information

MDPI

Article

EquiFlowShard: A Blockchain Sharding Protocol with Optimized Account Distribution

Yemin Chen *, Yongdong Wu and Tong Li

Department of Cyberspace Security, Jinan University, Guangzhou 510632, China; wuyongdong@jnu.edu.cn (Y.W.); litong1998@stu2022.jnu.edu.cn (T.L.)

* Correspondence: shinobu@stu2022.jnu.edu.cn

Abstract: Blockchain sharding is a scalable solution for distributed ledgers, but may be hindered due to cross-shard transactions and uneven workload distribution. This paper presents EquiFlowShard, an advanced blockchain sharding protocol designed to improve robustness and enhance cross-shard efficiency. Specifically, by employing Optimized Account State Distribution Algorithm (OSADA), EquiFlowShard dynamically assigns and segments account states, so as to minimize cross-shard transaction volume and balance shard workloads. In addition, the protocol introduces the SFlow mechanism to facilitate secure and consistent state transfers and a Smooth Transition scheme to mitigate performance impacts during state reconfigurations. Evaluation results confirm that EquiFlowShard outperforms existing benchmark protocols in terms of throughput, transaction confirmation latency, and cross-shard transaction ratio, demonstrating its effectiveness in dynamic blockchain environments.

Keywords: blockchain; sharding technology; account; state segment/aggregate; intermediary account

Received: 13 December 2024
Revised: 21 January 2025
Accepted: 23 January 2025
Published: 25 January 2025

Citation: Chen, Y.; Wu, Y.; Li, T. EquiFlowShard: A Blockchain Sharding Protocol with Optimized Account Distribution. *Information* **2025**, *16*, 92. https://doi.org/10.3390/info16020092

1. Introduction

The advent of blockchain technology has revolutionized the landscape of digital transactions and data management, offering a decentralized and immutable ledger system. Despite its transformative potential, blockchain technology faces significant scalability challenges that hinder its widespread adoption. Traditional blockchain architectures, which rely on linear transaction processing and global consensus mechanisms, struggle to meet the demands of high transaction throughput and low latency, especially in dynamic and resource-constrained environments like the Internet of Things (IoT). The increasing adoption of IoT in various industries further amplifies these demands, as IoT devices continuously generate vast amounts of data that require secure, decentralized handling across distributed networks.

To enhance transaction throughput, overcome the limitations of traditional blockchains, and meet real-world application demands, researchers have extensively studied methods to improve blockchain scalability. Among these, sharding technology is regarded as a key approach for effectively enhancing system scalability and addressing the issue of insufficient blockchain throughput. Prominent sharding techniques [1–6] propose dividing the entire blockchain network into multiple segments that independently process transactions and maintain states, thereby improving overall efficiency. Sharding can be further categorized into three types: network sharding, transaction sharding, and state sharding. Among these, state sharding allows each node to store only the ledger of its shard, significantly reducing storage overhead and improving throughput. Despite its potential to significantly

enhance blockchain performance, state sharding faces substantial challenges, making it the focus of this study. Although theoretically promising, practical implementations of state sharding remain scarce, and existing solutions encounter major challenges. For instance, a high proportion of cross-shard transactions [4,6–8] incurs excessive communication overhead, causing latency issues. Moreover, uneven workload distribution among shards often leads to performance bottlenecks, where certain shards are overloaded while others remain underutilized.

To address these challenges, several solutions have been proposed. Pyramid [6] introduced a two-layered sharding architecture, enabling certain nodes with superior hardware to belong to multiple shards, moderately improving throughput. However, this approach increases shard management complexity and hardware requirements, limiting its scalability in dynamic environments. BrokerChain [9] and LB-Chain [10] proposed account-partitioning methods to balance workload across shards and reduce transaction delays, while these methods alleviate workload imbalance, they fall short in significantly reducing cross-shard transactions, which continue to hinder performance. Building on these efforts, Huang et al. [11] proposed the Transformers protocol, which uses the community-aware CLPA algorithm for account allocation, balancing cross-shard transaction ratios and workload distribution. However, CLPA, as a non-overlapping community detection algorithm, is suitable for account partitioning but not for segmentation. Additionally, most existing methods rely on static configurations and fail to adapt to the blockchain's dynamic nature, leading to potential performance degradation over time. These limitations highlight the need for a dynamic and adaptive protocol that can optimize workload distribution while minimizing cross-shard transactions.

Motivation: Existing blockchain sharding solutions face significant limitations that hinder their scalability and adaptability in dynamic environments. First, static and inflexible account allocation methods are prevalent in many existing protocols, such as CLPA, a non-overlapping community detection algorithm that can only identify the single shard most closely associated with a node. This approach is limited to account partitioning but is unsuitable for account segmentation, as it fails to account for dynamic and overlapping relationships among shards, which are crucial in reducing cross-shard transactions and balancing workload in real-world applications. Second, BrokerChain [9] adopts a static intermediary setup, where high-frequency accounts are predefined as intermediaries based on historical transaction data. However, this approach cannot dynamically respond to blockchain's fluctuating workload or evolving transaction patterns, especially in scenarios like IoT, where node behavior and transaction characteristics are highly dynamic. Consequently, the lack of dynamic selection, allocation, and updating mechanisms for intermediaries can lead to suboptimal performance and increased cross-shard transaction delays. Additionally, existing account segmentation strategies are overly simplistic, often distributing account states uniformly across shards without considering the closeness of relationships between accounts and shards. This may result in insufficient balances in certain shards, necessitating excessive cross-shard transfers to complete transactions. Such transfers not only introduce latency but also degrade the overall performance and scalability of blockchain systems. In summary, these limitations highlight the need for a more dynamic, efficient, and context-aware sharding solution to address the gaps in account allocation, intermediary management, and state segmentation.

Figure 1 illustrates the impact of different agent account configurations on sharding. An agent account handles cross-shard transactions by splitting each into two intra-shard sub-transactions. In Figure 1a, the system includes six accounts and four transactions(TX), among which $TX_{A\to D}$ and $TX_{B\to E}$ are cross-shard transactions. The system's transaction load is 6, with each cross-shard transaction contributing to the load of both involved

shards. As shown in Figure 1b, when account C is selected as the agent account, the cross-shard transactions $TX_{A\to D}$ and $TX_{B\to E}$ are split into sub-transactions handled by C. This increases the total number of transactions to 6 but eliminates cross-shard transactions. The system's transaction load remains 6. In contrast, when account D is selected as the agent account, as depicted in Figure 1c, the number of transactions decreases to 5, reducing the system's transaction load from 6 to 5. Account D's state is evenly divided across two shards. In Shard 2, account D participates in three transactions, two as the sender, while in Shard 1, it participates in only two transactions, both as the recipient. Clearly, account D should hold a larger proportion of its total balance in Shard 2 to reduce the likelihood of insufficient balance for initiating transactions, which would otherwise require transferring balance from Shard 1 to Shard 2—a process that constitutes a cross-shard transaction. After adjusting the state allocation ratio, as shown in Figure 1d, account D's total state is set to 1, with 0.9 allocated to Shard 2 and only 0.1 remaining in Shard 1. This allocation better reflects the actual transaction distribution across shards, minimizing the likelihood of intra-account state transfers and further reducing cross-shard transactions.

Figure 1. The effects of different agent account settings. (**a**) Without agent account. (**b**) Agent account with low association. (**c**) Agent account with high association and even split of state. (**d**) Agent account with high association and appropriate split of state.

This paper proposes an efficient cross-shard blockchain protocol, EquiFlowShard. Before each consensus round, ordinary accounts are dynamically allocated to optimal shards based on historical transactions, while active accounts are segmented across relevant shards to serve as intermediaries, termed LinkAgents, for handling cross-shard transactions.

EquiFlowShard introduces a novel approach to blockchain sharding by addressing critical challenges in state management and cross-shard transaction efficiency. Unlike existing sharding protocols such as BrokerChain, which focus primarily on static account allocation or predefined intermediary configurations, EquiFlowShard dynamically optimizes account state distribution through the Optimized Account State Distribution Algorithm (OASDA). This innovation allows for real-time adaptability to changing transaction patterns and shard workloads, reducing cross-shard transaction overhead and ensuring balanced resource utilization. Furthermore, the integration of the State Flow (SFlow) mechanism and Smooth

Transition scheme enhances state consistency and mitigates the performance impact of state reconfigurations, setting EquiFlowShard apart from traditional approaches.

The potential applications of EquiFlowShard extend beyond general blockchain systems into industrial and IoT environments. In IoT systems, where vast networks of devices generate high transaction volumes with diverse and dynamic patterns, EquiFlowShard's ability to adaptively manage account states and balance workloads across shards becomes particularly advantageous. Its real-time monitoring and feedback mechanism ensures optimal shard performance, even in scenarios with fluctuating device connectivity and varying data traffic. Moreover, the protocol's focus on minimizing cross-shard transactions aligns with the need for low-latency and high-throughput operations, which are critical in IoT-based applications such as supply chain management, smart manufacturing, and energy grid optimization. By addressing these challenges, EquiFlowShard not only enhances blockchain scalability but also provides a robust foundation for integrating blockchain technology into IoT ecosystems.

The main contributions of this paper are summarized as follows:

1. A dynamic Optimized Account State Distribution Algorithm (OASDA) is proposed to allocate ordinary accounts to the optimal shard while selecting the most active accounts as LinkAgents and segmenting their states across an optimal set of shards. This reduces cross-shard transaction proportions, balances cross-shard workloads, and significantly enhances throughput.
2. The State Flow (SFlow) mechanism is designed, comprising two parts: one for the system's optimization phase, where state redistribution is achieved via a communication mechanism based on OASDA results, and another for the consensus phase, where state transfers between segmented sub-accounts are executed through transactions.
3. To ensure smooth state updates, a Smooth Transition scheme is developed. During state reconfiguration, fine-grained account locking is used to minimize the impact on system performance. Additionally, a transitional phase is introduced for LinkAgent deactivation, establishing a three-phase life cycle.
4. The EquiFlowShard protocol is implemented and simulated using Ethereum historical transaction data. Experimental results show that this protocol outperforms other benchmark protocols in terms of throughput, transaction confirmation latency, and the proportion of cross-shard transactions.

2. Related Work

2.1. Blockchain Sharding

Numerous sharding solutions have been proposed to enhance blockchain scalability. Elastico [1] is recognized as the inaugural sharding protocol for public blockchains, segmenting transactions across different shards to facilitate parallel processing. However, it implements only transaction sharding, requiring nodes to store the entire state ledger, which incurs significant storage and communication overhead. To address these limitations, OmniLedger [3] introduces a state sharding protocol using a scalable BFT-based consensus algorithm to enhance transaction throughput. RapidChain [2] further reduces the overhead associated with sharding reconfiguration. Chainspace [12] adopts a distributed atomic commit protocol to support smart contract sharding. Monoxide [5] employs an account/balance-based sharding model and introduces a relay transaction mechanism to handle cross-shard transactions efficiently. Other advancements include OptChain [4], which minimizes cross-shard transactions in UTXO-based systems, and Prism [13], which optimizes network throughput by deconstructing the blockchain structure into atomic functionalities. Dynamic sharding systems, such as those proposed by Tao et al. [14], improve throughput by adapting to smart contract-based operations.

In blockchain sharding architectures, operations are segmented into epochs, each comprising two pivotal phases: the consensus phase and the reconfiguration phase. In the consensus phase, shard committees process transactions and generate blocks by achieving intra-shard consensus. In the reconfiguration phase, blockchain nodes are allocated to different shards based on specific policies to avoid Sybil attacks.

2.2. Processing a Cross-Shard Transaction

In state sharding, nodes within each shard store only a portion of the overall state ledger. Since cross-shard transactions involve inputs and outputs across different shards, verifying these transactions requires communication and coordination between multiple shards to ensure consistent state updates. Therefore, efficiently and securely handling cross-shard transactions is a major challenge in state sharding.

Omniledger [3] employs the Atomix protocol, based on the Two-Phase Commit mechanism with locking and rollback processes, to handle cross-shard transactions, ensuring transaction atomicity and state consistency. However, its reliance on the 2PC mechanism introduces communication overhead and delays, especially in environments with frequent cross-shard interactions. Furthermore, the protocol heavily depends on the client. Chainspace [12] integrates the BFT mechanism with the S-BAC (Sharded Byzantine Atomic Commit) protocol to manage cross-shard transactions, significantly improving system efficiency and scalability. Its lightweight communication overhead and rapid state synchronization enable high throughput and low latency. However, its complex architecture and potential bottlenecks require further optimization to adapt to more dynamic and intricate blockchain environments. RapidChain [2] reduces communication overhead and latency significantly by parallelizing the processing of transaction parts in each shard and employing a lightweight communication protocol along with local and global consensus mechanisms. However, its performance is highly sensitive to network latency and stability, degrading significantly in high-latency or unstable environments. Monoxide leverages the asynchronous reference mechanism, allowing cross-shard transactions to reference the state and results of other shards via relay transactions, eliminating the need for synchronous communication between shards. However, its account allocation strategy, which maps accounts to shards based on the modulo operation of the last 8 bits of their addresses, can lead to a "hot shard" issue, where a large number of accounts become concentrated in a single shard. Pyramid [6] introduces a hierarchical sharding consensus protocol, where special nodes with better hardware handle cross-shard transactions. However, this approach imposes high hardware requirements and results in significant storage overhead. Subsequently, BrokerChain [9] proposed a special type of account to act as a broker, replacing Monoxide's relay transaction mechanism for managing cross-shard transactions, thereby ensuring atomicity and security. However, the brokers in BrokerChain are statically predefined, making it difficult to adapt to the dynamic nature of blockchain systems. In UTXO model, Estuary [15] divides all accounts into small state units, enabling all user transactions to be handled within a single shard. However, this approach may lead to inefficiencies, as the segmentation of all accounts can result in insufficient balances for some. This could trigger frequent state transfers between shards during transaction processing and may struggle to adapt to the dynamic and rapidly changing nature of blockchain systems. Sharon [16] converts cross-shard transactions into intra-shard transactions through shard rotation and merging mechanisms. However, during each consensus phase, Sharon requires repeated pairwise shard merging, with a time complexity of $O(n^2)$. Additionally, the merging process involves synchronizing large amounts of state data, resulting in significant transaction processing delays and storage overhead.

A comparative analysis of EquiFlowShard and related protocols reveals its advantages: (1). Monoxide employs a relay transaction mechanism for cross-shard transactions. However, its static, address-based sharding method often leads to hot shard issues, resulting in workload imbalances. In contrast, EquiFlowShard leverages the OSADA to dynamically allocate account states, effectively mitigating hot shard problems and enhancing workload balance across shards. (2). BrokerChain introduces static brokers to manage cross-shard transactions, ensuring atomicity. However, static brokers perform poorly in dynamic environments. EquiFlowShard addresses this limitation by implementing a dynamic LinkAgent selection mechanism and life-cycle management, significantly improving the protocol's adaptability and scalability. (3). Estuary reduces the proportion of cross-shard transactions by segmenting all accounts into smaller state units. However, excessive segmentation granularity may introduce additional challenges. EquiFlowShard overcomes this by selectively segmenting only active accounts based on the LinkAgent selection algorithm, leaving other accounts unsegmented. Furthermore, the state segmentation ratio is determined by the account's association with specific shards rather than equal distribution, ultimately reducing the number of state transfers between accounts.

3. Overview of EquiFlowShard

In this section, we introduce EquiFlowShard, an efficient cross-shard blockchain sharding protocol capable of adaptively allocating and segmenting accounts.

3.1. System Model

The overview of EquiFlowShard is shown in Figure 2. Similar to existing blockchain protocols [2,9,11,17], EquiFlowShard employs the PBFT [18] protocol for consensus, with a partially synchronous peer-to-peer network for communication, where nodes exchange messages through the Gossip protocol. Each node is assigned to a single shard and maintains its own independent ledger. A fixed system runtime interval, defined as an epoch, consists of a consensus phase and an optimization phase. EquiFlowShard includes two types of shards:

- **Optimization Shard (O-shard)**: O-shard is responsible for optimizing the distribution of account states. In each epoch, it adaptively allocates account states and updates the LinkAgent.
- **Processing Shard (P-shard)**: P-shard is responsible for generating blocks during the consensus phase by processing transactions and achieving intra-shard consensus.

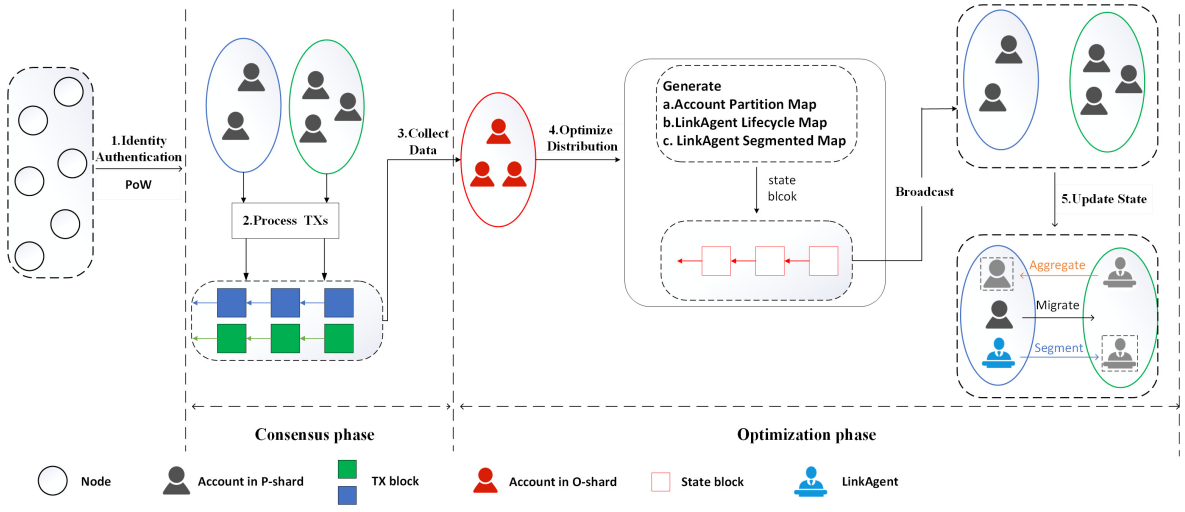


Figure 2. Workflow of EquiFlowShard in an epoch.

To prevent Sybil attacks [19] and join–leave attacks [20,21], the Cuckoo rule is periodically applied to update the composition of O-shards and P-shards.

3.2. Workflow of EquiFlowShard

As shown in Figure 2, the EquiFlowShard protocol's workflow in an epoch consists of the following five steps:

- Step 1, **identity authentication**: To prevent Sybil attacks, at the beginning of each epoch, all consensus nodes generate unbiased and unpredictable randomness based on a Verifiable Random Function (VRF). Similar to [9], nodes must solve a PoW puzzle using their IP address, public key, randomness from the previous epoch, and a flag indicating their willingness to become a LinkAgent in order to obtain a valid identity for this epoch. Accounts intending to become LinkAgents must stake tokens in advance.
- Step 2, **process transactions**: The P-shard continuously receives user transactions and adds them to the transaction pool. During each epoch, the P-shard packages the transactions from the pool, generates blocks, broadcasts them to the shard network, and reaches consensus using the PBFT consensus algorithm.
- Step 3, **collect data**: The O-shard continuously receives block information broadcasted by the P-shard and uses the transactions within the blocks to construct and update a transaction distribution graph.
- Step 4, **optimize distribution**: Based on the transaction distribution graph, the O-shard periodically executes the OSADA to generate the optimal account partition map and updates the composition of LinkAgents and the segmentation map. The O-shard reaches consensus on the algorithm results, generates a state block, and then sends a state reconfiguration message to all P-shards.
- Step 5, **update state**: The P-shard updates its ledger state and transaction pool based on the state block. Active LinkAgents will segment the shard's state according to the segmentation map and distribute it to the respective shards. LinkAgents in the reintegration phase must aggregate all segmented states back into the main shard.

4. EquiFlowShard Protocol Design

This section introduces the EquiFlowShard protocol, covering the optimized account state distribution algorithm, the State Flow mechanism, the Smooth Transition strategy, and the real-time monitoring and feedback mechanism.

4.1. Formulation of Distribution Problem

In EquiFlowShard, the state of ordinary accounts is allocated to a single shard, while the state of LinkAgents is partitioned and distributed across closely related shards.

First, consider the allocation of ordinary accounts. Define an account transaction graph as $G(V, E)$, where V represents a set of N accounts, expressed as $V = \{1, 2, \ldots, N\}$, and E denotes the set of edges between accounts, expressed as $E = \{e_{i,j} | i, j \in [N]\}$. Each edge $e_{i,j}$ represents a transaction between accounts i and j, with its weight indicating transaction volume. The set of neighboring nodes for account i is defined as $Nbr(i)$. We define $S = \{1, 2, \ldots, K\}$ as the set of K shards. Each ordinary account belongs to a single shard, represented by a binary variable $\delta_{i,k} (k \in [K])$, which indicates whether account i is in shard k. If $\delta_{i,k} = 1$, account i belongs to shard k; if $\delta_{i,k} = 0$, account i does not belong to shard k. The partitioning result for accounts is defined as $Pmap$.

One factor to consider in account partitioning is the workload balance of shard k, denoted as W_k, which consists of two components: the intra-shard workload W_k^{in}, generated

by transactions within the shard, and the cross-shard workload W_k^{cross}, generated by cross-shard transactions. Thus, W_k is calculated as follows:

$$W_k = W_k^{in} + W_k^{cross} = \sum_{i=1}^{N}\sum_{j=1}^{N} \delta_{i,k} \cdot \delta_{j,k} \cdot e_{i,j} + \sum_{i=1}^{N}\sum_{j=1}^{N} \delta_{i,k} \cdot (1 - \delta_{j,k}) \cdot e_{i,j} \tag{1}$$

Similar to [9], to enhance blockchain throughput, the workload imbalance index ΔW of EquiFlowShard should be minimized, as shown in the following calculation.

$$\Delta W = \max_{k \in S} \left| W_k(p) - \frac{\sum_{k=1}^{K} W_k(p)}{K} \right| \tag{2}$$

where p represents the partitioning scheme and $W_k(p)$ denotes the workload of shard k under partitioning scheme p. Selecting the absolute difference between the maximum shard workload and the average shard workload as the imbalance metric allows for a direct quantification of system load imbalance, demonstrating higher sensitivity to extreme shard workloads.

Secondly, selecting suitable accounts as LinkAgents and effectively segmenting their states is necessary. A common approach is to select the accounts with the highest number of transactions (i.e., hot accounts) and evenly distribute their states across all shards. However, this method has limitations: if most transactions of a hot account are concentrated in a single shard, segmenting its state does little to reduce cross-shard transactions. Additionally, improper allocation can lead to insufficient balances for LinkAgents in certain shards, requiring frequent transfers from other shards. This increases cross-shard transactions and can degrade blockchain performance.

Finally, it is necessary to address state segmentation and aggregation during system operation to maintain inter-shard consistency. Three scenarios are encountered: (1) When an account is selected as a LinkAgent, its state must be segmented across relevant shards; (2) When a LinkAgent reverts to an ordinary account, each shard's sub-states should be aggregated back to the main shard; (3) When a LinkAgent processes a transaction but has an insufficient balance in the current shard, it may require a balance transfer from other shards.

4.2. Optimized Account State Distribution Algorithm

To optimize account state distribution, this paper introduces the **Optimized Account State Distribution Algorithm** (**OASDA**), which consists of three components: account partitioning, LinkAgent selection, and LinkAgent segmentation.

Account partitioning: SLPA [22] is a well-known overlapping community detection algorithm that initially assigns a random community label to each node, ultimately forming a set of closely associated labels for each node. By combining SLPA [22] with CLPA [11], we propose the Optimized Shard Label Propagation Algorithm (OSLPA). Ordinary accounts receive a single optimal shard label, while LinkAgents obtain an optimal set of shard labels. During the algorithm's execution, each node updates its shard label by calculating the score function $\eta(i,k)$, defined as follows:

$$\eta(i,k) = \alpha \cdot \frac{\sum_{j \in Nbr(i)} e_{i,j} \cdot \delta_{j,k}}{\sum_{j \in Nbr(i)} e_{i,j}} + (1 - \alpha) \frac{min_{m \in S} W_m}{W_k} \tag{3}$$

where α is a hyperparameter that balances the impact of cross-shard transaction rates and inter-shard workload balance on the score. Each node ultimately receives an optimal score set $\eta(i)$. Ordinary accounts select the label with the highest score as their shard, while

LinkAgents select an optimal set of shard labels, with the highest-scoring label designated as the primary shard.

LinkAgent selection: Active accounts should exhibit high transaction frequencies and substantial cross-shard transactions. Therefore, an activity metric θ_i is defined to represent the activity level of each account:

$$\theta_i = TX_i + \beta \cdot TX_i^{cross} \tag{4}$$

where TX_i represents the number of transactions for account i and TX_i^{cross} represents the number of cross-shard transactions for account i. β is the weight that adjusts the influence of the number of cross-shard transactions.

Define the number of new LinkAgents selected in each update round as n_A. All accounts are ranked in descending order based on their activity score θ_i, and the top n_A accounts are selected as LinkAgents for the current round, added to the new LinkAgent queue Q_{new}. Using Q_{new}, the LinkAgent life-cycle map, represented by $Lmap_{LA}$, is updated accordingly.

To adaptively respond to changes in the blockchain, the O-shard may dynamically increase the LinkAgent count, defining the additional amount as n_D. To prevent excessive fragmentation of account states, an upper limit of n_{max} is set for Q_{new}. In EquiFlowShard, the number of active LinkAgents per round is constrained within $[n_A, n_{max}]$.

LinkAgent segmentation: First, for each LinkAgent's score set $\eta(i)$, calculate the closeness $C_{i,k}$ between each shard and the LinkAgent:

$$C_{i,k} = \frac{\eta(i,k)}{\sum_{S=1}^{k} \eta_{i,s}} \tag{5}$$

Define the closeness threshold as γ, and the LinkAgent will segment its state to shards where $C_{i,k} \geq \gamma$. The LinkAgent's state segmentation map is represented as $Smap_{LA}$, where $Smap_{LA}$ denotes the proportion of LinkAgent i's state assigned to shard k, directly correlated with the closeness level, calculated as follows:

$$Smap_{LA}(i,k) = \frac{C_{i,k}}{\sum_{s \in \{x | C_{i,x} \geq \gamma\}} C_{i,s}} \tag{6}$$

Upon completing OASDA, the O-shard broadcasts the algorithm results, reaches consensus, and generates the state block $Sblock_O$. It then broadcasts a message to notify the P-shards to proceed with state reconfiguration.

4.3. State Flow Mechanism

In each epoch, account redistribution triggers dynamic state flows during operation, including state migration, segmentation, and aggregation. To ensure stable and secure system performance, this paper introduces the **State Flow mechanism** (**SFlow**).

The SFlow mechanism comprises two components: (1) **state reconfiguration** (**SR**) during the optimization phase and (2) **cross-state account transfer** (**CSAT**) during the consensus phase.

State reconfiguration: During the optimization phase, upon receiving a state reconfiguration message from the O-shard, the P-shard initiates state migration, along with LinkAgent segmentation and aggregation. Figure 3 illustrates this process.

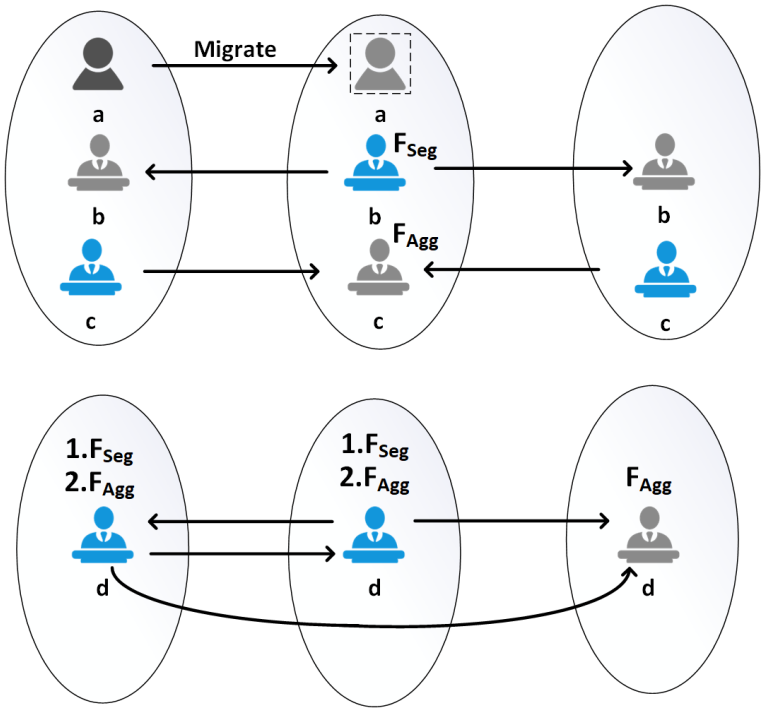


Figure 3. Example of state reconfiguration.

The migration of account a from shard s_1 to shard s_2 proceeds as follows:

1. **Verify** $Pmap$: Confirm that $Pmap(a) = s_2$ and that account a has an existing state $state_{i,1}$ in shard s_1.
2. **Construct and transmit the message**: Shard s_1 sends a message to shard s_2: $Msg_{s_1 \to s_2} = (state_{a,1}, s_2, epochId, nonce, hash(*), \sigma_{s_1})$. The $EpochId$ corresponds to the current epoch of $Sblock_O$, preventing account migration desynchronization due to transmission delays. To protect against malicious nodes replaying $Msg_{s_1 \to s_2}$ to increase account a's balance, a nonce field is included in the message. The $hash(*)$ is a digest of the message content to ensure its integrity and immutability. σ_{s_1} is the signature of shard s_1, guaranteeing the authenticity of the message and preventing repudiation.
3. **Update local ledger**: When s_2 receives the message $Msg_{s_1 \to s_2}$, it verifies the signature and hash values, verifies the nonce and EpochId, and then updates the local ledger state accordingly.

The communication process for LinkAgent state segmentation and aggregation resembles the ordinary account migration process, with key differences: (a) the LinkAgent mapping $Lmap_{LA}$ is checked, and (b) the segmentation function F_{Seg} and aggregation function F_{Agg} are executed.

When executing the segmentation function F_{Seg}, the account balance state is divided based on the state ratio in $Smap_{LA}$. For example, for account b in shard s_2, the segmentation function is executed as $F_{Seg}(state_{b,2}) = (segstate_{b,1}, segstate_{b,2}, segstate_{b,3})$, where $segstate_{b,k} = r_{b,k} \cdot state_{b,2}$.

Executing F_{Agg} aggregates all received sub-states into a consolidated local state. For instance, aggregating account c's states from shards s_1 and s_3 into s_2 would be represented as $F_{Agg}(state_{c,1}, state_{c,2}, state_{c,3}) = state_{c,2}$.

To ensure LinkAgent sub-states across shards match the proportions in $Smap_{LA}$, the system prioritizes F_{Seg}. Due to potential communication delays, some shards may receive aggregation messages with segmented states from other shards before the O-shard's reconfiguration message arrives. Executing F_{Agg} before F_{Seg} could cause inconsistent state

division. Therefore, the system stipulates that F_{Agg} is only called if the current shard has received the O-shard reconfiguration message and does not need or has completed F_{Seg}.

After SR concludes, the P-shard generates a state block, and consensus is reached across shards to ensure consistency among all nodes.

CSAT: During the consensus phase, LinkAgent may encounter insufficient balance in a sub-state within a shard, necessitating a transfer request from sub-states in other shards, known as cross-shard account transfer (CSAT).

To improve the efficiency of state updates in each epoch, SR uses a communication mechanism; however, this approach has lower security and limited flexibility and involves extensive account state locking. For example, if an account has three sub-states and requires state reconfiguration, all sub-states are locked and unable to process transactions. In the consensus phase, if one sub-state needs to initiate a transfer request to another, we aim to ensure the remaining sub-states remain unlocked and can continue processing transactions. To address this, we propose a more secure and efficient CSAT.

CSAT is transaction-triggered. When a LinkAgent in shard k processes a transaction tx and finds the current shard's balance insufficient to support it, the LinkAgent checks its state list, sorted in descending order by balance. It selects the shard with the largest balance as the target shard and initiates CSAT, requesting a transfer to shard k. If a single shard's balance cannot meet the transaction amount of tx, multiple CSAT requests are issued across multiple shards, following the order in the list. The CSAT process is as follows:

- Account i initiates a withdraw transaction TX_w on source shard s_1, specifying destination shard s_2 and transfer amount $txvalue$. To minimize the frequency of CSAT triggers, account i may adjust the transfer amount based on its circumstances, provided the amount is not less than the CSAT-triggering transaction amount $txvalue$. After broadcasting transaction TX_w in s_1, once TX_w is confirmed, i's state $state_{i,1}$ in s_1 is reduced by $txvalue$, and i receives proof of the successful deduction, denoted as $proof_w$.
- LinkAgent i, based on TX_w and $proof_w$, constructs a deposit transaction TX_d on destination shard s_2. Upon broadcasting transaction TX_d in s_2, and after TX_d is confirmed, i's state $state_{i,2}$ in s_2 is increased by $txvalue$.

To ensure LinkAgent functionality, CSAT transactions are prioritized over regular transactions, with shards prioritizing CSAT inclusion in blocks.

4.4. Smooth Transition

To ensure consistency of states before and after reconfiguration, the simplest approach is to suspend all account transactions during reconfiguration, resuming them only once reconfiguration is complete. However, this system-wide lock approach has a coarse granularity and impacts performance. Additionally, when a LinkAgent becomes inactive and reverts to an ordinary account, its state must be aggregated back to the main shard. Communication delays may leave unprocessed cross-shard transactions in the transaction pool, requiring reprocessing. To minimize these impacts and enable smooth state transitions, a **Smooth Transition** scheme is proposed.

Fine-grained account locking:As shown in Figure 4, upon receiving an SR message from the O-shard, each P-shard will lock only the specific accounts needing migration, segmentation, or aggregation. Locked accounts cannot process transactions until their state is fully reconfigured. During this period, unlocked accounts continue to process transactions as usual.

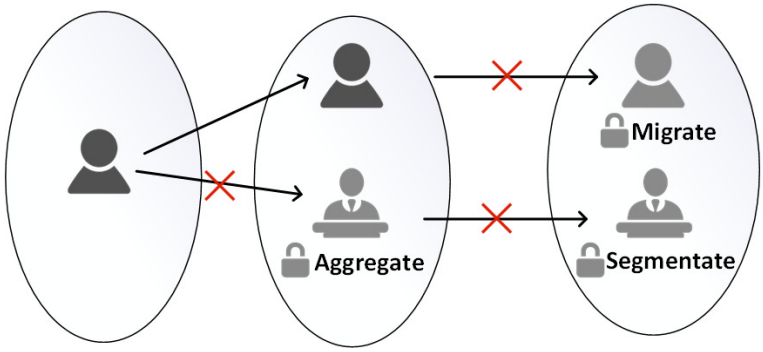


Figure 4. Example of transaction processing in SR.

Transition stages: As shown in Figure 5, each LinkAgent undergoes three life-cycle phases—active, inactive, and reintegration. In the active phase, the LinkAgent handles all transactions. In the inactive phase, it continues processing existing transactions in the shard's pool but does not handle new cross-shard transactions. In the reintegration phase, the shard begins state reclamation for the LinkAgent, aggregating its state back to the main shard and rolling back any unprocessed cross-shard transactions associated with it.

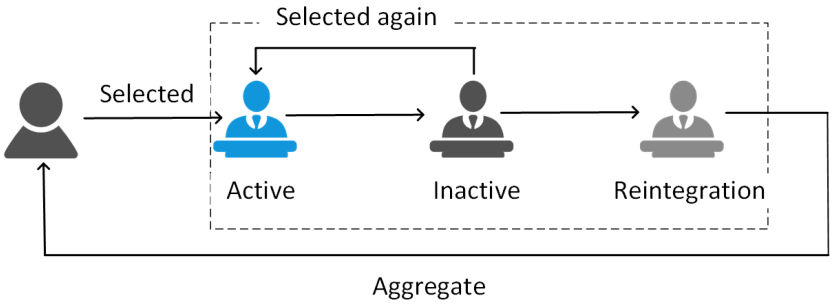


Figure 5. Life cycle of LinkAgent.

4.5. Real-Time Monitoring and Feedback

The optimized state distribution is derived from historical transaction data, but future transaction patterns may not align closely with past characteristics, and blockchain dynamics allow nodes to join or leave at any time. To further optimize state distribution and improve system performance, EquiFlowShard introduces real-time monitoring and feedback.

During the consensus phase, the O-shard continuously monitors critical performance metrics of each P-shard, including transaction throughput (TPS), transaction confirmation latency (TCL), and the cross-shard transaction ratio (CTR). These metrics enable quick off-chain identification of system bottlenecks and anomalies. Additionally, the O-shard conducts real-time monitoring of each LinkAgent, tracking transaction volume, balance changes, and CSAT occurrences. If a LinkAgent's balance is too low or its transaction-handling capacity is insufficient, a warning is issued, and a replacement with a new LinkAgent may be considered.

Based on real-time monitoring data, the following feedback strategies are generated: (1) Dynamic adjustment may be performed of key parameters, such as the score function parameter α in OSADA, the state segmentation threshold γ for LinkAgents, and the additional number of LinkAgents n_D. (2) If a shard experiences a sudden increase in workload, the O-shard can segment active accounts within that shard, temporarily increasing the number of LinkAgents or initiating partial account migration. (3) If a LinkAgent's CSAT usage is excessively high, the distribution ratio of its sub-states can be adjusted accordingly.

(4) When malicious behavior by a LinkAgent is detected, its status can be revoked and the staked tokens confiscated.

For security, all feedback mechanism outcomes require O-shard consensus approval. With real-time monitoring by the O-shard, the entire blockchain system operates efficiently, enhancing responsiveness and adaptability to dynamic environmental changes.

5. Experimental Results

We implemented the EquiFlowShard protocol using Golang on the open-source blockchain prototype BlockEmulator [23]. This transaction-driven blockchain sharding simulator was used to evaluate the proposed sharding protocol. The system runs on hardware configured with an Intel i7-12700H CPU and 32 GB RAM.

Dataset: We utilized real historical Ethereum [24] transaction data, consisting of one million transactions, to evaluate system performance. These transactions were replayed to the blockchain system at a specified arrival rate at the start of each epoch, with transactions prepared by time and assigned to different P-shards based on account states.

Baselines: We considered the following three baselines:

(1). Monoxide [5]: Monoxide allocates accounts by taking the first few bits of the hash address and applying modulo with the number of shards to determine the shard assignment. For cross-shard transactions, Monoxide employs a relay transaction mechanism. When a cross-shard transaction occurs, the source shard generates a relay transaction and updates its internal state. This relay transaction is then broadcast to the destination shard, which directly references the updated state from the source shard without requiring direct synchronization.

(2). BrokerChain [9]: Similar to Monoxide, BrokerChain allocates accounts using the first few bits of the hash address and applying modulo with the number of shards. However, for cross-shard transactions, BrokerChain introduces intermediary accounts called brokers. Cross-shard transactions are split into two sub-transactions: the first half involves the sender transferring assets to the broker in the source shard. Once confirmed, the broker initiates the second half of the transaction, transferring the assets to the recipient in the destination shard. The transaction is considered successful once the second sub-transaction is confirmed.

(3). CLPA-Broker [11]: CLPA-Broker uses the CLPA community detection algorithm for account allocation. Its approach to handling cross-shard transactions is similar to BrokerChain, utilizing brokers to split transactions into two phases for secure processing.

Since the brokers in BrokerChain and CLPA-Broker are predefined based on prior knowledge, brokers in the experiments are randomly selected to ensure a fair comparison between the different approaches.

Metrics: The system's performance is measured by transaction throughput (TPS), transaction confirmation latency (TCL), and cross-shard transaction ratio (CTR), which are defined as follows: (1) TPS: Number of transactions processed per second. (2) TCL: Average time taken for a transaction to move from the transaction pool to final confirmation on the blockchain. (3) CTR: The proportion of cross-shard transactions to the total transactions.

Other parameter settings are as follows: The number of shards $k = 8$, with $m = 4$ nodes per shard and a maximum block size of 2000 TXs, and the account repartitioning interval is set to 50 s. The parameter α, which balances the cross-shard transaction ratio and inter-shard workload distribution, is set to $\alpha = 0.5$ to ensure equal weighting of the two optimization objectives and to prevent dominance by any single factor. A smaller α would result in a higher cross-shard transaction ratio, while a larger α could lead to significant workload imbalance among shards. For the parameter β, which adjusts the influence of cross-shard transactions on LinkAgent activity, experimental results indicate that $\beta = 0.5$

is an optimal value, effectively balancing transaction volume and cross-shard transaction ratio. The parameter γ, which defines the closeness threshold for state segmentation, was experimentally determined to be $\gamma = 0.1$, achieving a good balance between segmentation granularity and the reduction of cross-shard interactions.

5.1. Comparing with Baselines

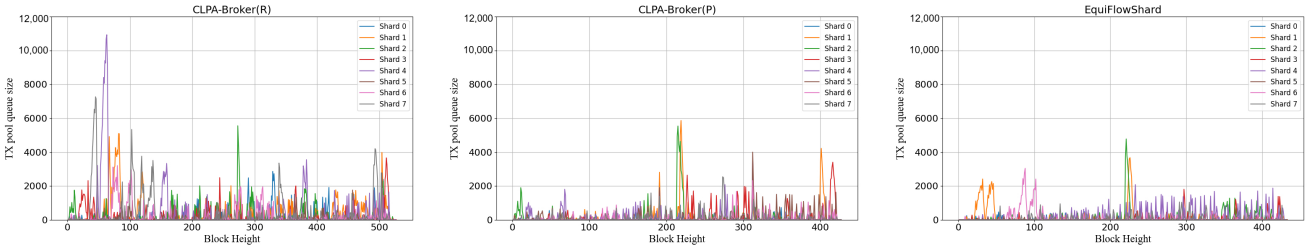
We conducted a comparative analysis between the proposed solution and existing protocols, including Monoxide, BrokerChain, and CLPA-Broker. Both BrokerChain and CLPA-Broker were evaluated in two modes: a random broker mode, denoted by the suffix (R), and a pre-selected broker mode based on historical transaction data, denoted by the suffix (P), where brokers remain unchanged during operation. The number of intermediary accounts was fixed at $n_i = 30$, with a TX arrival rate of 2000 TX/s and a total of 3 million transactions.

The comparative results are presented in Table 1. By comparing BrokerChain and CLPA-Broker, it is evident that account partitioning moderately improves throughput and reduces the proportion of cross-shard transactions. However, a comparison between CLPA-Broker(R), BrokerChain(P), and CLPA-Broker(P) shows that account segmentation leads to greater performance improvements than partitioning alone. Therefore, the selection of accounts for segmentation plays a important role in enhancing performance. Our results indicate that EquiFlowShard slightly outperforms CLPA-Broker(P), with the performance gap increasing as the transaction volume grows, suggesting that pre-defined intermediary accounts cannot adapt well to the dynamic nature of blockchains.

Table 1. Comparing EquiFlowShard and baselines.

Method	TPS	TCL (s)	CTR
Monoxide	1052	1,427,721	0.8817
BrokerChain(R)	918	6499	0.8535
CLPA-Broker(R)	1296	1691	0.4527
BrokerChain(P)	1595	1271	0.3572
CLPA-Broker(P)	1607	967	0.1583
EquiFlowShard	1685	862	0.0632

Additionally, EquiFlowShard is compared with CLPA-Broker(R) and CLPA-Broker(P) by analyzing the transaction queue size in each shard, as shown in Figure 6. A larger queue size indicates a heavier workload and lower processing capacity for the shard. Under CLPA-Broker(R), the maximum transaction queue size reaches 10,000, with some shard queues around 2000–4000, while in CLPA-Broker(P), the maximum queue size is approximately 5000. In contrast, EquiFlowShard maintains a maximum queue size of only 4000, with minimal variation in transaction queues across shards for most of the time.



(**a**) CLPA-Broker(R). (**b**) (**c**)

Figure 6. TX pool queue size between different shards. (**a**) CLPA-Broker(R). (**b**) CLPA-Broker(P). (**c**) EquiFlowShard.

To validate the appropriateness of the account segmentation ratio, comparative experiments were conducted with BrokerChain and CLPA-Broker, where intermediary accounts were represented by hot accounts, denoted by the suffix (H). The intermediary account states were fully segmented and evenly distributed across all shards. In the experiment, the closeness threshold γ varied within the range 0, 0.1, 0.2. As shown in Table 2, when $\gamma = 0.1$ or $\gamma = 0.2$, the number of CSATs was lower than in other schemes, indicating the reasonableness of the segmentation strategy. However, a larger γ is not necessarily better, as there is an optimal value. A higher γ means that the connection between the account and the shard must be very strong for the state to be segmented into that shard, which could reduce the number of shards holding the account's state. Since LinkAgent accounts are typically the most active and transact with many different shards, fewer segmented states may lead to more cross-shard transactions. When the segmentation threshold γ is set too high, no shard may meet the required closeness with the account. In such cases, EquiFlowShard automatically assigns the shard with the highest score as the account's state shard for the current round. However, this results in LinkAgents having states in only a single shard, rendering them incapable of handling cross-shard transactions. If too many LinkAgents, or even all, are reduced to this state, the system may become unable to process cross-shard transactions effectively. In experiments, when $\gamma > 0.3$, a significant number of LinkAgents were found to have states in only one shard. To prevent system failures, it is necessary to designate a minimum number of global LinkAgents. Moreover, the reduction in CSATs due to increasing γ may not outweigh the increase in cross-shard transactions, ultimately affecting system performance. Setting the γ threshold requires a balance between segmentation granularity and system efficiency. A lower γ allows LinkAgent sub-states to distribute across more shards, reducing the load on individual shards but increasing cross-shard interactions, which raises the CTR and communication overhead. Conversely, a higher γ allocates states only to shards with stronger correlations, reducing CTR but potentially increasing the load on specific shards. If shard resources are insufficient, this may lead to a significant rise in TCL. Therefore, the ideal value for γ should be around $1.0/k$, where k is the number of shards.

Table 2. The number of CSATs.

Method	Number
BrokerChain(H)	18,261
CLPA-Broker(H)	26,565
EquiFlowShard ($\gamma = 0$)	24,973
EquiFlowShard ($\gamma = 0.1$)	17,499
EquiFlowShard ($\gamma = 0.2$)	3298

In conclusion, compared to the baselines, the proposed solution demonstrates significant performance advantages, effectively handling large transaction volumes. By selecting appropriate accounts as LinkAgents for state segmentation, the proportion of cross-shard transactions is reduced, resulting in faster transaction confirmation times.

5.2. The Effect of the Number of Segmented Accounts

The intermediary accounts can have their states segmented across different shards. Next, the TX arrival rate was fixed at 2000 TX/s, while the number of intermediary accounts n_i was varied from 10 to 60. The experimental results are shown in Figure 7. As the number of segmented accounts n_i increases, the throughput of all schemes improves gradually. EquiFlowShard's TPS remains between 1500 and 1700, CLPA-Broker's TPS between

1200 and 1400, and BrokerChain's TPS between 1000 and 1200. Clearly, EquiFlowShard outperforms the other approaches.

Figure 7b illustrates that TCL decreases as the number of segmented accounts increases, although the reduction is modest, particularly for EquiFlowShard, which already exhibits low latency.

Figure 7c shows that the cross-shard transaction ratios for both BrokerChain and CLPA-Broker exhibit slight fluctuations as n_i increases. This is because both schemes use randomly selected brokers, which may result in the selection of inactive accounts for segmentation. Even with an increase in the number of segmented accounts, the cross-shard transaction ratio may not significantly decrease. However, compared to BrokerChain, CLPA-Broker applies the CLPA method to allocate accounts to appropriate shards, which reduces the cross-shard transaction ratio to some extent. EquiFlowShard, by adaptively selecting hot accounts for segmentation, steadily decreases the cross-shard transaction ratio as n_i increases.

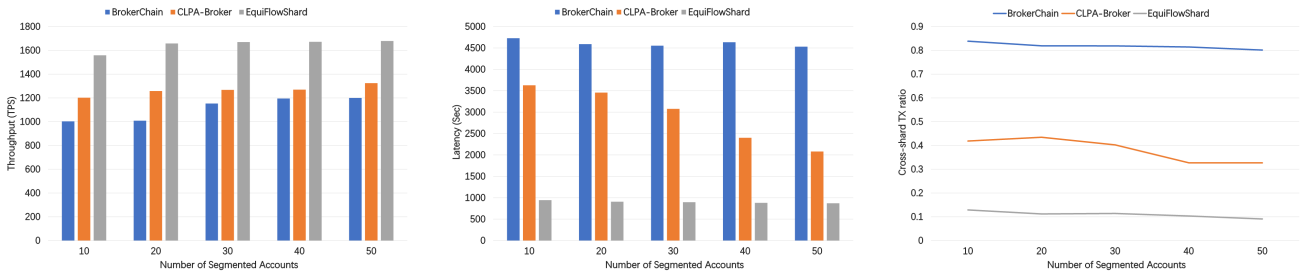


Figure 7. Effect of the number (**a**) Throughput. (**b**) TX confirmation latency (s). (**c**) Cross-shard TX ratio.

5.3. The Effect of TX Arrival Rate

We compared the impact of different TX arrival rates on BrokerChain, CLPA-Broker, and EquiFlowShard. The number of intermediary accounts was fixed at $n_i = 30$, and the TX arrival rate varied within the range 1000, 1500, 2000, 2500, 3000, 3500 TXs/s. The results are shown in Figure 8. As the TX arrival rate increases, the throughput of all schemes shows a significant rise, but beyond a certain threshold, the throughput stabilizes. This is because when the TX arrival rate is low, all transactions can be included in the blocks. However, when the TX arrival rate exceeds the maximum block size, the number of transactions that can be packed is constrained by the block size. Compared to the other schemes, EquiFlowShard demonstrates superior throughput, highlighting its higher capacity to handle a large volume of transactions.

Furthermore, the TCL increases significantly with the rising TX arrival rate, also due to the block size limitation. However, it is evident that EquiFlowShard maintains a lower TCL compared to the other schemes, and the increase in TCL caused by higher TX arrival rates is relatively smaller.

In conclusion, across different TX arrival rates, EquiFlowShard consistently exhibits outstanding performance, with higher throughput, lower transaction latency, and a reduced proportion of cross-shard transactions compared to the other schemes.

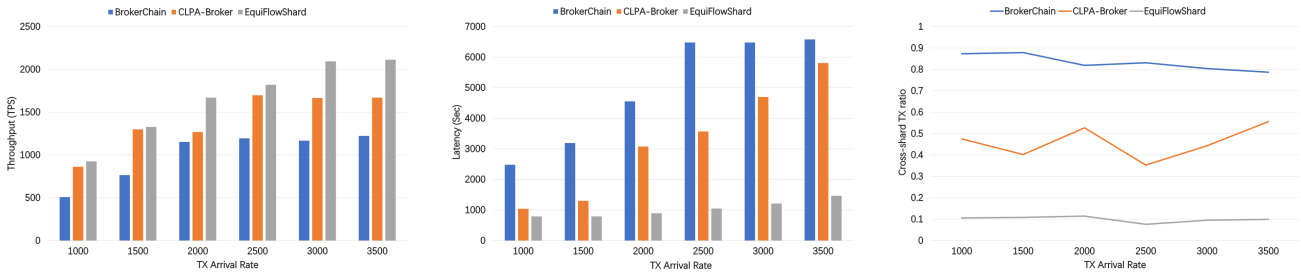


Figure 8. Effect of TX arrival rate. (**a**) Throughput. (**b**) TX confirmation latency (s). (**c**) Cross-shard TX ratio.

6. Discussion

6.1. System Security

EquiFlowShard follows the same configuration as related works [1,2,9,11], ensuring it achieves a comparable security level under the same settings. Each shard's consensus uses the PBFT protocol, which tolerates up to one-third of malicious nodes. Thus, the upper bound of system failure probability per epoch, $Pr(failure)$, can be determined using the cumulative hypergeometric distribution function, calculated as follows:

$$Pr(failure) \leq (K+1) \sum_{n_m=\lfloor n_s/3 \rfloor}^{n_s} \frac{\binom{f \cdot n}{n_m}\binom{(1-f)\cdot n}{n-n_m}}{\binom{n}{n_s}} \tag{7}$$

Let n represent the total number of consensus nodes with valid identities in each epoch and $K+1$ represent the number of shards (K P-shards and 1 O-shard). $n_s = n/(K+1)$ denotes the number of consensus nodes per shard, and $f \in [0,1]$ represents the proportion of malicious nodes, with n_m indicating the number of malicious nodes in a shard.

As shown in the formula, the system remains secure as long as the proportion of malicious nodes in each shard does not exceed one-third, i.e., $n_m < n_s/3$.

To demonstrate the system's security, the following measures are analyzed for resisting potential attacks: (1). Sybil attack: In EquiFlowShard, nodes participating in shard consensus must solve a PoW challenge in each round to obtain identity authentication, significantly increasing the cost for attackers. This serves as the primary defense against Sybil attacks. Achieving a malicious node count exceeding one-third of the total network nodes would require substantial computational power, rendering the attack economically infeasible. Additionally, EquiFlowShard reconfigures nodes at the end of every epoch, further reducing the likelihood of a successful attack. (2). Replay attack: To prevent replay attacks, all communication messages in EquiFlowShard include a unique nonce field to ensure message uniqueness and integrity. This mechanism effectively blocks attempts to reuse or duplicate messages. (3). Bribery attack: EquiFlowShard addresses bribery attacks by reconfiguring shard nodes at the end of each epoch. This frequent and random reassignment of nodes reduces the window of opportunity for bribery and increases its cost and complexity. In the future, we plan to adopt a reputation-based node reconfiguration mechanism to further enhance security. Additionally, the system imposes strict penalties on bribery attempts, including confiscating collateral and permanently removing malicious nodes from the network.

6.2. LinkAgent Security

First, all nodes wishing to become LinkAgents must obtain their identity through PoW in order to be selected. Therefore, to increase the chances of becoming a LinkAgent by creating multiple nodes, a significant amount of hashing power is required.

Second, the selection of LinkAgents is performed by the O-shard, with final consensus reached through the PBFT protocol. As long as the proportion of malicious nodes in the O-shard does not exceed one-third, the security of LinkAgent selection is guaranteed. This paper employs a configuration similar to that of related work [11], indicating that conducting a Sybil attack on the O-shard would likewise demand significant hashing power from malicious nodes.

It is worth noting that LinkAgent selection is primarily based on account transaction volume, allowing malicious nodes to increase their likelihood of selection by initiating numerous small transactions in a short period. This tactic can be countered by implementing transaction analysis at the application layer to identify these malicious-signature nodes. The O-shard blacklists these nodes, excludes them from LinkAgent selection, and prevents consensus on state blocks containing them. Additionally, the system confiscates tokens previously staked by these malicious nodes to qualify as LinkAgents.

Lastly, even if a malicious node becomes a LinkAgent, security remains assured. Malicious actions by a LinkAgent can occur in three scenarios: (1) during state migration, where the LinkAgent may forge or tamper with messages or perform replay attacks; (2) while handling cross-shard transactions, where the LinkAgent may alter transaction messages or withhold the sender's tokens, failing to process the subsequent parts of the transaction; (3) under high congestion, where the LinkAgent may degrade system performance by performing malicious actions.

For the first scenario, as described in Section 4.2, the state reconfiguration messages include a hash digest of the message content to prevent tampering. Additionally, the messages incorporate a nonce field, which increments with each transmission. Replay attacks are detected as repeated nonces are flagged.

For the second scenario, the way LinkAgents handle cross-shard transactions is similar to the broker in [9], where the original transaction Tx_{raw} is split into two parts: Tx_1 (the first part) and Tx_2 (the second part). All accounts, including LinkAgents, have an incrementing nonce field. This field is included in transactions, and after a transaction is sent, the nonce is incremented by 1. Combined with the token locking mechanism, this prevents double-spending attacks. EquiFlowShard also monitors nonce updates; if a malicious LinkAgent attempts a double-spending or replay attack, the cross-shard transaction will be forced into failure, rolling back the transaction and returning the tokens to the sender.

For the third scenario, under high congestion situation, LinkAgents may engage in the following three types of malicious behavior:

(a). Sending meaningless requests: Malicious LinkAgents may generate a large number of invalid cross-shard transaction requests, increasing inter-shard communication overhead. This can cause transaction queues in the pools to grow rapidly, leading to system performance degradation or even congestion. To address this, the system can employ real-time monitoring to identify accounts generating frequent invalid requests and impose penalties, such as revoking LinkAgent status, confiscating collateral, or expelling the account. Additionally, request frequency limits can be established to flag and penalize accounts exceeding thresholds. Nodes can further prioritize economically valuable transactions when generating blocks, effectively filtering out invalid requests to optimize system throughput.

(b). Delay attacks: Malicious LinkAgents may intentionally delay processing cross-shard transaction requests during periods of high congestion, creating cascading delays that

increase transaction confirmation times and degrade user experience. To mitigate this, the system can dynamically adjust the distribution of LinkAgents, introducing additional agents to alleviate congestion. Real-time monitoring of transaction processing times can identify and penalize LinkAgents causing abnormal delays. However, defining and adapting delay thresholds dynamically based on network conditions remains challenging, and future work will explore effective solutions in this area.

(c). Malicious exit or offline behavior: Malicious LinkAgents may intentionally exit or go offline at critical moments, disrupting the completion of cross-shard transactions. As previously explained, even if a LinkAgent exits or goes offline, token lock mechanisms and timeout rollback mechanisms ensure the atomicity of cross-shard transactions. However, such behavior increases the system's resource burden for handling rollbacks and amplifies the load on other LinkAgents, further degrading performance under high congestion. To counteract this, the system can pre-configure backup global LinkAgents to automatically take over transaction processing when primary LinkAgents fail. Furthermore, designing improved cross-shard transaction mechanisms to penalize malicious LinkAgents by confiscating their collateral while completing transactions without triggering rollbacks will be prioritized in future research. These measures aim to enhance the robustness and reliability of the sharding system under adverse scenarios.

6.3. Future Prospects

First, a proper incentive mechanism should be designed. In EquiFlowShard, LinkAgents take on the additional task of processing cross-shard transactions. Without an effective incentive system, accounts may lack the motivation to become LinkAgents. The Broker2Earn protocol [25] has already implemented an incentive scheme that maximizes intermediary rewards.

Second, the setup of LinkAgents should be decoupled from account segmentation. Most hot accounts can be segmented, but only a small portion of them will be selected as LinkAgents. The primary purpose of LinkAgents is to ensure the atomicity and security of cross-shard transactions, with limited impact on blockchain throughput. By segmenting hot accounts, the proportion of cross-shard transactions can be effectively reduced, balancing the workload between shards and improving overall throughput.

Third, we plan to extend the EquiFlowShard protocol to optimize the allocation of smart contract accounts, addressing their inherent complexity and operational constraints. Unlike regular accounts, smart contract accounts often involve multiple interconnected transactions and dependencies, requiring precise execution order and dependency resolution. This complexity makes direct application of community detection algorithms insufficient. To address these challenges, we may explore integrating a dependency-aware scheduling mechanism into EquiFlowShard, which could accommodate the execution order of contract calls. Additionally, we might design state allocation strategies that potentially cluster interdependent contracts within the same shard, thereby reducing cross-shard interactions. These efforts could enhance the protocol's adaptability to the dynamic and complex nature of smart contract operations, broadening its potential applications in real-world blockchain scenarios.

Lastly, future efforts may focus on integrating EquiFlowShard into existing blockchain networks and adapting it for IoT environments, with an emphasis on optimizing efficient state management in dynamic settings. Additionally, a comprehensive security analysis will be conducted to ensure its robustness against real-world attack scenarios.

7. Conclusions

This paper presents EquiFlowShard, a novel sharding protocol that addresses scalability and performance challenges in blockchain systems. By employing the OSADA, EquiFlowShard dynamically optimizes account state distribution to balance workloads and minimize cross-shard transactions. The SFlow mechanism ensures secure and consistent state management, while the Smooth Transition scheme reduces the performance impact of state reconfiguration. Experimental results, based on Ethereum transaction data, demonstrate significant improvements in throughput, transaction confirmation latency, and cross-shard transaction ratios, outperforming existing protocols.

Beyond its technical contributions, EquiFlowShard holds promising potential for real-world adoption, particularly in dynamic blockchain environments such as the IoT. Its ability to adapt to high-volume and heterogeneous transaction patterns makes it a strong candidate for scalable blockchain applications in industries requiring robust data handling and real-time operations.

Future research will prioritize several areas to expand EquiFlowShard's capabilities: (1). Enhanced adaptability: Developing intelligent mechanisms for LinkAgent selection and workload allocation to address dynamic changes in transaction patterns. (2). Diversified validation: Evaluating the protocol using larger, more varied datasets to further validate its robustness and generalizability in diverse blockchain use cases. (3). IoT integration: Exploring seamless integration with IoT-based blockchain applications, focusing on efficient data processing and scalable consensus mechanisms.

Author Contributions: Conceptualization, Y.C.; methodology, Y.C.; software, Y.C.; validation, Y.C.; formal analysis, Y.C.; investigation, Y.C.; resources, Y.C.; data curation, Y.C.; writing—original draft preparation, Y.C.; writing—review and editing, Y.W. and T.L.; visualization, Y.C.; supervision, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Luu, L.; Narayanan, V.; Zheng, C.; Baweja, K.; Gilbert, S.; Saxena, P. A secure sharding protocol for open blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 17–30.
2. Zamani, M.; Movahedi, M.; Raykova, M. Rapidchain: Scaling blockchain via full sharding. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 931–948.
3. Kokoris-Kogias, E.; Jovanovic, P.; Gasser, L.; Gailly, N.; Syta, E.; Ford, B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–23 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 583–598.
4. Nguyen, L.N.; Nguyen, T.D.; Dinh, T.N.; Thai, M.T. Optchain: Optimal transactions placement for scalable blockchain sharding. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–9 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 525–535.
5. Wang, J.; Wang, H. Monoxide: Scale out blockchains with asynchronous consensus zones. In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), Boston, MA, USA, 26–28 February 2019; pp. 95–112.
6. Hong, Z.; Guo, S.; Li, P.; Chen, W. Pyramid: A layered sharding blockchain system. In Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications, Virtual, 10–13 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–10.

7. Li, M.; Lin, Y.; Zhang, J.; Wang, W. Jenga: Orchestrating smart contracts in sharding-based blockchain for efficient processing. In Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Bologna, Italy, 10–13 July 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 133–143.
8. Okanami, N.; Nakamura, R.; Nishide, T. Load balancing for sharded blockchains. In Proceedings of the Financial Cryptography and Data Security: FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, 14 February 2020; Revised Selected Papers 24; Springer: Berlin/Heidelberg, Germany, 2020; pp. 512–524.
9. Huang, H.; Peng, X.; Zhan, J.; Zhang, S.; Lin, Y.; Zheng, Z.; Guo, S. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, Virtual, 2–5 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1968–1977.
10. Li, M.; Wang, W.; Zhang, J. LB-Chain: Load-balanced and low-latency blockchain sharding via account migration. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 2797–2810.
11. Li, C.; Huang, H.; Zhao, Y.; Peng, X.; Yang, R.; Zheng, Z.; Guo, S. Achieving scalability and load balance across blockchain shards for state sharding. In Proceedings of the 2022 41st International Symposium on Reliable Distributed Systems (SRDS), Vienna, Austria, 19–22 September 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 284–294.
12. Al-Bassam, M.; Sonnino, A.; Bano, S.; Hrycyszyn, D.; Danezis, G. Chainspace: A sharded smart contracts platform. *arXiv* **2017**, arXiv:1708.03778.
13. Bagaria, V.; Kannan, S.; Tse, D.; Fanti, G.; Viswanath, P. Prism: Deconstructing the blockchain to approach physical limits. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 585–602.
14. Tao, Y.; Li, B.; Jiang, J.; Ng, H.C.; Wang, C.; Li, B. On sharding open blockchains with smart contracts. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1357–1368.
15. Jia, L.; Liu, Y.; Wang, K.; Sun, Y. Estuary: A Low Cross-Shard Blockchain Sharding Protocol Based on State Splitting. *IEEE Trans. Parallel Distrib. Syst.* **2024**, *35*, 405–420.
16. Jiang, S.; Cao, J.; Tung, C.L.; Wang, Y.; Wang, S. Sharon: Secure and Efficient Cross-shard Transaction Processing via Shard Rotation. In Proceedings of the IEEE INFOCOM 2024—IEEE Computer Communications, Vancouver, BC, Canada, 20–23 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 2418–2427.
17. Li, P.; Song, M.; Xing, M.; Xiao, Z.; Ding, Q.; Guan, S.; Long, J. SPRING: Improving the Throughput of Sharding Blockchain via Deep Reinforcement Learning Based State Placement. In Proceedings of the ACM on Web Conference 2024, Virtual, 13–17 May 2024; pp. 2836–2846.
18. Castro, M.; Liskov, B.; et al. Practical byzantine fault tolerance. In Proceedings of the OsDI, New Orleans, LA, USA, 22–25 February 1999; Volume 99; pp. 173–186.
19. Zhang, S.; Lee, J.H. Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE Trans. Ind. Inform.* **2019**, *15*, 5715–5722.
20. Puttaswamy, K.P.; Zheng, H.; Zhao, B.Y. Securing structured overlays against identity attacks. *IEEE Trans. Parallel Distrib. Syst.* **2008**, *20*, 1487–1498.
21. Awerbuch, B.; Scheideler, C. Towards a scalable and robust DHT. In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, Cambridge, MA, USA, 30 July–2 August 2006; pp. 318–327.
22. Xie, J.; Szymanski, B.K.; Liu, X. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops, Vancouver, BC, Canada, 11 December 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 344–349.
23. Huang, H.; Ye, G.; Chen, Q.; Yin, Z.; Luo, X.; Lin, J.; Yang, Q.; Zheng, Z. Blockemulator: An emulator enabling to test blockchain sharding protocols. *arXiv* **2023**, arXiv:2311.03612.
24. Wood, G.; et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
25. Chen, Q.; Huang, H.; Yin, Z.; Ye, G.; Yang, Q. Broker2Earn: Towards Maximizing Broker Revenue and System Liquidity for Sharded Blockchains. In Proceedings of the IEEE INFOCOM 2024-IEEE Conference on Computer Communications, Vancouver, BC, Canada, 20–23 May 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 251–260.