Computer Standards & Interfaces 97 (2026) 104123

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

V-Bridge: A dynamic cross-shard blockchain protocol based on off-chain payment channel

Xueting Huang [a], Xiangwei Meng [a,c,b], Kai Zhang [a,c,b], Ce Yang [a,c,b], Wei Liang [a,c,b,*], Kuan-Ching Li [a,c,b,*]

[a] College of Computer Science and Engineering, Hunan University of Science and Technology, XiangTan 411201, China
[b] Hunan University of Science and Technology Sanya Research Institute, SanYa 572000, China
[c] Hunan Key Laboratory for Service Computing and Novel Software Technology, XiangTan 411201, China

ARTICLE INFO

Keywords:
Blockchain
Sharding
Cross-shard
Off-chain payment channel

ABSTRACT

Sharding technology effectively improves system throughput by distributing the blockchain transaction load to multiple shards for parallel processing, and it is the core solution to the scalability problem of blockchain. However, as the number of shards increases, the frequency of cross-shard transactions increases significantly, leading to increased communication and computational overhead, transaction delays, uneven resource allocation, and load imbalance, which becomes a key bottleneck for performance expansion. To this end, this article proposes the cross-shard transaction protocol V-Bridge, which draws on the concept of off-chain payment channels to establish distributed virtual fund channels between Trustors in different shards, convert cross-shard transactions into off-chain transactions and realize the logical flow of funds. To further enhance cross-shard transaction performance, our V-Bridge integrates an intelligent sharding adjustment mechanism, and a cross-shard optimized critical path protection algorithm (CSOCPPA) to dynamically balance shard loads, alleviate resource allocation issues, and minimize performance bottlenecks. Experimental results show that compared with existing state-of-the-art protocols, our proposed V-Bridge's average throughput is increased by 26% to 46%, and transaction delays are reduced by 15% to 24%.

1. Introduction

Blockchain [1,2], as a decentralized, transparent, and tamper-proof technology, has great potential in various fields such as privacy protection, medical applications, and the Internet of Things [3–5]. In cross-border payments, for example, it offers an alternative to traditional banking systems, enabling fast and cost-effective transactions [6]. However, blockchain systems face significant scalability challenges during high transaction volumes [7–9]. For instance, Ethereum [10] often experiences network congestion during peak usage, leading to transaction delays and increased GAS fees. Addressing scalability has become a critical priority. Sharding [11,12] technology offers a promising solution by partitioning the blockchain network into independent shards, enabling parallel transaction processing and faster confirmations [13]. However, the benefits of sharding are hindered by challenges posed by cross-shard transactions. These transactions require data synchronization and state validation across shards, which increases communication overhead, introduces complex synchronization issues, and creates potential performance bottlenecks—diminishing the efficiency gains of sharding [14].

Various cross-shard transaction protocols have been proposed to address these challenges to enhance processing efficiency and reduce synchronization overhead. For instance, Monoxide [15] employs an asynchronous consensus mechanism to minimize inter-shard waiting times. However, while transactions are processed in parallel, the system struggles to ensure state consistency and communication overhead remains significant in large-scale sharding environments. BrokerChain [16] introduces brokers to coordinate cross-shard transaction processing, but its approach appears impractical in real-world applications. The system relies heavily on securing sufficient intermediary nodes to stabilize cross-shard transactions. However, such nodes are challenging to acquire in decentralized networks, especially under heavy transaction loads. This reliance amplifies dependence on individual nodes, increasing the risk of centralization and limiting scalability. Consequently, BrokerChain [16] struggles to address complex scenarios

* Correspondence to: College of Computer Science and Engineering, Hunan University of Science and Technology, No. 2 Taoyuan Road, Yuhu District, Xiangtan 411201, Hunan Province, China.
E-mail addresses: wliang@hnust.edu.cn (W. Liang), aliric@hnust.edu.cn (K.-C. Li).

https://doi.org/10.1016/j.csi.2025.104123
Received 17 December 2024; Received in revised form 27 October 2025; Accepted 26 December 2025
Available online 31 December 2025

involving dynamic node allocation and shard optimization, exposing key weaknesses in its design's practicality and adaptability. To address shard distribution, BrokerChain [16] employs *epoch*, where the Metis algorithm [17] optimizes user account partitioning to prevent related transactions from being spread across shards in subsequent *epochs*. This strategy is similarly adopted in protocols like X-Shard [18]. However, in practice, the Metis algorithm [17] suffers from partition imbalance and fragmentation of critical nodes. This fragmentation can distribute high-frequency transaction paths across multiple shards, increasing cross-shard communication overhead and processing latency. Ultimately, this undermines system performance and scalability. In conclusion, significant challenges remain while both BrokerChain [16] and X-Shard [18] offer innovative approaches to cross-shard transaction optimization. Improvements are needed in node allocation, security management, and performance optimization to realize the full potential of cross-shard systems.

To address the shortcomings of existing cross-shard transaction protocols, we propose the V-Bridge, a novel solution leveraging the off-chain transaction model of bidirectional payment channels. The V-Bridge facilitates logical fund interactions across shards by constructing virtual channels. To reduce reliance on single nodes in decentralized environments, the V-Bridge introduces trustee groups as relay nodes within each shard. These groups, supported by a flexible and robust management framework, ensure seamless cross-shard transactions while distributing the load effectively. Virtual fund channels between trustees are settled on-chain only upon closure, which significantly reduces on-chain synchronization overhead and enhances transaction efficiency. Additionally, the V-Bridge incorporates an intelligent shard adjustment mechanism based on a Consistent Hashing Ring [19,20] and GINI coefficients [21], integrated with the Cross-Shard Optimized Critical Path Protection Algorithm (CSOCPPA). This combination improves the coordination of dynamic shard adjustments and cross-shard transactions, further enhancing system performance.

The main contributions of this article are summarized as follows:

- *Blockchain Sharding Protocol Based on Virtual Fund Payment Channels (V-Bridge):* We propose a virtual channel solution inspired by off-chain payment channels to facilitate cross-shard transactions. This approach minimizes direct interactions between shards, significantly reducing communication overhead and transaction delays. Consequently, V-Bridge enhances system throughput and overall performance.
- *Cross-shard Optimization and Dynamic Shard Adjustment Mechanism:* The V-Bridge Protocol integrates the Cross-Shard Optimized Critical Path Protection Algorithm (CSOCPPA) and a dynamic shard adjustment mechanism to improve system performance. CSOCPPA enhances the Metis algorithm [17] for better account allocation, reducing cross-shard assignments. The dynamic adjustment mechanism uses Consistent Hashing and GINI coefficient analysis to optimize shard splitting and merging for balanced load distribution.
- *System Implementation:* We implemented the V-Bridge protocol and evaluated its performance on Ubuntu 20.04.1. Experimental results demonstrate that, compared to BrokerChain and X-Shard, V-Bridge outperforms in terms of throughput, transaction confirmation latency, workload balancing, and consensus success rate under identical conditions.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents an overview of the V-Bridge system. Section 4 details the protocol design. Section 5 introduces cross-shard optimization and dynamic shard adjustment. Section 6 provides a security analysis. Section 7 reports experimental results, and Section 8 concludes the paper.

2. Related work

2.1. Cross-shard transaction

To address the challenges of cross-shard transactions, a variety of protocols have been developed [22–24], with a focus on enhancing performance, scalability, and consistency. Monoxide [15] uses an asynchronous consensus mechanism and temporary payment channels to decompose cross-shard transactions into intra-shard operations, boosting throughput. However, the complexity of state synchronization and consistency verification across shards increases communication overhead, impacting large-scale system performance. OmniLedger [25] employs the Atomix protocol to ensure the atomicity of cross-shard transactions. Through a client-driven two-phase commit process, it freezes the UTXO of the input shard and then releases the UTXO of the output shard, maintaining consistent transaction state updates. RapidChain [26] improves system throughput by reducing communication rounds through parallel verification. However, it struggles with performance bottlenecks when handling cross-shard transactions with long dependency chains due to prolonged waiting times. BrokerChain [16] simplifies cross-shard communication by introducing third-party intermediary nodes that convert cross-shard transactions into intra-shard transactions. However, this approach introduces risks related to decentralization and trust management. Pyramid [27] implements a hierarchical sharding protocol where BridgeShard processes transactions across multiple shards in a single consistency round, significantly reducing confirmation latency. However, it increases system management complexity. CHERUBIM [28] leverages pipeline processing based on the 2PC protocol to enhance cross-shard transaction throughput but falls short in mitigating long transaction latencies. Building on these advancements, we propose the V-Bridge protocol, offering an efficient and flexible solution to the challenges of cross-shard transactions. V-Bridge supports seamless operation in complex transaction scenarios, ensuring the efficient functioning of blockchain systems.

2.2. Payment channel

Payment channels are off-chain mechanisms that improve blockchain performance by optimizing transaction processing, reducing network load, delays, and fees. Users lock funds via smart contracts, enabling multiple off-chain transactions without recording each one on the blockchain. The mechanism defines rules for initial fund allocation and status updates. Users can update the status off-chain, with each update verified by signed certificates from both parties. When the channel closes, the final status is submitted to the blockchain to settle the fund distribution. By minimizing on-chain interactions, payment channels reduce blockchain resource use. A key feature is the dynamic adjustment of participant balances while maintaining total fund immutability. This improves transaction efficiency and ensures security. Payment channels leverage cryptographic tools like digital signatures to ensure transaction integrity and reduce the costs and delays of on-chain operations, especially during congestion.

In recent years, payment channels have been widely used in many fields, such as privacy protection [29], network scalability [30] and cross-chain interoperability [31], Internet of Things expansion [32] and other directions. In the sharding architecture of this article, the off-chain payment channel also provides an efficient solution for processing cross-shard transactions. By transferring transactions between shards to off-chain, payment channels effectively reduce the complexity of cross-shard communications, significantly improve the throughput and performance of the sharding system, and provide important support for the further development of sharding technology.

3. Overview

This section introduces the V-Bridge system model and its workflow, followed by the deployment process of Trustor.

3.1. System architecture and workflow

Similar to BrokerChain [16], V-Bridge is also executed in *epoch*, and V-Bridge includes two types of shards: settlement and account shards, of which there are S settlement shards and 1 status shard. The specific definitions are as follows:

- *Settle shard (S-Shard):* Generates transaction blocks by packaging transactions and achieves intra-shard consistency at the beginning of each *epoch*.
- *Account Shard (A-shard):* A-shard optimizes account allocation based on the user's transaction history data to alleviate the problem of load imbalance in the shard system. During the system startup phase, A-shard collects user transaction data from S-shard, uses this data to build a user transaction network, and optimizes user status distribution through the CSOCPPA. After the optimization, A-shard generates a status block and sends it to S-shard to update the user account status distribution.

Other Settings: We adopt a Byzantine fault-tolerant adversarial model [33], assuming the presence of malicious actors capable of compromising specific shard nodes and performing arbitrary dishonest actions, such as data tampering, delayed messaging, or refusing to submit required information. These adversaries are slow-adaptive, meaning they cannot frequently rotate the compromised nodes within a single *epoch*. The system assumes a partially synchronous communication model, where messages may experience delays but are guaranteed eventual delivery.

Building upon this adversarial model, the V-Bridge mechanism is designed to ensure resilience against such malicious actors. The system architecture is structured into three key layers: the Network Initialization Layer, the Account State Reconstruction Layer, and the Transaction Processing and Consensus Layer.

- *Network Initialization Layer (NIL):* This layer is the core foundation of V-Bridge and is responsible for the reasonable sharding of nodes and transaction loads in the system to form multiple independent sharding structures.
- *Transaction Processing and Consensus Layer (TPCL):* According to the distribution of accounts and transactions, this layer optimizes and reconstructs the account status within the shard through the CSOCPPA, generates a dynamic state reconfiguration plan, and improves system performance.
- *Account State Reconstruction Layer (ASRL):* This layer is responsible for transaction verification and consensus, ensuring the security and consistency of all transactions.

Based on the above architecture, each layer works together to realize each *epoch* process from node identity authentication to transaction processing. As shown in Fig. 1, the workflow can be divided into the following five steps:

Step1 PoW verification: Nodes verify their identity using PoW [34] (via public key and IP) to prevent Sybil attacks. Verified nodes are evenly distributed across shards in a round-robin fashion.

Step2 Select Trustor: After shard assignment, nodes apply to serve as Trustors in this *epoch*. Each shard forms a trust group to support subsequent transactions.

Step3 Transaction consensus: The settlement shard validates transactions and adds them to the pool. Consensus is reached via PBFT [33]. For cross-shard cases, virtual channels (Section 4.2) manage interactions; failed verifications trigger rollback (Section 4.3).

Step4 State optimization: The CSOCPPA algorithm (Section 5.1) adjusts account placement based on access patterns. A consensus-derived state block is distributed to shards for the next *epoch*.

Step5 *Epoch* sharding: In the new *epoch*, PBFT [33] guides account migration and transaction routing based on the latest state diagram, ensuring consistency and performance.

3.2. Trustor deployment

In our V-Bridge, we introduce the concept of the Trustor. Before delving into the specifics of the Trustor, it is essential to understand the concept of a trust group. A trust group consists of nodes selected by the system to facilitate the transfer of funds across shards. These nodes provide liquidity through staking or contributing resources like funds or computing power, creating a bridge for cross-shard transactions. In return, they earn commissions as incentives. By ensuring sufficient liquidity, these nodes establish virtual transaction channels between shards, enabling successful cross-shard payments for users both inside and outside the shards. The node performing this critical role is the Trustor. The process for participating in system transactions involves the following key steps:

Step1 Trustor application: At the start of each *epoch*, nodes from various shards may apply to serve as Trustors by submitting collateral, undergoing a credit assessment (excluding low-reputation nodes with poor historical performance), and providing proof of sufficient liquidity to meet the minimum funding threshold.

Step2 Qualification verification: The system assesses applicant nodes based on their collateral, computational capacity, credit score, and liquidity. Each node is assigned an initial credit score that reflects its resources and historical performance. The system prioritizes selecting Trustors from nodes with higher credit scores. However, to mitigate the risk of excessive centralization, a probabilistic selection mechanism is employed. Specifically, high-reputation nodes (Top 30%) are assigned a 60% probability of selection, while medium-reputation nodes (Top 30%–60%) are allocated a 40% probability. Even if there are enough high-reputation candidates, there remains a 40% chance of selecting a medium-reputation node, thus distributing power within the system and reducing centralization risks. In the event that there are insufficient qualified candidates, the system will randomly select additional well-performing nodes to supplement the trust set. The trusted set is dynamically maintained: nodes with significantly reduced liquidity or credit scores are automatically removed.

Step3 Leader election: For each shard's trust set, the system designates the Trustor node with the highest credit score as the leader, with the leader's term being tied to the current *epoch*. The leader is not allowed to serve consecutive terms in two consecutive *epochs*. Each Trustor generates a unique identifier T_{id}, defined as:

$$T_{\text{id}} = \text{hash}(\text{ShardID} \parallel \text{Rep} \parallel \text{Deposit} \parallel \text{Value} \parallel \text{Relate})$$

where Rep is the credit score, Deposit is the collateral, Value is the remaining balance, Relate indicates the cross-shard relationships.

The Relate variable is defined as:

$$\text{Relate} = \begin{cases} 0 & \text{if no channel with other shards;} \\ \text{ShardID} & \text{if a channel with another shard exists.} \end{cases}$$

Each T_{id} is stored within the shard. The leader can query these T_{id} values to execute cross-shard transactions and select the most suitable executor from the trusted set.

4. V-Bridge protocol design

In this section, we introduced the core modules of the V-Brige protocol, including the new Merkle tree, the solution of cross-shard transactions based on virtual payment channels, and the final transaction settlement.

4.1. New Merkle Patricia Tree

To support efficient user state queries and cross-shard transaction routing, we design a shard management framework that combines a Consistent Hashing Ring with a New Merkle Patricia Tree (NMPT) (see Fig. 2).

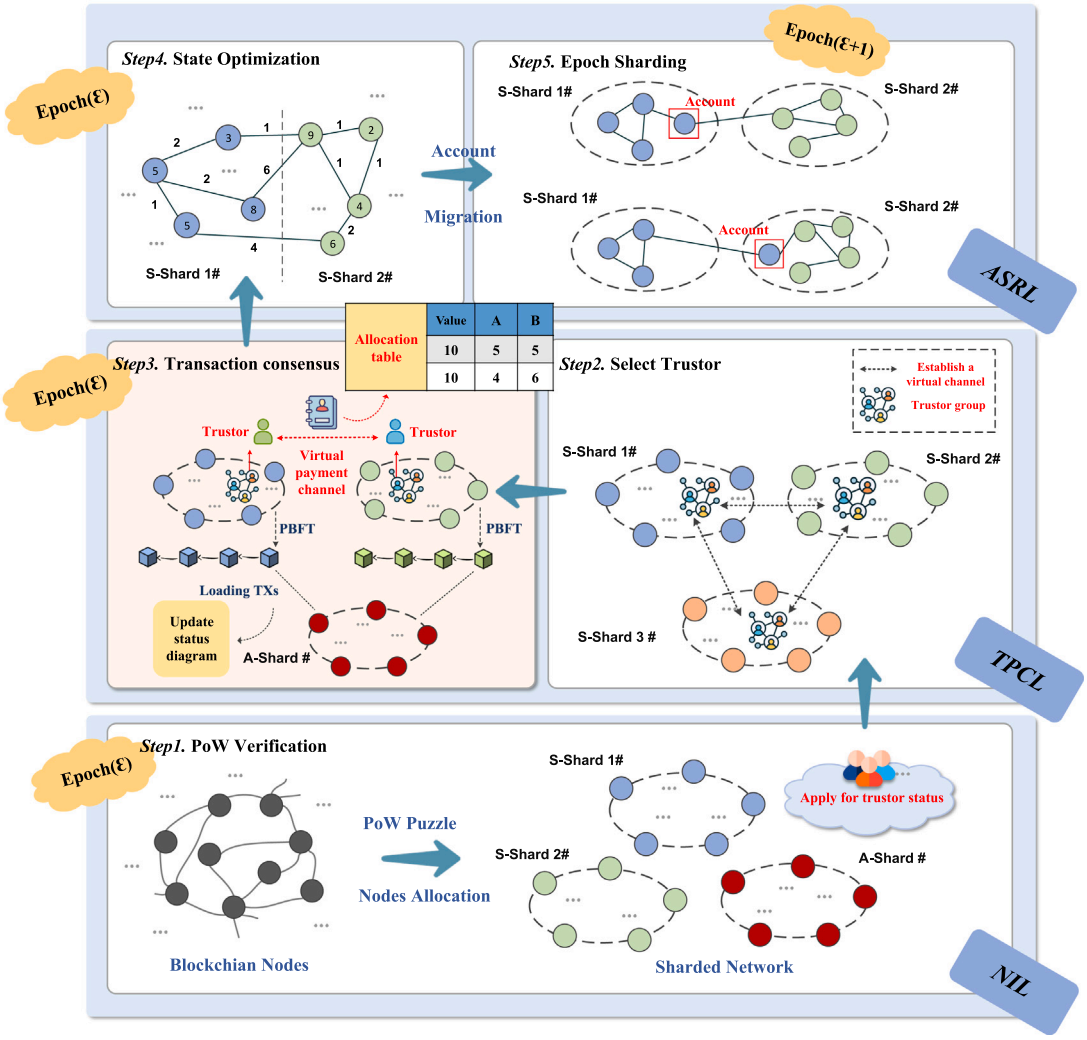

Fig. 1. Workflow diagram of V-Bridge for an *Epoch*.

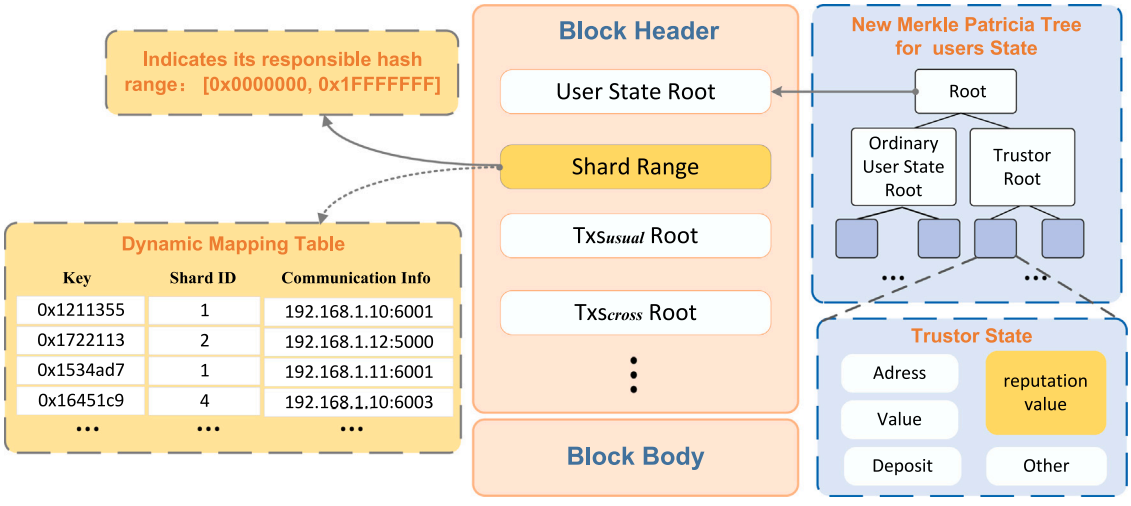

Fig. 2. The data structure and mapping of NMPT.

When a user initiates a query or transaction request that requires locating their associated shard, the system first computes the user's hash value H_u = Hash(ID$_u$) using the SHA-256 algorithm [35]. It then performs a clockwise search on the consistent hash ring [19] to identify the first shard whose assigned range contains H_u, thereby determining the user's query shard. Each shard maintains a uniquely assigned hash interval and dynamically records the user–shard mappings within that range to support efficient user lookup and cross-shard routing.

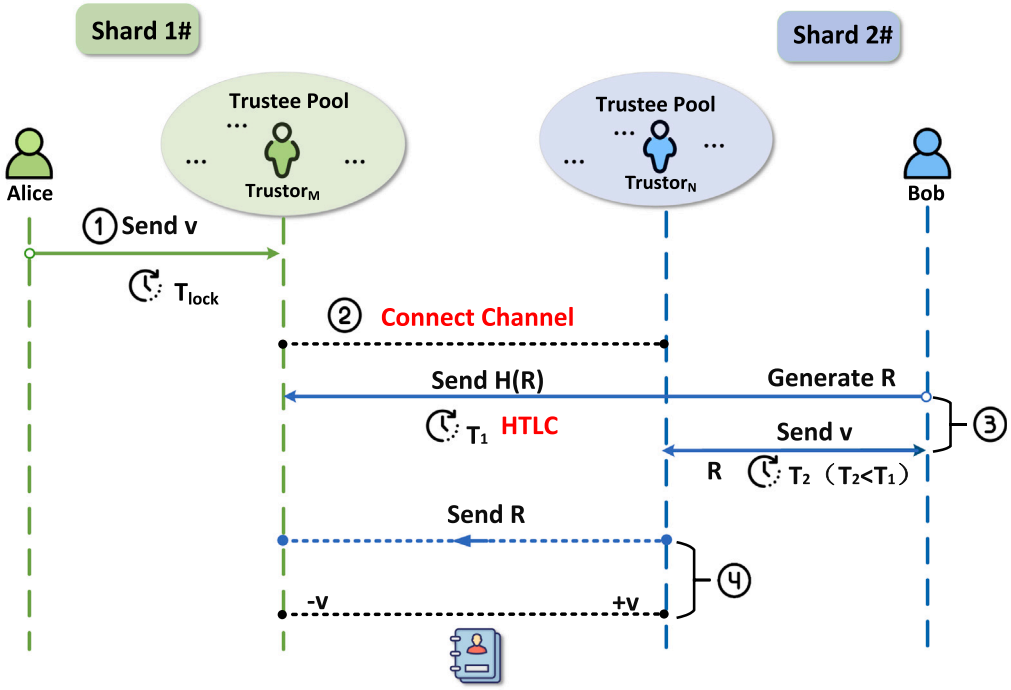

Fig. 3. Example of cross-shard Tx processing.

The NMPT serves as the core of state storage and improves on the traditional Merkle Patricia Tree by:

- Separating ordinary and Trustor user states into independent branches;
- Embedding a Shard Range module to explicitly encode shard positions on the hash ring;
- Enabling parallel updates, fast redirection, and reduced state conflicts.

These enhancements ensure consistency and rapid verification in dynamic, cross-shard environments. The usage of this framework in transaction execution is elaborated in the next section.

4.2. The specific implementation process of V-brige protocol

We establish a channel contract (ChannelContract) between two shards via a Trustor, converting most transactions into intra-shard operations. Fund transfers between channel participants occur off-chain, with on-chain interactions limited to channel creation and final settlement. This approach significantly reduces the on-chain overhead of cross-shard transactions and enhances system performance. The protocol involves four main steps: Initial Fund Locking, Trustor Coordination and Locking, $Trustor_N$ Payment Execution, and HTLC (Hashed Timelock Contract) Unlocking [36,37]. To illustrate the protocol's operation, we present a cross-shard transaction case that demonstrates how it ensures the seamless completion of cross-shard transactions (see Fig. 3). Alice, a user in Shard1, wants to initiate a cross-shard transaction to transfer an amount v to Bob, a user in Shard2. The process unfolds as follows:

Step1 Initial fund locking: First, Alice must locate Bob's shard position through the Hash Ring. Therefore, Alice will create a message containing the transaction information, structured as follows:

$$TX_{\text{request}} = \{\text{Property}_A, \text{ToUser}_B, \text{Time}_A, v, T_{\text{lock}}, \text{Sig}_A\}$$

where T_{lock} is the preset lock time for the funds, Time_A is the timestamp for creating the transaction, and v represents the transaction amount. This message is signed by Alice with Sig_A and sent to the leader of the trust group in her shard, denoted as M_{leader}.

Upon receiving the message, the leader verifies whether Alice's balance is sufficient to cover the transaction fee and ensures no duplicate transactions with the same timestamp, thereby preventing double-spending. Once verification is complete, the leader selects $Trustor_M$ with sufficient financial reserves based on the transaction requirements and then sends the verification results and the T_{id} information of the $Trustor_M$ responsible for this transaction to other Trustor nodes for confirmation. Suppose 2/3 of the Trustors agree to the transaction. In that case, the leader will record the transaction, send $Trustor_M$'s information to Alice for confirmation, and then $Trustor_M$ will begin processing the transaction.

Step2 Trustor coordination and locking: When the $Trustor_M$ receives the transaction request TX_{request}, it extracts the recipient Bob's user ID and uses a Hash Ring to locate his shard (e.g., Shard3). It then generates and sends a query message:

$$Q_M = \{\{TX_{\text{request}}\}, \text{Inform}, \text{Sig}_M\}$$

Upon receiving Q_M, Shard3 verifies the transaction and retrieves Alice's account by checking her signature. It consults the Dynamic Mapping Table to determine the shard locations of both Alice and Bob (e.g., Shard1 and Shard2), then forwards the query to Shard2 and broadcasts the involved shard locations.

Afterward, TX_{request} is submitted to Shard1 for verification. Simultaneously, the leader node N_{leader} in Shard2 analyzes the transaction and trust-related metrics to select a $Trustor_N$ to handle execution. This proactive selection validates the transaction without waiting for Shard1's response, reducing delay.

The selected $Trustor_N$ in Shard2 extracts $Trustor_M$'s address from Sig_M and deploys a contract C_B, which locks the specified funds. The contract enforces mutual agreement between sender's and receiver's Trustor to release funds. If consensus is not reached, the funds remain locked. C_B also includes timeout and fallback mechanisms to ensure progress under adverse conditions. Once completed, the verifier in Shard2 broadcasts the result to confirm transaction integrity.

In parallel, upon confirmation of TX_{request} in Shard1, $Trustor_M$ creates another contract C_A and locks funds under rules similar to C_B.

Table 1
Symbol explanations for algorithms.

Symbol	Explanation
G	Graph structure, containing nodes (accounts) and edges (transactions).
u, v	Account nodes in the graph, representing any two accounts.
a, b	Account nodes involved in matching or merging operations.
degree(a)	The degree of account a, representing the number of neighbors (transactions) it has.
maxDepth	Maximum depth of a node, used to prioritize high-frequency trading nodes.
neighbor(a)	The neighboring accounts of account a, connected in the graph.
edge(u, v)	Edge between accounts u and v, representing the transaction connection.
edge_weight(u, v)	Weight of the edge between accounts u and v.
target_size	Target size for the coarsened network, representing the desired scale.
threshold	Threshold value for merging or retaining nodes based on transaction volume.
region	Region formed by the depth-priority growing algorithm, containing multiple accounts.
sorted_accounts	List of accounts sorted by degree (transaction volume).

When C_A is finalized, its details are shared with Shard2 for synchronization.

Alice then sends the funds v and a message ζ to C_A, where $\zeta = \{v, \text{Time}_{\text{now}}, \text{ToUser}_B, T_{\text{lock}}\}$. The contract's unlocking condition requires that the corresponding transfer occurs within the time window T_{lock}, avoiding conflicts or double-spending. This ensures that the locked funds are correctly routed to $Trustor_M$.

Step3 $Trustor_N$ payment execution: In Shard2, after the contract is established, the $Trustor_N$ creates an intra-shard transaction message as follows:

$$TX_{\text{second}} = \{\text{Property}_N, \text{toUser}_B, \text{Time}_N, v, \text{Sig}_N\}$$

This transaction is sent to the verifier nodes within the shard for validation. Meanwhile, Bob generates a random number R and creates the following message:

$$\zeta' = \{\text{H}(R), \text{Sig}_B, \{TX_{\text{second}}\}\}$$

This message is sent to $Trustor_M$. Upon receiving the message, the $Trustor_M$ forwards H(R) to contract C_A, which automatically initiates the HTLC [36]. According to the rules of C_A, only when $Trustor_M$ provides the value of R corresponding to H(R) within the specified time T_1 ($T_1 < T_{\text{lock}}$), $Trustor_M$ can start the next transaction smoothly. Otherwise, when the time T_{lock} for Alice to lock the funds ends, the funds will return to Alice's account, and the system will start to query the initiator of the transaction failure and punish him (Section 4.3).

Once TX_{second} is included in the block, $Trustor_N$ will transfer the funds $s = v$ to contract C_B, locking them for the time period T_2 ($T_2 < T_1$). Bob is then notified that the funds are ready to be claimed. Once Bob provides the correct R within the specified time window, the locked funds will be released to Bob.

Step4 HTLC unlocking: The $Trustor_N$ completes the payment, yet the funds remain locked unless the process is correctly finalized within the designated time window T_2. To initiate the release, Bob must submit the correct value of R along with his digital signature, which enables public verification. Upon receiving and verifying R, $Trustor_N$ generates a channel message:

$$\theta_1 = \{\text{Sig}_N, \text{Table}_{\text{now}}, R\}$$

Here, $\text{Table}_{\text{now}}$ represents the latest balance allocation table, which is then sent to the $Trustor_M$. The $Trustor_M$ provides$R$ to the contract C_A for verification. If H(R) matches, the contract notifies $Trustor_M$ that the match is successful. At this time, $Trustor_M$ forwards the message to other idle Trustors in shard1 to jointly verify the allocation table and version number. If 2/3 of the Trustors verify that it is correct, $Trustor_M$ will generate the following channel message:

$$\theta_2 = \{\text{Sig}_M, \{\theta_1\}\}$$

and sends it back to $Trustor_N$. The idle Trustor group in Shard2 verifies the updated balance table and ensures that the latest channel states are properly recorded. Through this coordinated process, the updated balances are securely synchronized, and the final signed states preserve the integrity and consistency of the transaction.

4.3. Transaction settlement and failure handling

The transaction settlement process follows a standard payment channel model, addressing both cooperative and exceptional scenarios.

In the cooperative case, $Trustor_M$ and $Trustor_N$ mutually agree to close the channel. $Trustor_M$ generates and signs a closure message containing the final balance and fund allocation, then forwards it to $Trustor_N$ for verification and co-signature. The jointly signed message is broadcast to the blockchain, where Shard 1 and Shard 2 verify the signatures and release the allocated funds. If residual funds remain locked, the inter-shard smart contract redistributes them to the appropriate addresses, finalizing the settlement.

In the uncooperative case, $Trustor_M$ may unilaterally broadcast the most recent transaction state, initiating a challenge period during which $Trustor_N$ can dispute outdated information. If no challenge is made, the transaction is settled according to the submitted state.

In the event of abnormal failures, recovery mechanisms safeguard both fund security and system liveness. If $Trustor_N$ maliciously withholds the required secret R, and C_A fails to receive it within the timeout, a rollback is triggered: funds are refunded to $Trustor_M$, $Trustor_N$ forfeits their deposit, and suffers a reputation penalty. Repeated offenses may result in disqualification.

If failure occurs due to force majeure, $Trustor_N$ submits a failure proof

$$\mu = \{\text{Sig}_N, TX_{\text{request}}, TX_{\text{second}}, \text{Table}_{\text{now}}\}$$

to Shard 1 and Shard 2. Once validated that the transfer could not be completed within the lock time T_1, the locked funds in C_B are refunded to $Trustor_N$, and Shard 1 returns funds to $Trustor_M$, ensuring proper recovery and termination (see Table 1).

5. Cross-shard optimization and dynamic shard adjustment mechanism

5.1. CSOCPPA

Traditional Metis algorithms [17] coarsen transaction graphs using random or edge-weighted matching but often overlook critical paths and high-transaction nodes in cross-shard transactions. These elements are essential for throughput and latency, and mishandling them can increase cross-shard communication and cause load imbalance [38]. To address this, we propose CSOCPPA, a coarsening-phase optimization algorithm that identifies and preserves critical paths and high-transaction nodes during graph compression (The key symbols and notations used in our algorithms are summarized in Table 1). CSOCPPA effectively alleviates load imbalance, improves system throughput, and optimizes overall resource utilization.

(1) Adjustment during the coarsening stage: Consider a directed weighted graph $G = (T, E)$, where T represents the transaction nodes and E represents the transaction edges. Each node $t \in T$ represents a user account or contract, and each directed edge $e = (t_i, t_j) \in E$ denotes a transaction from t_i to t_j with associated cost or weight $w_e(e)$.

To evaluate transaction chain structure during coarsening, we define the longest path P as the path with the highest cumulative transaction cost or weight from the starting node to the terminal node. To capture the depth and height of such a chain, we introduce two metrics: Depth(t), which represents the maximum cumulative weight of any path ending at node t, and Height(t), which represents the maximum cumulative weight of any path starting from node t.

We define the cumulative transaction cost along a path P from t_i to t_j as:

$$\gamma(t_i, t_j) = \sum_{k=1}^{n} w_e(e_k) \tag{1}$$

where e_k are the edges along the path P.

The weight $\omega(t)$ of a node t denotes the sum of all weights of incoming and outgoing edges connected to t, representing the total transaction volume related to that account.

Based on this, Depth and Height can be recursively computed as:

$$\text{Depth}(t) = \max_{t_i \in \text{Predecessors}(t)} \Big(\text{Depth}(t_i) + \omega(t_i) + \gamma(t_i, t) \Big) \tag{2}$$

$$\text{Height}(t) = \max_{t_j \in \text{Successors}(t)} \Big(\text{Height}(t_j) + \omega(t_j) + \gamma(t, t_j) \Big) \tag{3}$$

These metrics allow us to identify high-impact paths in the transaction graph for partitioning and optimization purposes.

The formula for the longest path maxPath(e) can be adjusted as follows:

$$\text{maxPath}(e) = \text{Depth}(\text{source}(e)) + w_e(e) + \text{Height}(\text{target}(e)) \tag{4}$$

This formula calculates the longest path of transaction edge$e$ within the system. Here, source(e) represents the starting node of transaction edge$e$, and target(e) represents the terminal node of transaction edge$e$. If the maxPath(e) value of a path is high, it indicates that the accounts and transactions on the path have a significant influence or frequent transaction records. In the state partitioning process, such high-weight paths should not be fragmented; instead, the transactions on these paths should be assigned to the same partition to reduce communication and synchronization costs associated with cross-partition transactions. After completing the calculation of maxPath(e), we also need to address issues such as reducing the likelihood of transaction account isolation.

Therefore, to further improve the partitioning process, the following steps are executed, as shown in Algorithm 1.

Step 1: Degree calculation and initial matching

Calculate the degrees of all accounts and sort them in ascending order. Select the account with the lowest degree from the set of unmatched accounts and match it with its adjacent accounts. Prioritize the reduction of isolated accounts and prevent the spread of low-frequency trading accounts across shards to minimize cross-shard transactions.

Step 2: Multi-Edge matching phase

Match adjacent accounts in descending order of edge weight. If multiple accounts share the same weight, prioritize accounts with fewer merged edges to preserve high-frequency trading paths and minimize cutting critical paths.

Step 3: Network update and simplification

After each match, update the network's connections. If the network reaches the target size, proceed to the next step; otherwise, return to Step 2 and continue simplifying the network.

Step 4: Load balancing and super account creation

After coarsening, merge low-transaction accounts into super accounts. Ensure these super accounts remain connected to their original neighbors, accumulate all edge weights, and retain full transaction data.

(2) Initialization phased optimization: In the context of cross-shard transactions, the initial partitioning is critical for subsequent optimizations. The traditional Metis algorithm [17] grows regions by randomly selecting starting points, which may lead to high-frequency transaction accounts being distributed across different shards, thereby increasing the cost of cross-shard transactions and communication. To address this issue, we propose the Depth-Priority Growing Algorithm (DPGA). This algorithm prioritizes accounts with high transaction frequency and network importance (i.e., nodes with larger maxDepth) as starting points. Through a region-growing strategy, eligible neighboring nodes are merged into the same region until no more neighbors can be added. This approach reduces the dispersion of high-frequency transaction accounts and lowers the complexity of cross-shard transactions. The pseudocode is presented in Algorithm 2.

Algorithm 1: Network Coarsening with Isolation Prevention

```
Input: G , target_size , threshold
Output: G_coarsened
foreach a ∈ G.accounts do
    degree(a) ← count(neighbor (a))// Init node degree
sorted_accounts ← Sort G.accounts,
by degree // Sort by degree
while |G| > target_size do
    foreach a ∈ G.accounts do
        b ← Select neighbor(a), min_degree // Select
            low-degree neighbor
        Merge a, b // Merge nodes
        Update G, a, b // Update graph
        if degree(a) < threshold then
            Merge low-volume accounts into
            super accounts // Group small nodes
            Update connect super accounts to
            neighbors // Reconnect
    foreach (u, v) ∈ G.edges do
        edge_weight(u, v) ← calculate_weight(u, v)
            // Reweight edges
        if edge_weight(u, v) > threshold then
            Merge strongly connected accounts ;   // Preserve
             strong links
return G_coarsened ;
```

5.2. Dynamic sharding adjustment mechanism

By adjusting account allocation, the overall load of the sharding system can be significantly improved. To further ensure the system's performance, we incorporate a dynamic sharding mechanism combined with the CSOCPPA to maintain load stability across the system. The dynamic sharding adjustment mechanism monitors the GINI coefficient to assess load balance and evaluates transaction patterns to dynamically split or merge shards. This approach optimizes both load balancing and cross-shard transaction performance. The following sections provide a detailed explanation of shard load definitions, the GINI coefficient measurement criteria, and the dynamic adjustment mechanism.

(1) Definition and algorithm of shard load: In a distributed system, shard load is a key indicator for measuring the workload of shards. The calculation of shard load should comprehensively consider various factors, including the number of users stored within a shard, the transaction volume processed, and the frequency of cross-shard interactions. To ensure the comprehensiveness of the load indicator, the following load calculation methods are defined:

The user count load of shard S_i is defined as the number of users managed by the shard, expressed as:

$$l_j^u = U_j \tag{5}$$

The transaction volume load of shard S_i is defined as the total transaction volume processed by the shard within a unit of time, expressed

Algorithm 2: Depth-Priority Growing Algorithm for Cross-Shard Transaction Optimization

Input: G, target_size, threshold, maxDepth
Output: $G_{coarsened}$

foreach $a \in G.accounts$ **do**
  $degree(a) \leftarrow$ count(**neighbor** (a)) ;
  // Initialize degree
$sorted\_accounts \leftarrow$ **Sort** $G.accounts$, *by* **Priority**, *then by* **maxDepth** ;
while $|G| > target\_size$ **do**
  foreach $a \in G.accounts$ **do**
    if *maxDepth(a) > threshold* **then**
      // Prioritize high-frequency nodes
      $b \leftarrow$ **Select** *neighbor(a), max_degree* ;
      Merge *a, b* ;
      Update *G, a, b* ;
    if *degree(a) < threshold* **then**
      // Identify low-volume nodes
      Merge *low-volume accounts into super accounts* ;
      Update *connect super accounts to neighbors* ;
  foreach $(u, v) \in G.edges$ **do**
    $edge\_weight(u, v) \leftarrow$ calculate_weight(u, v) ;
    if *edge_weight(u, v) > threshold* **then**
      Merge *strongly connected accounts* ;
      // Merge strong connections
return $G_{coarsened}$;

as:

$$l_j^t = \sum_{k \in T_j} t_k \tag{6}$$

where T_i represents the set of all transactions related to shard S_i, and t_k represents the transaction volume of transaction k.

By combining these factors, the comprehensive load is defined as:

$$l_j = \alpha \cdot l_j^u + \beta \cdot l_j^t \tag{7}$$

where α and β are weighting coefficients used to balance the contribution of different load sources to the shard's pressure, this comprehensive load indicator can better reflect the actual pressure of the shard and provide a reference for continuous adjustment.

(2) Load Balancing measurement based on GINI coefficient: This study introduces the GINI coefficient as a measure to evaluate the load distribution state among shards. The GINI coefficient is a classical indicator of distribution inequality, with a range of $[0, 1]$. A value closer to 0 indicates a more balanced distribution, while a value closer to 1 indicates a more imbalanced distribution.

In the shard load scenario, the calculation formula for the GINI coefficient is as follows:

$$G = \frac{\sum_{j=1}^{m} \sum_{k=1}^{m} |l_j - l_k|}{2\, m \sum_{j=1}^{m} l_j} \tag{8}$$

where m represents the number of shards, and l_j represents the load of shard S_j. Based on the calculated GINI coefficient, the current balance of system shard loads can be directly judged. The specific judgment rules are as follows:

- $G < 0.3$: The system load is completely balanced, and all shard loads are identical.
- $G \in (0.3, 0.5]$: The system load is basically balanced, and load differences are within an acceptable range.
- $G > 0.5$: The system load is imbalanced, requiring redistribution of high-load shards or merging of low-load shards.

(3) Dynamic sharding adjustment: In a distributed shard system, load balancing operations, including shard splitting and shard merging, are triggered based on specific conditions to maintain system efficiency and fairness. When the load of a shard significantly exceeds that of others or the GINI coefficient surpasses a predefined threshold, the system triggers shard splitting. The shard with the highest load, $S_{\max}$, is identified based on the condition:

$$l_{\max} > \mu + \gamma \cdot \sigma \tag{9}$$

where $\mu = \frac{1}{m} \sum_{j=1}^{m} l_j$ is the average load of all shards, $\sigma = \sqrt{\frac{1}{m} \sum_{j=1}^{m} (l_j - \mu)^2}$ is the standard deviation, and γ is an adjustment coefficient. For users in the split shard, the system employs consistency hashing to find corresponding storage shards and updates the mapping to ensure query consistency. Conversely, when certain shards experience prolonged low load below a predefined threshold, the system triggers shard merging. The set of candidate shards for merging, S_{low}, is identified based on the condition:

$$l_j < \theta \tag{10}$$

where θ is the minimum load threshold, during the merging process, the system updates the user-to-shard mapping and ensures synchronization of query positions for affected users. These mechanisms collectively enhance system load distribution and ensure balanced utilization of resources.

6. Security analysis

6.1. Atomicity of transactions

V-Bridge uses a hypergeometric distribution to calculate the probability of failure in each *epoch*. In V-Bridge, atomicity depends on whether multiple shards can complete state verification and fund confirmation within a specified time window while maintaining a secure state. If any shard's consensus committee includes more than 1/3 malicious nodes, the shard is considered failed, potentially causing transaction abortion or triggering exceptional rollbacks.

Assume that the current *epoch* includes N valid registered nodes, with $t = f \cdot N$ being malicious. Each shard randomly selects n nodes to form its consensus group. The probability that a shard contains at least $\left\lfloor \frac{n}{3} \right\rfloor$ malicious nodes is:

$$P_{\text{shard-fail}} = \sum_{x=\left\lfloor \frac{n}{3} \right\rfloor}^{n} \frac{\binom{t}{x} \cdot \binom{N-t}{n-x}}{\binom{N}{n}} \tag{11}$$

Here, N is the total number of registered nodes, $t = f \cdot N$ is the number of malicious nodes, n is the number of consensus nodes per shard, and x denotes the number of malicious nodes in a single shard. If a transaction spans S_{txn} shards, the upper bound on the atomicity failure probability is:

$$Pr[\text{Atomicity Failure}] < S_{\text{txn}} \cdot P_{\text{shard-fail}} \tag{12}$$

As long as the proportion of malicious nodes in each shard does not exceed 1/3 (i.e., $f < 1/3$), V-Bridge can guarantee atomicity. In the subsequent analysis, we demonstrate how V-Bridge ensures atomic cross-shard transactions.

Theorem 1. *If a cross-shard transaction completes within the HTLC window T_1 and submits a valid R, V-Bridge guarantees atomicity without requiring rollback.*

Proof. Assume the transaction involves shards $S_1, S_2, \ldots, S_k$, where the funds in each shard are locked using HTLC contracts. The unlocking condition requires the receipt of a correct R that satisfies a predetermined hash value $H(R)$ within the window T_1, fulfilling the *release condition*.

Once the receiver Bob submits the correct R within T_1, all contracts are deemed releasable. Trustor nodes in each shard immediately execute the fund release and update the balance state as $\text{Table}_{\text{now}}$. This updated state is signed by both Trustors and broadcast across all involved shards for chain-level finalization. Since valid release requires the correct R and consistent co-signatures, the final state is valid only under mutual agreement. Therefore, if the transaction succeeds, all contracts release simultaneously; otherwise, if any contract fails to trigger, it indicates that R was not submitted, and the transaction is entirely aborted without partial execution. In conclusion, as long as R is submitted within T_1, the HTLC conditions ensure atomicity across all shards without the need for rollback logic.

Theorem 2. *If a cross-shard transaction misses the HTLC deadline T_1, V-Bridge ensures a consistent rollback across all shards, preventing fund loss or double spending.*

Proof. The HTLC contract in V-Bridge is equipped with a unified timeout parameter T_1. If the correct random secret R is not submitted within this time frame, the contract automatically triggers a rollback mechanism, returning the locked funds to the original accounts. Since the funds remain unreleased throughout the process, the state in each shard remains unchanged, and the system proceeds to restore consistency. There is no partial commitment or state divergence, and the design inherently prevents double spending. Therefore, even in the event of transaction failure, atomicity and consistency of the system are preserved. This completes the proof.

6.2. Trustor security

To address potential malicious or faulty behavior among Trustors, the system employs a dynamic reputation mechanism that continuously monitors node performance. Actions such as refusing to sign, forwarding delays, or failing to submit HTLC secrets are penalized. Nodes whose reputation scores fall below a defined threshold are demoted, stripped of execution privileges, and forfeit their staked collateral. In the event of partial trust failure, the system ensures continuity through leader re-election or transaction rollback. All critical operations require co-signatures from at least $2/3$ of the shard's Trustors, providing resilience against Byzantine behavior. We assume a majority of Trustors are honest in each round and that HTLC secrets are submitted within a bounded timeframe. Otherwise, contracts automatically roll back to preserve state consistency.

These assumptions are realistic in practice: Trustors must stake collateral and earn reputation over time through consistent, verifiable actions, making large-scale compromises both economically prohibitive and statistically improbable. The HTLC mechanism incorporates explicit timeouts and fallback logic, including automated rollback, which enables the system to function correctly without relying on perfect synchrony.

Execution rights are assigned dynamically based on reputation: high-reputation nodes are prioritized, while low-reputation nodes are sidelined to prevent abuse of authority. Even in cases of collusion, the co-signature requirement significantly raises the threshold for successful misconduct. Collectively, these mechanisms mitigate the risks associated with partial Trustor failures and HTLC disruptions. Rather than relying on idealized assumptions, the system maintains robustness through built-in safeguards such as incentive alignment, role rotation, and protocol-level fallback strategies.

7. Experimental evaluation

7.1. Setting

We developed a prototype of V-Bridge in Golang and evaluated its performance on Ubuntu 20.04.1. The testbed was configured with an 8-core AMD Ryzen 6000 processor, 16 GB LPDDR5-6400 memory, and a 1TB PCIe 4.0 SSD. To emulate real-world conditions, we introduced random network latency between 50–100 ms and limited the bandwidth to 500 Mbps. Each block accommodates up to 3000 transactions, with each transaction fixed at 512 bytes. The number of C-Shards was set as $S \in \{2, 4, 8, 16, 32, 64, 100\}$ to evaluate system scalability.

The dataset was extracted from the Ethereum blockchain using Python scripts from the XBlock-ETH project, including sender/receiver addresses, amounts, and timestamps. The preprocessed data was used as input for V-Bridge.

In dynamic load regulation, we assigned weight factors $\alpha = 0.4$ and $\beta = 0.6$ to user count and transaction volume. The GINI coefficient was used to evaluate shard imbalance. A split is triggered when GINI > 0.5, and merging is triggered when the condition $l_j < \theta = 0.3\mu$ is met, where μ is the average load. To reduce over-sensitivity, the adjustment factor was set to $\gamma = 1.5$. In the CSOCPPA module, all transaction edges and node weights were set to 1 to simplify computation, representing a uniform transaction cost.

For comparison, we implemented two additional load-balancing schemes: BrokerChain and X-Shard. BrokerChain utilizes a partitioning algorithm based on user account relationships, grouping frequently interacting accounts within the same shard to reduce cross-shard transactions and enhance throughput. X-Shard employs an optimistic cross-shard transaction strategy, processing transactions in parallel on input shards and verifying them via gate accounts to minimize delays. All solutions were tested under identical conditions using the PBFT [33] intra-shard consensus protocol.

7.2. System throughput

Fig. 4 illustrates how throughput varies with the number of shards (S) using a combination of boxplots, kernel density estimation curves, and scatter plots. The boxplot shows the quartile throughput range, with white dots marking the median. The smooth kernel density curve highlights data concentration, while the scatter plot visually represents individual data points. As S increases from 4 to 100, V-Bridge consistently demonstrates superior throughput, with values stabilizing around 3k to 3.5k TPS across different shard counts. This is evident in the tight interquartile ranges and smooth density curves. BrokerChain's throughput remains relatively stable but lower, hovering around 2k to 2.5k TPS, while X-Shard's performance lags behind, stabilizing just above 2k TPS. Moreover, X-Shard exhibits more variability, with wider boxplots and more scattered data points, highlighting its lower reliability in comparison to V-Bridge. These trends highlight V-Bridge's scalability and stability as the number of shards increases.

Fig. 5 examines throughput under varying transaction arrival rates (40–180 TX/s). Similar to Fig. 4, the boxplot depicts throughput distribution, while the kernel density curve and scatter plot highlight data concentration and individual variations. V-Bridge consistently achieves nearly 3k TPS across all arrival rates with minimal fluctuations, confirming its robust performance. BrokerChain maintains steady throughput around 2k TPS, though fluctuations slightly increase at higher arrival rates. X-sharp again underperforms, with throughput consistently below 2k TPS and significant variability at high arrival rates. These results reinforce the performance and stability advantages of V-Bridge under heavy transaction loads, while BrokerChain and X-sharp struggle to adapt. The detailed visualizations further validate the reliability of the data, providing a strong foundation for comparing system performance.

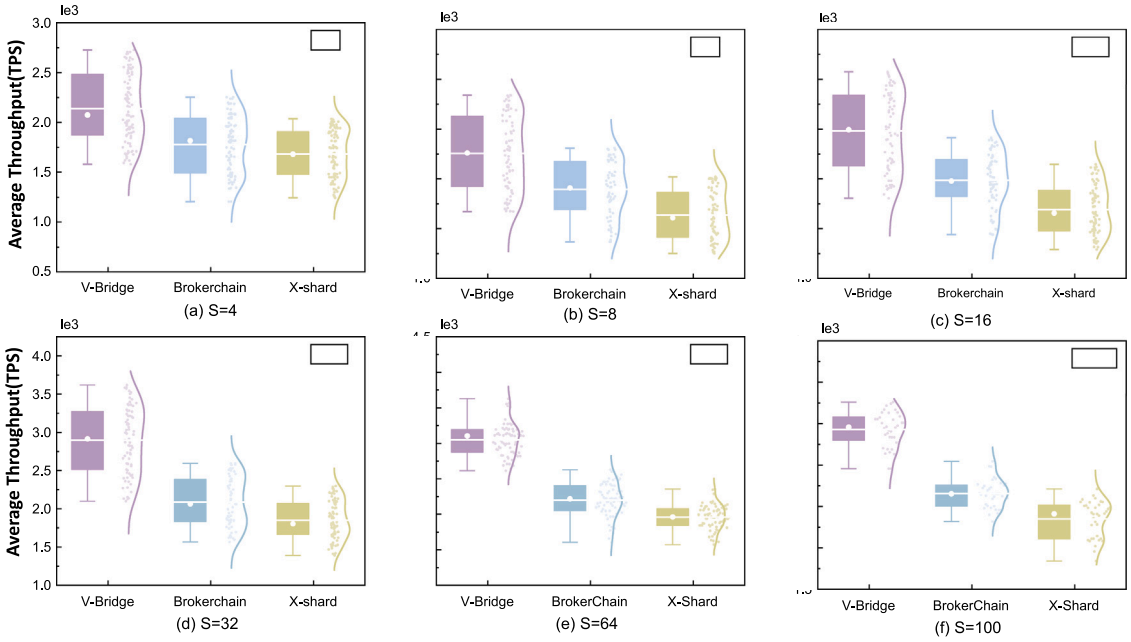

Fig. 4. The impact of the number of shards on throughput.

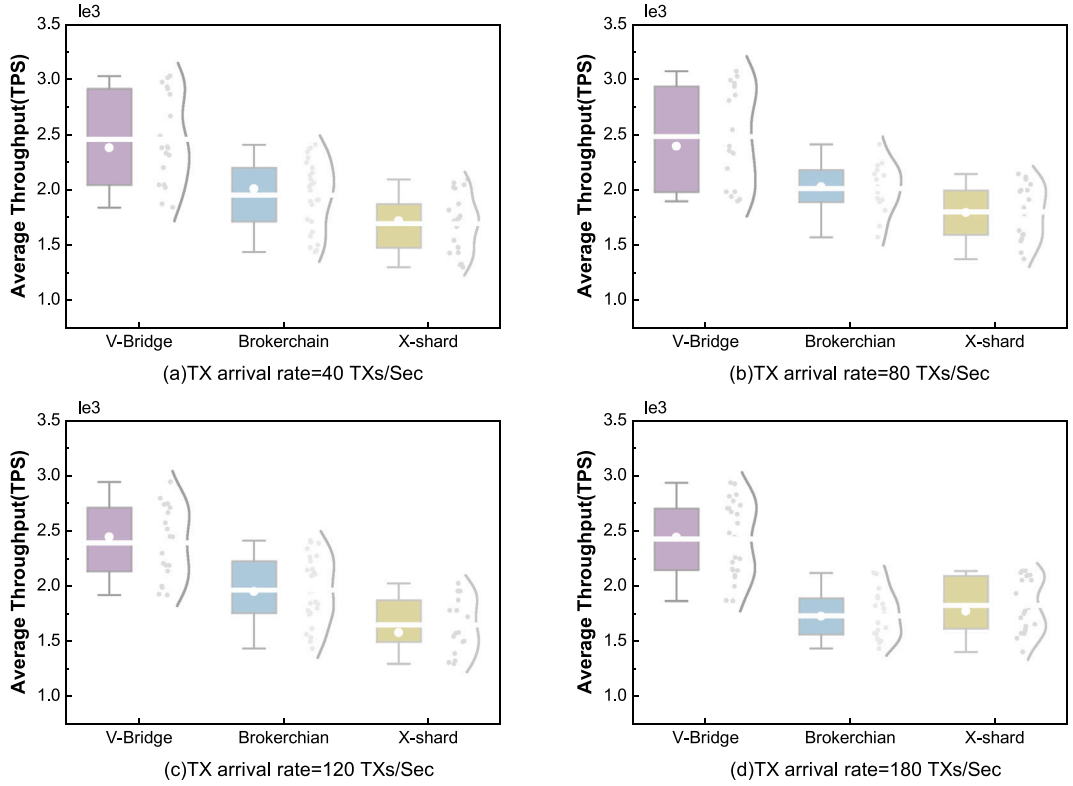

Fig. 5. Impact of transaction arrival rate on throughput.

7.3. Transaction processing delays

Fig. 6 depicts the transaction delay under different protocols as a result of factors such as the number of shards, transaction arrival rate, and cross-shard ratio. From the observed trends, these experimental results reveal key factors affecting delay and highlight the performance differences of various protocols in distributed environments.

(1) Impact of the number of shards (Fig. 6(a))

As the number of shards increases, the average transaction delay of all systems shows a gradual downward trend. This indicates that more shards can effectively distribute transaction processing workloads and improve system performance. V-Bridge demonstrates significant optimization at higher shard counts (e.g., 32 shards) with delays reduced to around 1400 ms, outperforming BrokerChain and X-Shard. This reflects V-Bridge's superior cross-shard processing efficiency. By contrast, BrokerChain and X-Shard show only slight delay reductions

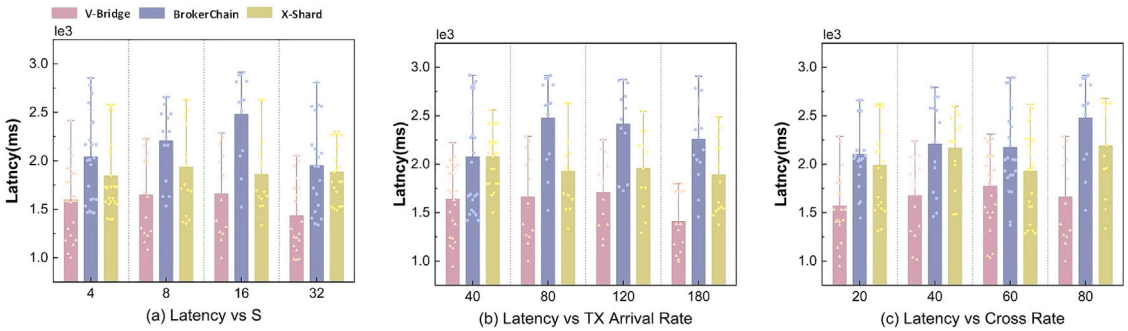

Fig. 6. Comparison of transaction delays.

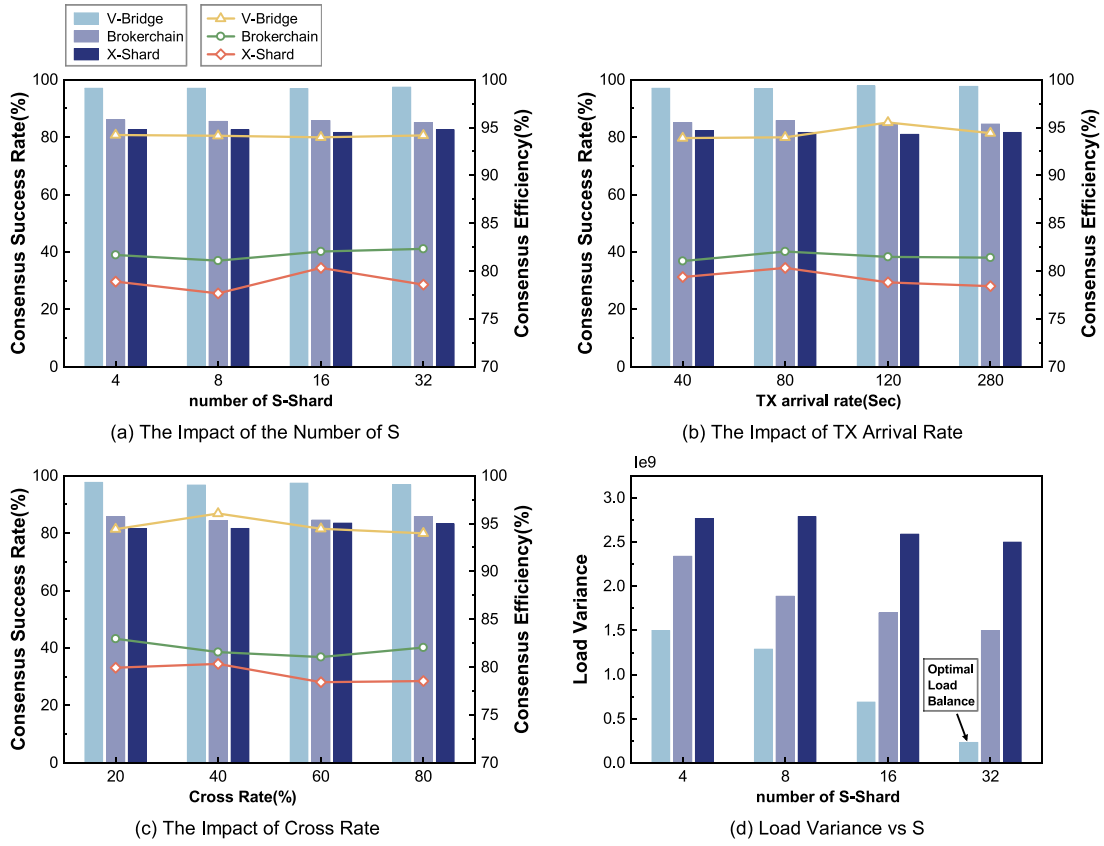

Fig. 7. Consensus and load comparison.

with more shards, and the overall delay remains higher than V-Bridge, with the gap further widening at 32 shards.

(2) Impact of transaction arrival rate (Fig. 6(b))

Increasing the transaction arrival rate leads to an apparent rise in transaction delay. All protocols exhibit relatively low delays at lower arrival rates (40 TX/s). However, as the arrival rate increases, delays grow significantly. V-Bridge maintains stable performance even at higher arrival rates (e.g., 180 TX/s), showing good scalability. In contrast, BrokerChain and X-Shard experience rapidly increasing delays at higher arrival rates (e.g., above 120 TX/s), indicating their limited capacity to handle higher loads and reflecting their performance bottlenecks in high-load environments.

(3) Impact of cross-shard ratio (Fig. 6(c))

At a lower cross-shard ratio (20%), delays across all protocols are relatively close. However, as the cross-shard ratio increases, inter-protocol differences become apparent. V-Bridge maintains stable performance even at an 80% cross-shard ratio, with an average delay of approximately 1400 ms, demonstrating excellent cross-shard processing capabilities. By contrast, BrokerChain and X-Shard experience significant delay increases, particularly when the cross-shard ratio exceeds 80%. Their delays surpass those of V-Bridge, highlighting the inefficiency of their cross-shard communication and the more significant impact of high cross-shard traffic.

7.4. Consensus and load comparison

Consensus success rate is a critical indicator in distributed systems that measures the proportion of successfully completed consensus processes within a given time. It reflects the system's ability to synchronize and process transactions efficiently while maintaining data consistency. A higher success rate directly correlates with better system performance and reliability. Conversely, a lower success rate can lead to transaction failures, increased delays, or even system reconfiguration. Optimizing consensus protocols enhances the system's stability, performance, and cross-shard transaction processing capabilities.

Fig. 7 illustrates the consensus success rate and transaction efficiency of three protocols (V-Bridge, BrokerChain, and X-Shard) under varying conditions, such as the number of shards, transaction arrival rates, and cross-shard ratios. V-Bridge consistently outperforms the other protocols across all conditions. In Fig. 7(a), as the number of shards increases, the consensus success rate and transaction efficiency for all protocols decline due to higher cross-shard communication overhead. However, V-Bridge maintains a high success rate (close to 95%) even with 32 shards, demonstrating excellent scalability. In contrast, BrokerChain and X-Shard experience significant performance degradation as the number of shards increases. Fig. 7(b) examines the impact of transaction arrival rates. V-Bridge maintains stable consensus success rates and efficiency even at high arrival rates (e.g., 180 TX/s). In contrast, BrokerChain and X-Shard exhibit significant declines in performance under heavy transaction loads, reflecting their processing limitations. Fig. 7(c) highlights the impact of cross-shard ratios. V-Bridge demonstrates strong performance, maintaining a stable success rate even at an 80% cross-shard ratio. Meanwhile, the success rates of BrokerChain and X-Shard drop sharply under high cross-shard ratios, revealing their weaknesses in handling intensive cross-shard scenarios.

Finally, the comparison of load balancing across different shard numbers shows that V-Bridge achieves the most balanced load distribution. Its variance remains minimal as the number of shards increases, approaching the "Optimal Load Balance" standard. In contrast, the load variance for BrokerChain and X-Shard is significantly higher, indicating poorer load balancing capabilities.

8. Conclusion and future work

We propose a novel virtual off-chain cross-shard transaction mechanism that employs logical fund interactions instead of actual currency transfers. This approach eliminates delays caused by continuous uploading and significantly enhances throughput. By integrating an intelligent sharding adjustment mechanism with the CSOCPPA, we address the limitations of traditional account optimization and mitigate load imbalance. Experimental results show that, compared to BrokerChain and X-Shard, V-Bridge achieves up to 50% higher average throughput and reduces transaction latency by at least 15%. Additionally, its consensus success rate consistently exceeds 90%. Across varying shard counts, V-Bridge demonstrates a progressively decreasing load, which remains consistently lower than those of the other two protocols. These results underscore V-Bridge's superior performance, scalability, and reliability as a solution for cross-shard transactions.

In future work, we plan to establish virtual fund channels among multiple Trustors to achieve interoperability across the entire shard network. Additionally, we will explore advanced optimization strategies for dynamic shard management to reduce communication further overhead. Meanwhile, we aim to incorporate zero-knowledge proofs and other cryptographic techniques into security management to enhance system security.

CRediT authorship contribution statement

Xueting Huang: Writing – original draft, Software, Methodology, Conceptualization. **Xiangwei Meng:** Writing – review & editing, Validation, Supervision. **Kai Zhang:** Formal analysis, Conceptualization. **Ce Yang:** Methodology, Formal analysis. **Wei Liang:** Writing – review & editing, Funding acquisition, Formal analysis, Conceptualization. **Kuan-Ching Li:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Key Program of the Joint Funds of the National Natural Science Foundation of China under Grant U2468205, the National Natural Science Foundation of China under Grants 62472168, 62072170 and 61976087, the Hunan Provincial Natural Science Foundation of China under Grants 2021JJ30141 and 2024JJ6066, the Key Research and Development Program of Hunan Province under Grant 2022GK2015, the Science and Technology Project of the Department of Communications of Hunan Province under Grant 202101, and the Research Projects of the Hunan Provincial Department of Education under Grants 23B0449 and 23B0288.

Data availability

Data will be made available on request.

References

[1] J. Xu, C. Wang, X. Jia, A survey of blockchain consensus protocols, ACM Comput. Surv. 55 (13s) (2023) 1–35.
[2] S. Zhang, Z. Yan, W. Liang, K.-C. Li, B. Di Martino, BCAE: A blockchain-based cross domain authentication scheme for edge computing, IEEE Internet Things J. 11 (13) (2024) 24035–24048.
[3] W. Liang, Y. Yang, C. Yang, Y. Hu, S. Xie, K.-C. Li, J. Cao, PDPChain: A consortium blockchain-based privacy protection scheme for personal data, IEEE Trans. Reliab. 72 (2) (2022) 586–598.
[4] W. Liang, S. Xie, K.-C. Li, X. Li, X. Kui, A.Y. Zomaya, MC-DSC: A dynamic secure resource configuration scheme based on medical consortium blockchain, IEEE Trans. Inf. Forensics Secur. 19 (2024) 3525–3538.
[5] J. Cai, W. Liang, X. Li, K. Li, Z. Gui, M.K. Khan, GTxChain: A secure IoT smart blockchain architecture based on graph neural network, IEEE Internet Things J. 10 (24) (2023) 21502–21514.
[6] M.M. Islam, M.K. Islam, M. Shahjalal, M.Z. Chowdhury, Y.M. Jang, A low-cost cross-border payment system based on auditable cryptocurrency with consortium blockchain: Joint digital currency, IEEE Trans. Serv. Comput. 16 (3) (2022) 1616–1629.
[7] Y. Lu, The blockchain: State-of-the-art and research challenges, J. Ind. Inf. Integr. 15 (2019) 80–90.
[8] H. Jin, J. Xiao, Towards trustworthy blockchain systems in the era of "internet of value": development, challenges, and future trends, Sci. China Inf. Sci. 65 (153101) (2022) 1–11.
[9] X. Meng, W. Liang, Z. Xu, K. Li, M.K. Khan, X. Kui, An anonymous authenticated group key agreement scheme for transfer learning edge services systems, ACM Trans. Sen. Netw. 20 (2024).
[10] T. Chen, Z. Li, Y. Zhu, J. Chen, X. Luo, J.C.-S. Lui, X. Lin, X. Zhang, Understanding ethereum via graph analysis, ACM Trans. Internet Technol. (TOIT) 20 (2) (2020) 1–32.
[11] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, P. Saxena, A secure sharding protocol for open blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM SIGSAC, 2016, pp. 17–30.
[12] Z. Hong, S. Guo, P. Li, Scaling blockchain via layered sharding, IEEE J. Sel. Areas Commun. 40 (12) (2022) 3575–3588.
[13] F. Cheng, J. Xiao, C. Liu, S. Zhang, Y. Zhou, B. Li, B. Li, H. Jin, Shardag: Scaling dag-based blockchains via adaptive sharding, in: 2024 IEEE 40th International Conference on Data Engineering, ICDE, IEEE, 2024, pp. 2068–2081.
[14] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, H. Zhang, Meepo: Multiple execution environments per organization in sharded consortium blockchain, IEEE J. Sel. Areas Commun. 40 (12) (2022) 3562–3574.
[15] J. Wang, H. Wang, Monoxide: Scale out blockchains with asynchronous consensus zones, in: 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 19, USENIX Association, 2019, pp. 95–112.
[16] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, S. Guo, Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, IEEE, 2022, pp. 1968–1977.
[17] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[18] J. Xu, Y. Ming, Z. Wu, C. Wang, X. Jia, X-Shard: Optimistic cross-shard transaction processing for sharding-based blockchains, IEEE Trans. Parallel Distrib. Syst. 35 (4) (2024) 548–559.
[19] Y. Levi, I. Keslassy, Beyond the ring: Quantized heterogeneous consistent hashing, in: 2023 IEEE 31st International Conference on Network Protocols, ICNP, IEEE, 2023, pp. 1–12.

[20] G. Mendelson, S. Vargaftik, K. Barabash, D.H. Lorenz, I. Keslassy, A. Orda, Anchorhash: A scalable consistent hash, IEEE/ACM Trans. Netw. 29 (2) (2020) 517–528.
[21] B. Hou, D. Wang, T. Xia, L. Xi, Z. Peng, K.-L. Tsui, Generalized Gini indices: Complementary sparsity measures to Box–Cox sparsity measures for machine condition monitoring, Mech. Syst. Signal Process. 169 (2022) 108751.
[22] X. Qi, Y. Li, LightCross: Sharding with lightweight cross-shard execution for smart contracts, in: IEEE INFOCOM 2024-IEEE Conference on Computer Communications, IEEE, 2024, pp. 1681–1690.
[23] S. Jiang, J. Cao, C.L. Tung, Y. Wang, S. Wang, SHARON: Secure and efficient cross-shard transaction processing via shard rotation, in: Proceedings of the IEEE International Conference on Computer Communications, INFOCOM, IEEE, 2024, pp. 20–23.
[24] Y. Zhang, S. Pan, J. Yu, Txallo: Dynamic transaction allocation in sharded blockchain systems, in: 2023 IEEE 39th International Conference on Data Engineering, ICDE, IEEE, 2023, pp. 721–733.
[25] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, Omniledger: A secure, scale-out, decentralized ledger via sharding, in: 2018 IEEE Symposium on Security and Privacy, SP, IEEE, 2018, pp. 583–598.
[26] M. Zamani, M. Movahedi, M. Raykova, Rapidchain: Scaling blockchain via full sharding, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM SIGSAC, 2018, pp. 931–948.
[27] Z. Hong, S. Guo, P. Li, W. Chen, Pyramid: A layered sharding blockchain system, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021, pp. 1–10.
[28] A. Liu, Y. Liu, Q. Wu, B. Zhao, D. Li, Y. Lu, R. Lu, W. Susilo, CHERUBIM: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains, IEEE Trans. Inf. Forensics Secur. 19 (2024) 3178–3193.
[29] X. Wang, C. Lin, X. Huang, D. He, Anonymity-enhancing multi-hop locks for monero-enabled payment channel networks, IEEE Trans. Inf. Forensics Secur. 19 (2023) 2438–2453.
[30] T. Cai, W. Chen, K.E. Psannis, S.K. Goudos, Y. Yu, Z. Zheng, S. Wan, Scalable on-chain and off-chain blockchain for sharing economy in large-scale wireless networks, IEEE Wirel. Commun. 29 (3) (2022) 32–38.
[31] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, Z. Liu, Cross-chain virtual payment channels, IEEE Trans. Inf. Forensics Secur. 18 (2023) 3401–3413.
[32] Z. Li, W. Su, M. Xu, R. Yu, D. Niyato, S. Xie, Compact learning model for dynamic off-chain routing in blockchain-based IoT, IEEE J. Sel. Areas Commun. 40 (12) (2022) 3615–3630.
[33] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, M.A. Imran, A scalable multi-layer PBFT consensus for blockchain, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2020) 1146–1160.
[34] H. Azimy, A.A. Ghorbani, E. Bagheri, Preventing proof-of-work mining attacks, Inform. Sci. 608 (2022) 1503–1523.
[35] A. Hosoyamada, Y. Sasaki, Quantum collision attacks on reduced SHA-256 and SHA-512, in: Annual International Cryptology Conference, Springer, 2021, pp. 616–646.
[36] C. Boyd, K. Gjøsteen, S. Wu, A blockchain model in tamarin and formal analysis of hash time lock contract, in: 2nd Workshop on Formal Methods for Blockchains, FMBC 2020, Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020, p. 13.
[37] Y. Liu, W. Liang, K. Xie, S. Xie, K. Li, W. Meng, LightPay: A lightweight and secure off-chain multi-path payment scheme based on adapter signatures, IEEE Trans. Serv. Comput. 17 (4) (2023) 1503–1523.
[38] J. Herrmann, J. Kho, B. Uçar, K. Kaya, Ü.V. Çatalyürek, Acyclic partitioning of large directed acyclic graphs, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, IEEE, 2017, pp. 371–380.