$$

in which D denotes the degree of difficulty, *EpochRandomness* is a random number produced by the consensus committee for each epoch, IP and *Publickey* are node inputs, and $Nonce$ is the riddle the node needs to solve. Elastico [26] then uses the final k bits of the identities (ID) to distribute identities equitably among 2^k committees.

3.1.2 Overlay Setup for Committees

The naive method of broadcasting identities for N nodes results in $O(N^2)$ messages, which lacks scalability. Elastico [26] adopts a hierarchical strategy to address this issue by utilizing directory committees for broadcasting all identities. Initially, the first C identities assume the role of directory servers, which may differ for each node (where C typically denotes the size of each committee in the network). Upon establishing their identity, nodes transmit their ID to the nearest directory server. Once each committee attains C or more nodes, the directory server broadcasts the committee list. Consequently, each directory server gossips the committee list to N nodes, thereby transmitting $O(NC)$ messages in total.

3.1.3 Intra-Committee Consensus

After the committees are established, a pre-existing Byzantine Fault Tolerant (BFT) protocol such as PBFT [12] is used to reach intra-shard consensus within each committee. More than half of the committee members must sign each transaction before the transaction amount is forwarded to the final consensus committee. There are simple ways to guarantee that every committee suggests different shards, including allocating every committee to work in a different shard according to its committee ID.

3.1.4 Final Consensus Broadcast

A final committee is assembled to achieve a conclusive consensus regarding the set of blocks received from the committees. This process involves amalgamating the consensual values from the committees to generate a final agreed-upon outcome. The nodes constituting the final committee comprise all members possessing a fixed s-bit committee ID. Other committees transmit their block headers to the final committee, which proceeds to verify the signatures, execute a Byzantine consensus protocol to ascertain the ultimate result, and broadcast the block to the network.

3.1.5 Epoch Randomness Generation

The final committee employs a distributed commit-and-xor scheme to generate randomness for the subsequent epoch. To mitigate issues related to prediction and agreement concerning a single random number, Elastico [26] permits different random numbers to be chosen by the nodes of the final committee. Each member of the final committee privately selects a random seed λ_i, incorporates Hash(λ_i) into the final block, and broadcasts λ_i alongside the final block. Any node can XOR any set of $2C/3$ of λ_i to derive randomness. Upon sending a Proof-of-Work (PoW) solution, the set of λ_i utilized is also transmitted to enable others to verify the correct construction of randomness.

3.1.6 Limitations and Drawbacks

1. The Proof of Work method used to establish identities for validators and assign them to shards is susceptible to bias since miners have the ability to drop blocks, thus influencing the randomness.

2. The traditional non-scalable PBFT [12] protocol employed in Elastico [26] results in an unacceptably high failure probability, with a total fault tolerance of only 25%. This can be mitigated by enlarging the consensus group, albeit at the expense of significant communication overhead.

3. Cross Shard transactions are not accounted for, assuming that transaction distribution obviates the need for cross-shard communication. The global state is maintained in each shard.

4. Elastico's distributed commit-and-xor technique [26] demonstrates robustness and limited availability. Furthermore, it is not a completely objective randomness generator unless a greater communication overhead is involved.

Despite this scalability and security threat of Elastico [26], it led to the foundation stone for sharding in permissionless blockchain which later improved by two other protocols Omniledger [23] and Rapidchain [38].

3.2 Omniledger

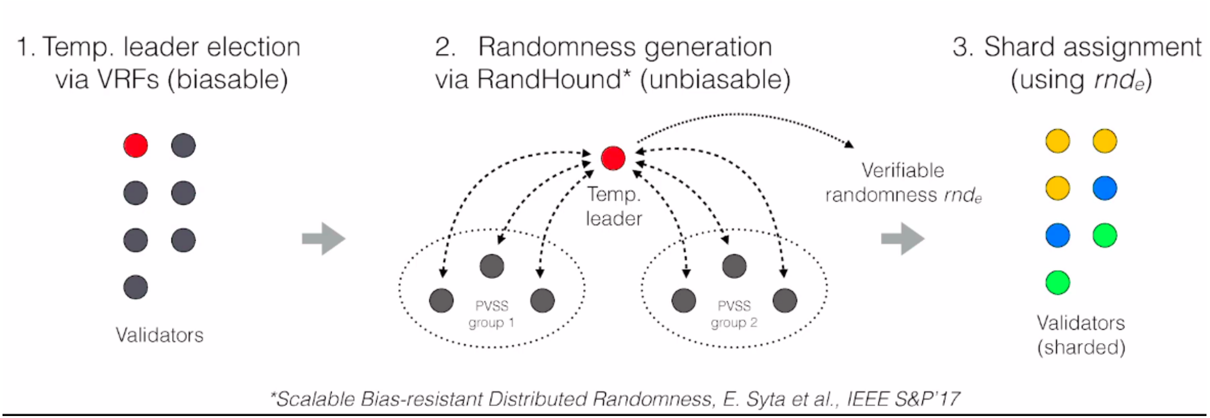
OmniLedger [23] endeavours to establish a secure, permissionless distributed ledger, aiming to enhance scalability and performance compared to Elastico [26]. OmniLedger [23] comprises multiple shards and integrates various techniques such as RandHound [75] and Algorand-based Verifiable Random Function (VRF) [18] to generate unpredictable and unbiased randomness, maintaining a 25% Fault Tolerance (FT) for the reallocation and leader-election processes of each shard. Additionally, OmniLedger [23] adopts a new scalable Byzantine Fault Tolerant (BFT)-based consensus algorithm named ByzCoinX (an enhanced version of ByzCoin [22]) for intra-shard consensus. Furthermore, it introduces a novel two-step atomic commit protocol named Atomix to facilitate cross-shard transactions.

3.2.1 Identity Establishment

A global identity blockchain is used by Omniledger [23], in which validators wishing to join the network must first register. Validators can choose to prove their identities using any proof-of-work or other Sybil-attack-resistant technique, then commit them to the identity blockchain in order to participate in an epoch.

3.2.2 Validator Assignment in Shards

Similar to Elastico [26], nodes are evenly distributed across all shards based on generated randomness in OmniLedger [23]. The source of randomness in OmniLedger [23] is a fusion of a VRF-based lottery [18] and an unbiased randomness protocol, RandHound [33] (Figure 6).



1. Temp. leader election via VRFs (biasable)
Validators
2. Randomness generation via RandHound* (unbiasable)
Temp. leader
PVSS group 1
PVSS group 2
Verifiable randomness rnd_e
3. Shard assignment (using rnd_e)
Validators (sharded)
*Scalable Bias-resistant Distributed Randomness, E. Syta et al., IEEE S&P'17

Figure 6: Shard validator assignment of Omniledger

At the onset of each epoch, a temporary leader is selected using a VRF-based lottery from all validators. This leader executes the RandHound [33] protocol to produce a randomness value rnd_e. This approach mitigates the risk of detrimental randomness stemming from a malicious leader. The resulting randomness (rnd_e) dictates the shard assignment process.

3.2.3 Intra Shard Consensus - ByzCoinX

For intra-shard consensus, Omniledger uses ByzCoinX, an enhanced version of the ByzCoin protocol [22]. Byzantine Fault Tolerance (BFT) and Proof of Work (PoW) algorithms combined in a tree-based structure were originally represented by ByzCoin, a groundbreaking scalable consensus protocol made possible by scalable collective signing (CoSi) [6], [33].

ByzCoinX uses a shallow tree structure with an increasing branching factor and a fixed depth of 3 (see Figure 7). Each group leader oversees a group, producing a subtree

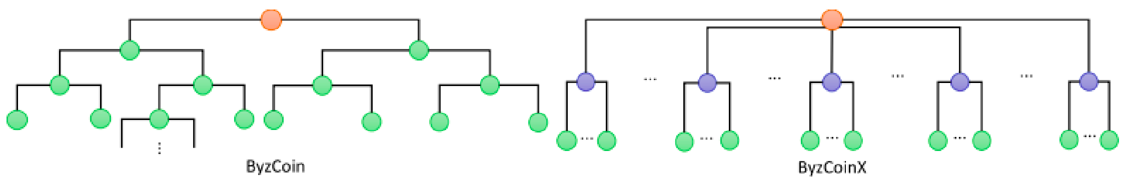


Figure 7: ByzCoin compared to ByzCoinX

with a predefined number of group members based on the size of the shard. Each subtree's leaders obtain signatures from at least two-thirds of their offspring, or leaves, and then forward each group's signatures to the root, or protocol leader. When the root has signatures from at least two thirds of its progeny (group leaders), the decision is considered final. When it comes to shards with hundreds of validators, ByzCoinX performs better than ByzCoin in terms of latency. This is mostly because of the fixed depth, shorter path from leaves to the root, and increased fault tolerance brought about by the growing branching factor. But ByzCoinX has more latency than ByzCoin once the number of validators surpasses a particular threshold. This is a result of ByzCoinX's rising branching factor in certain situations.

3.2.4 Cross-Shard Transaction

OmniLedger [23] presents a groundbreaking atomic protocol named Atomix, designed to uniformly commit cross-shard transactions across all shards. The protocol entails the locking of all input transactions at the input shards before finalizing the output transaction(s) at the output shard(s).

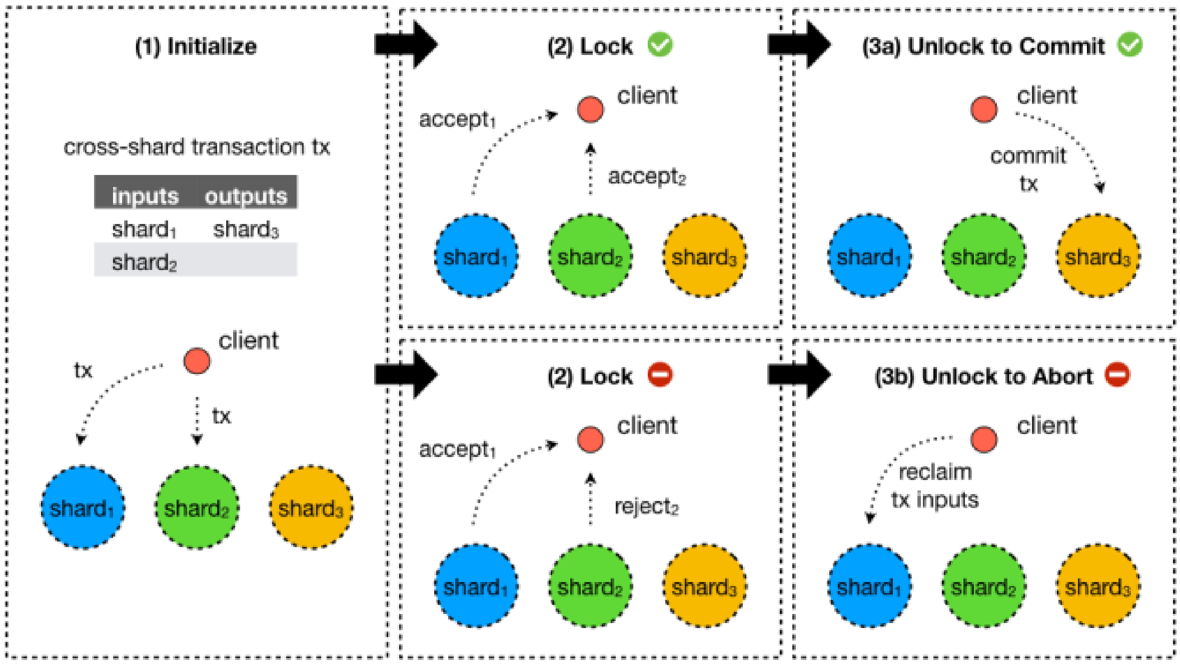


Figure 8: Atomix Protocol of Omniledger

The protocol can be can be divided into 3 stages (Figure 8:

1. **Initialize:** Unspent Transaction Outputs (UTXOs) from two distinct shards, tx_1 from shard S_1 and tx_2 from shard S_2, are used by the user when they initiate a transaction tx. The transaction tx is shared by the client and then propagates to all input shards.

2. **Lock:** Transactions inside each input shard are verified by that shard itself. If the transactions are deemed legitimate, a proof-of-acceptance is distributed and they are recorded as spent in the shard's ledger. On the other hand, a proof-of-rejection is sent out if any transaction is determined to be invalid.

3. **Commit:** The client can propagate unlock-to-commit transactions by chatting to all output shards if each input shard distributes proof-of-acceptance. Every output shard then verifies the transaction and adds it to its ledger. However, the transaction needs to be stopped and cannot be committed if any input shard spreads proof-of-rejection. To recover the money, the client uses rumors to spread an unlock-to-abort message.

3.2.5 Limitations and Drawbacks

1. The utilization of RandHound and VRF for generating epoch randomness is hampered by the dependence on third-party initial randomness pre-defined in the genesis block.

2. The ByzCoinX consensus mechanism improves scalability and lowers the likelihood of failure for intra-consensus within OmniLedger. However, this enhancement comes at the cost of increased transaction latency in large-scale networks.

3. The client-driven protocol is vulnerable to indefinite blocking if the client acts maliciously. For example, suppose the payee, who is coordinating a transaction to transfer funds from a payer's account, simulates an indefinite crash during the lock/unlock protocol. In that case, the payer's funds may become permanently locked.

4. Clients must actively participate in cross-shard transactions, which may present challenges for lower-end devices.

OmniLedger [23] introduces a more secure mechanism for node assignment to shards and proposes an atomic protocol for inter-shard communication, thereby reducing communication overheads relative to Elastico [26]. Challenges associated with the atomic protocol for cross-shard transactions have been addressed in RapidChain [38].

3.3 RapidChain

RapidChain [38] represents the inaugural full-sharding-based permissionless blockchain protocol, boasting Byzantine fault tolerance and necessitating no trusted setup. It asserts sublinear communication, denoted as $o(n)$, where n denotes the number of bits exchanged per transaction, and demonstrates resilience to corruption of up to 1/3 of the nodes. It leverages the Cuckoo Rule to counteract the "Slowly Adaptive Byzantine Adversary".

In contrast to prior methods, each node exclusively stores a fraction of 1/k of the ledger, where k signifies the number of shards. The implementation of the efficient routing mechanism Kademlia for cross-shard transactions reduces latency and diminishes storage overhead. Furthermore, it adheres to a decentralized bootstrapping algorithm that does not presuppose the existence of initial randomness.

3.3.1 Bootstrap

The bootstrap phase occurs solely once at the inception, after which the protocol proceeds in epochs, with each epoch comprising multiple Consensus and Reconfiguration phases. During initialization, the initial nodes elect an election committee, which collectively agrees upon a group of $O(\sqrt{n})$ nodes designated as the root group. The root group then generates and disseminates a sequence of random bits utilized to establish a reference committee (C_r) comprising $O(\log n)$ nodes. C_r assumes responsibility for Epoch Randomness.

At the conclusion of each epoch i, C_r generates a fresh randomness denoted as r_{i+1}, serving as the epoch randomness for the subsequent epoch $i+1$. This randomness is employed to: (1) Create sharding committees, (2) Facilitate nodes in acquiring fresh identities at each epoch, and (3) Reconfigure existing committees.

3.3.2 Consensus

Users transmit their transactions to the network via proxy nodes. These proxy nodes relay the transactions, utilizing the inter-committee routing Kademlia protocol, to their corresponding committee (referred to as the Output committee C_{out}) responsible for processing the transaction. The determination of C_{out} for a transaction is computed based on the hash value of that transaction.

C_{out} selects a local leader using the current epoch randomness, who subsequently dispatches the transaction within a verified block to all members of C_{out}. All members participate in a Synchronous Byzantine Fault Tolerance (BFT) consensus protocol (specifically, Abraham et al.'s Efficient Synchronous Byzantine [1]), which mandates the agreement of at least 1/2 of the nodes for optimal resiliency, to validate the block.

3.3.3 Reconfiguration

The Reconfiguration phase occurs at the conclusion of each epoch and is carried out by the members of C_r. This phase entails the generation of a reconfiguration block aimed at establishing two key pieces of information: (1) A fresh epoch randomness. (2) A new roster of nodes along with their committee affiliations.

C_r generates a fresh randomness denoted as r_{i+1} for the ensuing epoch and disseminates it to all committees. This freshly generated randomness is responsible for redistributing nodes among committees in an unpredictable manner and initiates a puzzle to ascertain the identity of nodes desiring participation in the subsequent epoch. Participants must solve a fresh Proof of Work (PoW) puzzle within 10 minutes to qualify for participation in the next epoch. After 10 minutes, C_r verifies the solution provided by each node and appends the verified identity to the list of participants eligible for the subsequent epoch.

Then, in a reconfiguration block, C_r arranges for an internal committee meeting to decide on and record the identity list in C_r's ledger. Along with the new committee memberships, this reconfiguration block includes r_{i+1}. The management of churn, or changes in C_r membership, is done in the same way as other committees. The epoch i

r_{i+1} produced by the C_r members determines the makeup of the new C_r members for epoch $i + 1$.

3.3.4 Cross-Shard Transaction

The user adopts a passive role in the transaction process, abstaining from active participation or attaching any proof akin to Omniledger. Refer to Figure 9 Instead, the output committee C_{out} solicits the input Unspent Transaction Output (UTXO) to be expended. A transaction $tx = \langle (I_1, I_2), O \rangle$ is transmitted to any proxy node, which subsequently relays it to its designated output committee C_{out}. This transaction comprises two inputs I_1 (from shard S_1) and I_2 (from shard S_2), along with one output O (to shard C_{out}). Notably, I_1 and I_2 originate from distinct shards (S_1 and S_2) other than C_{out}, where O is intended to reside.

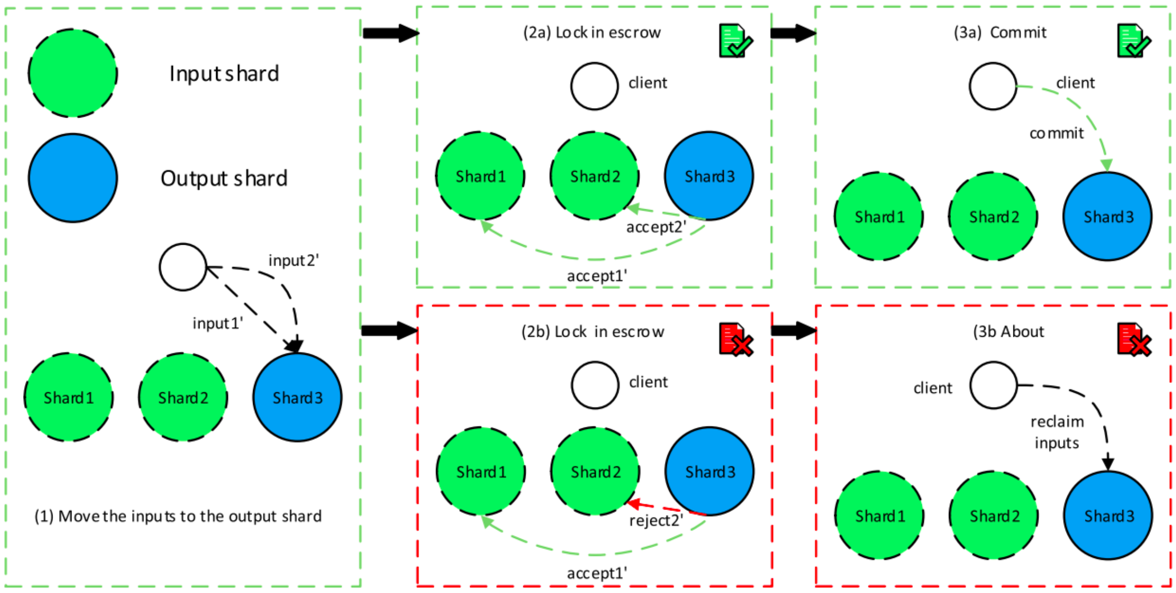


Figure 9: Cross-Shard Transaction in RapidChain

The leader of C_{out} then crafts three new transactions: $tx_1 = \langle I_1, I'_1 \rangle$, $tx_2 = \langle I_2, I'_2 \rangle$, and $tx_3 = \langle (I'_1, I'_2), O \rangle$. Subsequently, tx_1 and tx_2 are dispatched to S_1 and S_2 respectively, effectuating the transfer of I_1 and I_2 to S_3 as I'_1 and I'_2.

Upon successful execution of both tx_1 and tx_2, S_3 incorporates tx_3 into its ledger. Should one of the transactions between tx_1 and tx_2 encounter failure (e.g., tx_1 succeeds while tx_2 fails), Rapidchain preserves atomicity by explicitly instructing the owner of I_1 to utilize I'_1 for subsequent transactions, thereby completing the cross-shard transaction.

To mitigate the overhead resulting from a plethora of cross-shard transactions, an optimization method called Batching Verification Requests is employed. C_{out} consolidates multiple cross-shard UTXOs belonging to the same input committee into batches and dispatches a single UTXO request to the input committee.

3.3.5 Limitations and Drawbacks

1. Rapidchain's inability to maintain isolation is evident in certain scenarios. For instance, consider another transaction tx'_2 in S_2 that spends I_2 and is submitted

around the same time as tx. The shard serializes the transactions, resulting in the success of only one of tx_2 or tx'_2. If isolation were achieved, either tx_2 or tx'_2 would succeed. However, if tx'_2 fails while tx_2 succeeds but tx_1 fails, none of the transactions would be successfully processed.

2. With the increment of shards, nearly all transactions evolve into cross-shard transactions. Batch processing of cross-shard transactions provides partial alleviation.

3. The latency increases by 1.5 times if 10 new nodes join the network for each shard.

4. There is no provision for adaptive-based sharding in RapidChain. Restarting the entire network is imperative to modify the number of shards.

5. RapidChain's methodology proves inadequate for non-UTXO distributed transactions as it violates both atomicity and isolation principles.

3.4 Monoxide

Monoxide [35] introduces a groundbreaking sharding mechanism devoid of the necessity for generating randomness. It employs a novel Chu-Ko-Nu mining technique rooted in the Nakamoto-based Proof-of-Work (PoW) [27] algorithm for its intra-consensus, thereby addressing the issue of computing power dilution post-sharding.

The concept of asynchronous consensus zones, where each zone represents a shard, is adopted by Monoxide [35]. This protocol uses an account/balance transaction model that is similar to a bank account model, departing from the traditional Unspent Transaction Output (UTXO) transaction models. To guarantee transaction atomicity across zones, it presents a future atomicity method that makes use of relay transactions.

3.4.1 Consensus Zones

Through workload division and management among numerous independent, parallel instances, referred to as Consensus Zones, the system grows horizontally. The condition of the entire network is divided among the zones, and each zone is accountable for the portion that it has been assigned. Blocks and transactions are examples of key data structures that are unique to each zone and are replicated and stored only inside those zones. Within each zone, mining competition, chain expansion, and transaction confirmation happen independently and asynchronously.

The number of zones, denoted as n, equals 2^k, where k represents the number of bits used to derive a user's zone index. The zone index of an initiating transaction is determined by the payer's address, while the zone index of a relay transaction is determined by the payee's address.

3.4.2 Chu-Ko-Nu Mining Technique

As first suggested in [20] and explored in [4], merged mining serves as the model for chu-ko-nu mining. In mixed mining, a parent chain and multiple auxiliary chains that all use the same Proof-of-Work (PoW) algorithms share mining power. As a result, auxiliary chains that possess relatively less mining power can gain from the parent chain's overall mining power. Monoxide uses a similar strategy, however it mines across a number of concurrent shards in an unhierarchical manner. Because of the way it is made, attacking a single shard is just as difficult as attacking the system as a whole.

For all suggested blocks coming from different shards, the storage framework is the Merkle Patricia tree (MPT) [10]. Effective mining power can be increased by a factor of k by integrating an MPT root (where k is the number of shards that a given miner mines).

With the restriction of one block per shard, the Chu-ko-nu mining method enables miners in any shard to generate numerous blocks upon successfully solving the hash puzzle. Because of this, miners in each shard have more processing power, making it necessary to attack a single shard with the same amount of computing power as attacking the system as a whole.

The consensus algorithm in Monoxide is expressed as follows:

$$H(\phi||H(x||MPT_M)) \leq \epsilon$$

The hash function in this case is represented by H, the random nonce by ϕ, the block header content by x, the Merkle Patricia tree (MPT) root aggregated by $B_0, B_1, ..., B_{n-1}$ from each involved shard by MPT_M, and the PoW target difficulty by ϵ.

To reach $Pk/n \approx P$, Monoxide needs the majority of miners in each shard to engage in Chu-ko-nu mining ($k \rightarrow n$), where P is the total mining power and n is the number of shards. This requirement is essential since a single shard's security might not always be ensured by distributed mining power. On the other hand, it can lead to an excessive concentration of mining power, which would result in extra overhead.

3.4.3 Cross-Zone Transaction

Asynchronous execution technique called "eventual atomicity" is introduced by Monoxide [35] to reduce the overhead related to state locking or unlocking. An initiation transaction and a relay transaction are the two components that make up a single transaction in this method. Next, the relay transaction is added to the set of outgoing transactions, allowing all consensus groups to attain high throughput without obstructing one another. Eternal atomicity is intended to be lock-free even though it is not instantaneous. In an asynchronous network, the Chu-ko-nu mining technique is specifically used to maximize global throughput.

Examining a cross-shard payment transaction, we see that user A sends user B x tokens in different shards. This transaction consists of an initiation Tx and a relay Tx. First, there is a successful packing of the transaction involving the transfer of x tokens from A to B. The tokens are then transferred via the relay Tx across shards to shard b. Transfers and receipts are additionally represented as withdrawal and deposit operations, respectively, to preserve atomicity. The deposit operation will eventually be checked in addition to the withdrawal operation. However, because of the unpredictability of cross-shard transactions in the originating and destination shards, additional delay and replay may be introduced.

3.4.4 Limitations and Drawbacks

1. Improving the efficiency of mining power relies on an incentive scheme designed to encourage miners to engage in Chu-ko-nu mining across $k \rightarrow n$ shards. However, this presents the challenge of centralizing power and imposes extra burdens on lightweight miners.

3.5 Ethereum 2.0

Since its inception in 2014, Ethereum [10] has been at the forefront of decentralized blockchain platforms, introducing a Turing-complete programming language for smart contract development. As the demand for high throughput grows, the concept of Casper-FFG with sharding, often referred to as Shasper [8], emerged as a proposal to transition the existing Ethereum mainnet, a Proof-of-Work (PoW) based single chain, also known as Ethereum 1.0, to a new sharded architecture.

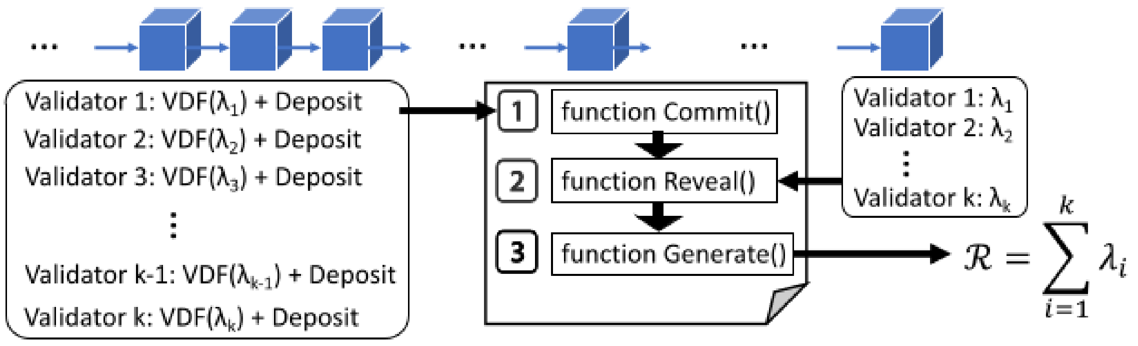
The goal of Ethereum 2.0 is to combine several parallel shards, including the present Ethereum mainnet, where each shard handles transactions and data storage simultaneously [9]. Every round, a local validator is chosen in each shard to produce new blocks until the validator group is globally redistributed. The block headers of blocks with multiple shards are signed by specific validators known as attestors, who may or may not be members of the same shard. Validators store headers for all shards. To maintain network liveness, the blocks are ultimately kept on the main chain, also known as the Beacon chain.

The root validators registry for the shard chains is maintained and recorded by the Beacon chain, which is the manager of the sharding architecture. It also keeps track of network validators and their stakes. In Ethereum 2.0, a node must make a one-way transaction stake deposit into a particular contract account in order to become a validator. The receipt can be used in the shards as an attestation for voting the validators when it has been confirmed. In addition to supporting cross-shard transactions, the Beacon chain is used to elect block validators, manage validators and their stakes, and impose incentives and penalties.

However, this concept was ultimately abandoned in favour of a new approach called proto-danksharding [17], which does not involve multiple chains.

3.5.1 Randomness Generation

To generate randomness, Shasper utilizes a combination of RANDAO [19] and a verifiable delay function (VDF) [5].



Validator 1: VDF(λ_1) + Deposit
Validator 2: VDF(λ_2) + Deposit
Validator 3: VDF(λ_3) + Deposit
Validator k-1: VDF(λ_{k-1}) + Deposit
Validator k: VDF(λ_k) + Deposit
1 function Commit()
2 function Reveal()
3 function Generate()
Validator 1: λ_1
Validator 2: λ_2
Validator k: λ_k
$\mathcal{R} = \sum_{i=1}^{k} \lambda_i$

Figure 10: Randomness generation in Ethereum 2.0

RANDAO is executed using smart contracts on the beacon chain, featuring primarily three functions (Figure 10):

1. Commit(): If all validators involved have deposited a minimum of 32 ETH, they will collectively choose a seed R and execute a verifiable delay function (VDF)

according to the following procedure:

$$\text{VDF}(R_i) = H(H(H(...H(R_i)))) \text{ ... } (6)$$

2. Reveal(): Validators can disclose their local seed R via a smart contract, allowing verification to confirm its alignment with the commitment:

$$H^{-1}(H^{-1}(H^{-1}(...H^{-1}(\text{VDF}(R_i))))) \text{ ... } (7)$$

3. Generate(): Randomness is generated by consolidating all R_i, and validators who fail to disclose their seeds within the allotted time will face penalties.

3.5.2 Consensus Algorithm

In order to overcome the 1% attack vulnerability within intra-consensus, Shasper uses a consensus method based on BFT. In particular, Shasper's Casper-FFG [8] is comparable to ByzCoin [22] and ByzCoinX [23] from OmniLedger, scalable BFT-based PoS consensus algorithms [15], [18].

Using validators from different shards, referred to as "attesters," in intra-shard consensus, Shasper divides the member allocation and consensus procedures. Every slot, the set of testers linked to a certain shard can be modified. Therefore, regardless of the original shard allocation at the start of each epoch, an eligible validator in Shasper must retain all block headers (also known as collations) of all shards. The procedures can be outlined as follows:

1. A node must deposit a specific amount of Ethereum (ETH) into an official smart contract on the original PoW-based mainnet (currently set at 32 ETH [8], [28]) in order to be eligible to operate as a validator. The system registers the node as a legitimate validator on a brand-new, unique chain—known as the beacon chain—as soon as the deposit is verified. The beacon chain serves as the hub for the whole Shasper protocol, maintaining the global validator pool, creating randomness, controlling rewards, and enabling message exchanges, among other responsibilities.

2. The validator pool is globally reshuffled on a regular basis, reassigning all validators to various shards according to data that is generated at random. This rotation happens every 6.4 minutes at the moment [8], [30]. A proposer is chosen at every 8-second interval throughout each epoch from the local validator pool in each shard [8]. The proposer then notifies all attesters assigned to that shard of a proposed collation that contains transactions for their particular shard. In the event that the consensus procedure is successful, the local ledger has the finalized collation.

3. Apart from retaining the hash value of every block on the PoW-based mainnet, the beacon chain selects 400 validators at random from the global validator pool for every shard every 100 collations to complete a checkpoint [16]. After that, all checkpoints are combined by these chosen validators and uploaded to the beacon chain. The beacon chain acquires the local state and a set of finished transactions (and related receipts) for each shard by saving the checkpoints with the collation headers of all shards. Unlike Ethereum 1.0, which relies on probabilistic finality, this enables deterministic finality.

To achieve maximum security but at a considerable overhead, the set of attesters can be redistributed for every proposed collation inside a time frame.

3.5.3 Limitations and Drawbacks

1. The computation overhead of a VDF, which involves performing n iterations of the $Hash$ function, is $O(n)$, rendering it inefficient.

2. The intra-consensus algorithm, utilizing attesters for security, results in notable overhead due to several factors: 1) each shard conducting consensus among validators that are continuously updated; 2) validators requiring storage of data from multiple shards; and 3) the execution of reallocation within a single slot period.

3. This design is vulnerable to a censorship attack [7], wherein malicious validators can inundate a block with irrelevant transactions offering high gas fees. This could potentially disrupt the Commit process as the block's gas limit is reached.

4. This design is susceptible to a grinding attack [13] if the seed R_i originates from the hash of the parent block. In this scenario, malicious validators could submit various transactions to identify the most skewed seed by collecting different sets of transactions.

3.6 Gearbox

Gearbox [15] presents a novel method that makes use of the difference between liveness and safety in shard consensus. By separating these ideas, the protocol enables the dynamic modification of shard parameters to maximize efficiency in response to the system's current degree of corruption. In order to accept modest deviations from honesty while maintaining overall safety, the protocol first selects a relatively small shard, maybe with a lower honesty ratio, and carefully weighs safety and liveness in the consensus procedure. In the worst case, a shard must have a greater honesty ratio in order to survive. The researchers use a "control chain" that is always safe and operational to identify liveness failures. To guarantee that all shards continuously function securely and effectively, shards that don't fulfill liveness standards are resampled with larger shard sizes and liveness tolerances until they reach the required level of liveness.

3.6.1 Safety-Liveness Dichotomy Consensus

When assuming that 30% of the total node population is malicious, it follows that any random subset from this population is also expected to contain approximately 30% malicious nodes. However, real-world deployments may experience a higher percentage of adversarial nodes, presenting significant challenges. The Chernoff bound offers a means to limit this additional corruption (ϵ) to an arbitrarily small value when the random subset is sufficiently large. For instance, in a population of 10,000 nodes with a minimum of 30% malicious nodes, a shard size of at least 5,886 nodes is necessary to ensure that the shard's corruption does not exceed 1/3rd.

Existing approaches employing smaller shards either tolerate less than 30% overall corruption across the entire population or compromise on security. This paper's challenge lies in determining the optimal shard size while considering the current corruption ratio in the entire network.

The security of a blockchain system is characterized by two fundamental properties: (1) Liveness ensures that the blockchain will ultimately deliver new messages to all peers, while (2) Safety ensures consensus among peers regarding the sequence of messages.

The safety threshold (S) and liveness threshold (L) indicate the maximum proportion of adversarial participants within a shard, ensuring the preservation of safety and liveness guarantees.

For partially synchronous model, $2L + S < 1$, ensuring $S < \frac{1}{3}$, if $L = S$ and for synchronous model $L + S < 1$, ensuring $S < \frac{1}{2}$, if $L = S$. This paper essentially focuses on a partially synchronous model network.

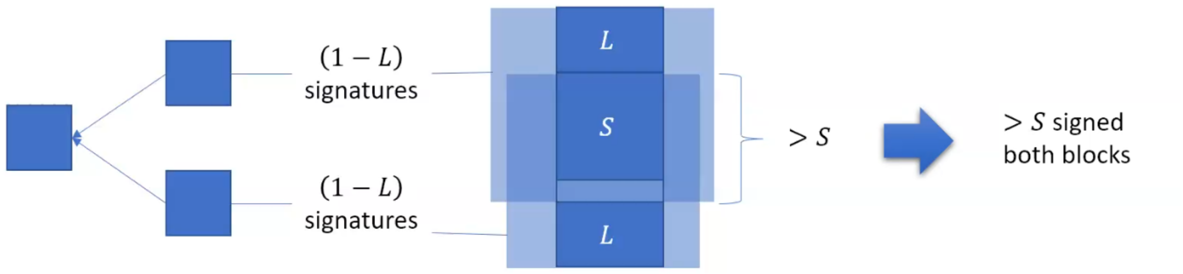
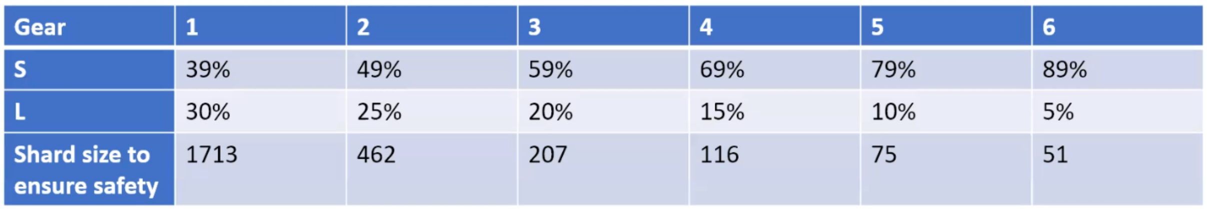


Figure 11: Safety Liveness Dichotomy

To ensure liveness during block commitment, the block must be endorsed by a fraction of nodes equal to $1 - L$, where the proportion of corrupted nodes is no more than L. Regarding safety, in the case of two conflicting blocks, if $1 - L$ of the nodes have endorsed both blocks, there will be an intersection among the endorsers indicating that more than S parties have endorsed both blocks. As S represents the maximum number of malicious parties, there must be at least one honest node that endorses both blocks (Figure 11), thus leading to the detection of the conflict.

To run a consensus based on the dichotomy of this partially synchronous network $2L + S < 1$, the protocol needs to continuously change in L and S based on the malicious nodes present in the network (Assumption of 30% malicious nodes is not always typical behaviour in real-world). Gearbox [15] presents 6 gears to change the configuration of the network based on the situation (Figure 12). When the fraction of malicious nodes is less, we can lower the size of the shard as well as the liveness threshold (L) to preserve more safety (S). Similarly, in a situation where the fraction of malicious nodes are high, we can increase the size of the shard to preserve safety (S) thus increasing L.



Gear	1	2	3	4	5	6
S	39%	49%	59%	69%	79%	89%
L	30%	25%	20%	15%	10%	5%
Shard size to ensure safety	1713	462	207	116	75	51

Figure 12: Gears of GearBox

3.6.2 Shard Liveness Monitoring

At the highest gear, Gearbox [15] is expected to perform the highest throughput without compromising security but may get into a liveness attack by malicious nodes as the fraction of L is very high. So, the challenge is to monitor the liveness of each shard.

A control chain employing $S = L = \frac{1}{3}$ (ensuring constant safety and liveness) and encompassing all nodes within the system is utilized to oversee the liveness of every shard. Upon the addition of a block, each shard transmits heartbeat status updates to this control chain, signifying its progression. If a shard encounters a liveness attack and fails to provide progress updates to the control chain, it triggers a heartbeat timeout for that shard within the control chain. Consequently, the control chain initiates a shard reset by adjusting its parameters and extending the timeout, while assigning a different committee. Through this process, the optimal shard size for the affected shard will be determined over time.

3.6.3 Limitations and Drawbacks

1. Simply altering shard sizes in both directions to accommodate dynamic changes in L is not feasible. Attackers could exploit this method to trigger a loop of "switching gears," resulting in frequent and expensive shard adjustments.

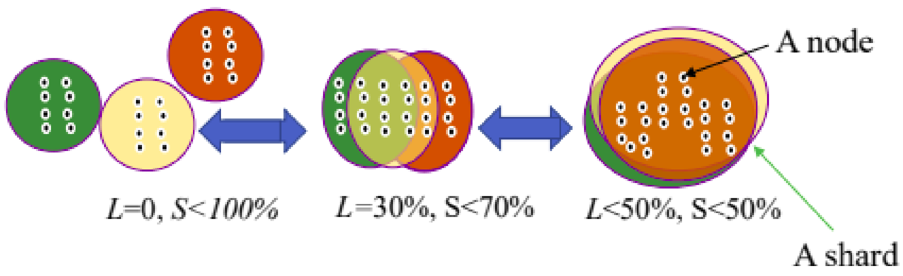


Figure 13: Overlapping of Shards with Increase in Liveness

2. Gearbox might face issues with overlapping shards (Figure 13) since it adjusts shard sizes without altering the total number of shards. These overlapping shards could lead to duplicated workloads, requiring additional transactions involving multiple shards. Consequently, there might be a rise in cross-shard transactions and a decrease in parallelism.

3. Shards must wait for the finalization of their individual blocks within the control chain before advancing to the next epoch.

3.7 Reticulum

Reticulum [36] uses a dynamic, two-phase architecture that allows it to adapt transaction throughput to adversarial attacks on safety and liveness that occur in real time. It is made up of shards labeled "control" and "process," arranged into two levels that represent the two stages. Subsets of control shards are called process shards, and each process shard is supposed to have at least one extremely reliable, honest node in it. On the other hand, it is anticipated that the majority of nodes under control shards will be trustworthy. In the initial phase, Reticulum [36] uses unanimous voting to involve fewer nodes in block acceptance or rejection, allowing for more parallel process shards. The control shard then settles disagreements as they come up and completes the decisions taken in the first phase.

As a member of the public communication chain (PC), every node in the system is responsible for overseeing the sharding protocol, recording shard metadata, and controlling precisely one process shard and its corresponding control shard. The control-shard-chain and the process-shard-chain, which are both dedicated to the control shard and the process shard, respectively, show that every node maintains two other chains in addition to the PC. For the control-shard-chain, a recommended block is called a process block, while for the process-shard-chain, a suggested block is called a control block. A process block is made up of a group of transactions, while a control block is made up of hashes of process blocks and the vote signatures associated with them.

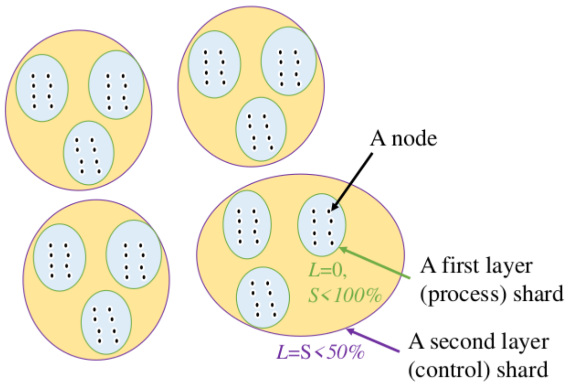


Figure 14: Structure of Reticulum Protocol

Reticulum comprises three main components: bootstrapping, two-layer-shard consensus, and cross-shard transactions.

3.7.1 Bootstrapping

Nodes are arbitrarily and impartially assigned to shards during the bootstrapping stage. All participants in the protocol are first given access to a predefined list of N nodes. Next, utilizing a random beacon, a random sequence C, where $|C| = N$, is collectively created [14]. All participants must be able to access this random beacon, which must also be verifiable by all parties and independent of any one organization. A function called getShardIndex is incorporated into the protocol to ascertain a node's shard membership. When a node $Node_j$ is entered, this function finds the index C_{index} of $Node_j$ in the sequence C, provides the node's control shards, and process shards, respectively. The protocol bootstrapping procedure is only carried out when it first starts or when it restarts with a new node membership. By preventing a node from being assigned to more than one shard, this phase guarantees the secure assignment of shard memberships.

3.7.2 Two-Phase-Shard Consensus

An epoch goes through two stages in the consensus process. The first phase must be finished in the allotted time, T_1, and the second phase must be finished in the allotted time, T_2. While T_2 is dependent on the number of process shards and their performance in the first phase, T_1 stays constant.

1. *First-Phase* - The first stage is to promote consensus about the process blocks among the process shards. Using a conventional binary vote-based synchronous

consensus procedure [17], each process shard aims to produce and come to an agreement on a block at the start of this phase, working within the time constraint T_1. To decide on a block, voting within each process shard requires unanimity. Consequently, agreement cannot be reached and the shard cannot advance to a new state if there is even one adversary node present within the process shard. Consequently, the process blocks show $L = 0$. Generally, this stage ends when a block is accepted by all votes or when no block receives all votes.

Only when a block contains transgressions like as overspending or double-spending transactions that go against earlier rulings will nodes vote against it. Using a Byzantine broadcast protocol called $(\Delta + \delta)$-BB [2], votes cast by nodes during the first phase are broadcast to all nodes in the control shard. Here, $f \leq \lfloor (N_c - 1)/2 \rfloor$, and N_c denotes the number of nodes in the control shard. This protocol makes guarantee that all trustworthy nodes in a control shard will end the broadcast protocol with the same value V if a vote V is broadcast to every node in that shard. As such, they are able to ascertain jointly whether a node in the process shard voted and if so, with what accuracy. Reticulum may identify malicious nodes taking part in the consensus process with reliability thanks to this capacity.

2. *Second-Phase* - As a precautionary measure, the second phase provides a final resolution for blocks that failed to garner unanimous support in the first round of voting. Nodes in the process shards communicate the latest status of the process shard and the problematic block to all nodes in the same control shard in the event that they are unable to come to a consensus on a block.

   A control block outlining the process blocks from every shard under its jurisdiction is proposed by a leader selected from the control shard. This data contains the signatures of the voters and whether the process block was approved unanimously. The block's hash and a choice to accept or reject it are included if there isn't unanimous agreement. If nodes in the control shard agree with the decision made by the current leader, they validate these blocks and vote for the control block. The second phase's time, T_2, is modified in accordance with the quantity of process shards that successfully complete the control shard (designated as N_s).
   The calculation for T_2 can be expressed using the following equation:

$$T_2 = \lambda(\lfloor N_c/N_p \rfloor - N_s + 1)$$

   where λ represents a pre-defined constant value, N_p and N_c are the size of process shards and control shards respectively. To guarantee that the network bandwidth needed by nodes in the second phase is similar to that of the first phase, T_2 is changed. Only after a control block is accepted does a new epoch commence. A new proposed block, headed by a new leader, goes through voting in the same amount of time as the control block if the majority of nodes in the control shard reject it. T_2.

   The relevant process shard in epoch $X + 1$ will remain in the same state as in epoch X if a block from epoch $X + 1$ is rejected inside an accepted control block.

   A node risks expulsion from the system and the confiscation of its Proof of Stake (PoS) if it either proposes a block with erroneous information (as a leader node), votes for a block with incorrect information, or fails to vote at least $\tau - 1$ times during every τ rounds of voting.

3.7.3 Cross-Shard Transaction

Cross-shard transaction design is quite similar to Rapidchain's methodology. A series of transactions with a single sender address and a single recipient address in several shards can be decomposed from each cross-shard transaction. We call these individual transactions Tx_{cross}. ps_{send} creates the Tx_{cross}, which is then sent to $ps_{receive}$, which stands for the shards carrying the recipient's address and the sender's address, respectively.

3.7.4 Limitations and Drawbacks

1. Reticulum employs a fixed sharding scheme, meaning that no new members can be added during runtime and shards cannot be respawned. A system reboot is required to add new nodes.

2. No proof for cross-shard isolation is given in the paper. It could fail the same way Rapidchain's isolation failed.

A comparison table (Table 1) presenting details of the seven discussed sharding protocols is provided on the subsequent page.

		Elastico	**Omniledger**	**RapidChain**	**Monoxide**	**Ethereum 2.0**	**GearBox**	**Reticulum**
Network Model		Partial Sync	Partial Sync	**Intra** Sync **Total** Partial Sync	Partial Sync	Partial Sync	Async	Sync
Threat Model		Arbitrary Behaviour of Attacker, Round-Adaptive Adversary	Arbitrary Behaviour of Attacker, Mildly-Adaptive Adversary	Arbitrary Behaviour of Attacker, Slowly-Adaptive Adverasary	Similar to Bitcoin	Arbitrary Behaviour of Attacker, Uncoordinated Majority	Static Adaversary	Arbitrary Behaviour of Attacker, Adaptive but Upper-Bounded Adversary
Fault Tolerance	**Intra**	33%	33%	50%	50%	33%	Adaptive	50%
	Total	25%	25%	33%	50%	33%	33%	33%
Intra-Consensus Protocol		PBFT	ByzCoinX	50% BFT	PoW-based (Chu-Ko-Nu) Mining	BFT-Based PoS	BFT with varied FTR	BFT with Unanimous Majority
Transaction Structure		UTXO	UTXO	UTXO	Account	Account	UTXO / Account	N/A
Cross-Shard Communication		N/A	Atomix Involves User (Lock / Unlock)	Divides into 3 new Txns (Lock / Unlock)	Relay Transaction (Lock Free)	Receipt Based (Lock / Unlock)	Customized Atomix (Lock / Unlock)	Customized Rapidchain Protocol (Lock / Unlock)
Additional Security Chain		Global Ledger	Global Identity Blockchain	Reference Committee	N/A	Beacon Chain	Control Chain	Committee Shard, Public Communication Chain
Throughput		13.5 times of Bitcoin's TPR, 1600 Nodes	4000 TPR 1800 Nodes, 25% FTR, Shard Size=600	7300 TPR, 4000 Nodes, 33% FTR, Shard Size=250	1000 times of Bitcoin's TPR, 48000 Nodes	100000 TPR on mainnet	32000 TPR, Shard Size=82. 1561 TPR, Shard Size=2264	2000 TPR with 50% Process Shard Acceptance. 7000 TPR with 95% Process Shard Acceptance.
Adaptive Shard Size		No	No	No	No	No	Yes	No

Table 1: Comparison of the above Blockchain Systems

4 Issues and Future Work

Despite the considerable research efforts dedicated to sharding in recent years, significant challenges persist that necessitate resolution before its broad implementation on a large scale. These challenges encompass various aspects of blockchain technology, including scalability, security, consensus mechanisms, performance optimization, and decentralization.

4.1 Limiting Cross-Shard Communication

One significant challenge in blockchain sharding is the effective limitation of cross-shard transactions, a topic that has received limited attention in research. As the number of shards increases, the need to restrict cross-shard transactions becomes crucial for optimizing throughput. Further exploration in this area is vital to unlock the potential for significantly enhancing throughput in sharded blockchain systems.

4.2 Reducing Transaction Latency

Another challenge in blockchain sharding is the reduction of transaction latency, particularly concerning the deterministic confirmation and finalization time of transactions, which is critical for user experience. Transaction latency tends to worsen as scalable Byzantine Fault Tolerance (BFT) consensus mechanisms implement larger shard sizes to mitigate potential attacks like 1% attack. Addressing this issue is essential for ensuring efficient and responsive transaction processing in sharded blockchain systems.

4.3 Pruning Shard's Ledger

Storing a complete ledger version for each shard imposes significant disk storage overhead on validators, who must track the history of every shard to facilitate cross-shard transactions and epochal reallocation. Implementing a beacon chain to store the global state ledger may introduce throughput bottlenecks due to the beacon chain's limited throughput capacity. Additionally, shard members would face extra overhead in verifying transactions from beacon chain members. This highlights the challenge of balancing storage efficiency and transaction verification speed in sharded blockchain systems.

4.4 Data Validity Problem

Sharding presents a challenge: without validating an entire shard's history, users can't ensure the state's validity [31]. A naive solution relies on a majority of honest validators, assuming less than 1/3 are malicious. However, adaptive corruption weakens security, making validation uncertain. Vlad Zamfir suggests storing neighboring shard states, while fisherman challenges and Snarks offer potential solutions.

4.5 Data Availability Problem

The data availability problem arises when malicious full nodes produce invalid blocks but withhold the content from light nodes. This issue extends to sharding [31], where validators validate every transaction in their shard, but others, including the beacon chain, only download headers. Validators function as full nodes for their shard, while non-validator nodes operate as light nodes. This discrepancy in node roles poses a challenge for ensuring data availability and integrity across the network.

5 Conclusion

Existing sharding schemes have demonstrated commendable throughput improvements and it is evident that there remains untapped potential for further innovation and refinement. This survey has underscored the pivotal role of sharding in the trajectory of scalable blockchain systems, offering a comprehensive overview of the latest advancements in intra-consensus security, cross-shard transaction atomicity, and associated challenges. By meticulously evaluating the strengths and limitations of each approach, we have provided valuable insights into the iterative enhancements documented across various research papers. However, this survey also highlights the need for continued exploration and development to address the remaining challenges and unlock the full capabilities of sharding. Looking ahead, potential avenues for future research offer promising opportunities to push the boundaries of scalability and efficiency in blockchain systems. By fostering collaboration and innovation in these areas, we can propel the evolution of sharding methodologies and pave the way for a new era of decentralized applications and services.

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397*, 2017.

[2] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhiting Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021.

[3] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 72:201–224, 2014.

[4] BitCoinWIKI. Merged mining specification, Aug. 2015. Accessed: Aug. 1, 2019.

[5] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Proceedings of the Annual International Cryptology Conference*, pages 757–788, Santa Barbara, CA, USA, August 2018. Springer.

[6] Dan Boneh, Mary Maller, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT*, pages 435–464, Cham, Switzerland, 2018. Springer.

[7] Vitalik Buterin. The problem of censorship. Blog, 2015 2015. Accessed: Aug. 1, 2019.

[8] Vitalik Buterin. Convenience link to casper+sharding chain v2.1 spec. Ethresear.ch, August 2018. Accessed: Aug. 1, 2019.

[9] Vitalik Buterin. Ethereum sharding faq. https://github.com/ethereum/wiki/wiki/ShardingFAQ, Apr. 2019. Accessed: Aug. 1, 2019.

[10] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[11] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, pages 1–4. Chicago, IL, 2016.

[12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proc. OSDI*, volume 99, pages 173–186, Feb. 1999.

[13] Alexei Chepurnoy. Interactive proof-of-stake, Jan. 2016.

[14] Sarani Das, Tyler Yurek, Zekun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

[15] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 683–696, 2022.

[16] Justin Drake. Ethereum sharding. YouTube, LinkTime 2019. Accessed: Sep. 1, 2019.

[17] Ethereum. Danksharding - Ethereum Roadmap, 2024.

[18] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proc. 26th Symp. Operating Syst. (SOSP)*, pages 51–68, New York, NY, USA, 2017. ACM.

[19] Zhiliang Jia, Rui Chen, and Jia Li. Delottery: A novel decentralized lottery system based on blockchain technology. In *Proceedings of the 2019 2nd International Conference on Blockchain Technology and Applications*, pages 20–25, Xi'an, China, December 2019.

[20] Aljosha Judmayer, Alexei Zamyatin, Nicholas Stifter, Alexandros G. Voyiatzis, and Edgar Weippl. Merged mining: Curse or cure? In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 316–333. Springer, Cham, Switzerland, 2017.

[21] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. https://peercoin.net, 2013.

[22] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, pages 279–296. USENIX Association, Aug. 2016.

[23] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE symposium on security and privacy (SP)*, pages 583–598. IEEE, 2018.

[24] Manlu Liu, Kean Wu, and Jennifer Jie Xu. How will blockchain technology impact auditing and accounting: Permissionless versus permissioned blockchain. *Current Issues in auditing*, 13(2):A19–A29, 2019.

[25] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Computer Science Review*, 46:100513, 2022.

[26] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30, 2016.

[27] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 9:15, 2008.

[28] J. Y. Park. Preparing for ethereum pos staking in 2019, December 2018.

[29] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[30] Joseph Prestwich. What to expect when eth's expecting. Hackernoon, January 2019. Accessed: Aug. 1, 2019.

[31] Near Protocol. Unsolved problems in blockchain sharding, 2022.

[32] Ai Qu. The blockchain trilemma: What is it and can it be solved?, 2021.

[33] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities 'honest or bust' with decentralized witness cosigning. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 65–80, Austin, TX, USA, August 2017. USENIX Association.

[34] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61, 2019.

[35] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, pages 95–112, 2019.

[36] Yibin Xu, Jingyi Zheng, Boris Düdder, Tijs Slaats, and Yongluan Zhou. A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance. *arXiv preprint arXiv:2310.11373*, 2023.

[37] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.

[38] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018.