ShardBridge: Enabling Efficient Cross-Shard Transactions via Payment Channels

***Abstract*—Sharding has been promising for enhancing blockchain scalability. However, as the number of shards increases, most transactions become cross-shard, causing significant confirmation delays. In this paper, we propose ShardBridge, a cross-shard protocol that leverages payment channels (PCs) in ShardBridge to handle cross-shard transactions, avoiding the high communication complexity between shards. ShardBridge also processes cross-shard transactions in batches to further improve the transaction throughput. In ShardBridge, each shard maintains only its own blockchain to achieve state sharding, which reduces the storage overhead. We also theoretically analyze the security properties of ShardBridge. Finally, we implement a prototype of ShardBridge and evaluate its performance using real-world Ethereum transactions. Evaluation results demonstrate that ShardBridge can efficiently process cross-shard transactions and outperform state-of-the-art protocols in terms of transaction throughput and confirmation latency.**

***Index Terms*—Blockchain, Sharding, Payment Channel.**

I. Introduction

Sharding enhances blockchain scalability by dividing the validators of the entire blockchain network into smaller groups (shards, committees, or zones) [1–3]. The consensus protocol is executed independently in each shard, and each shard maintains its blockchain. Practical Byzantine Fault Tolerance (PBFT) consensus protocol and its variants are widely adopted in many industrial blockchains (e.g., Hyperledger Fabric [4], Zilliqa [5], Tendermint [6]) for their resilience to Byzantine faults and immediate finality [7, 8].

Despite the scalability benefits of state sharding, it presents several design challenges. First, 96% of the transactions become cross-shard as the number of shards increases, involving different sender and receiver shards [9, 10]. Second, cross-shard transactions require cooperation between shards, leading to high transaction confirmation latency. This is because requesting and retrieving state information from other shards causes extra overhead. Third, fewer validators per shard heighten the risk of attacks compared to the entire network [11], such as Sybil attacks [12] and double-spending attacks [10]. Additionally, a sharding system needs to ensure cross-shard transaction atomicity. Most existing schemes [1, 3] also require validators to sign and broadcast every valid cross-shard transaction to other shards, resulting in $O(m^2)$ cross-shard communication overhead, where m is the number of validators per shard. Finally, to process cross-shard transactions, most existing sharding protocols [3, 9, 13, 14] require some or all validators to store the entire state of the network.

To minimize the cross-shard transaction latency without extra communication or storage overhead, while ensuring the sharded system security, we leverage the payment channel (PC) to handle the cross-shard transactions. PC has been a second-layer solution to address blockchain scalability issues by conducting payments off-chain. Two popular implementations are: Bitcoin's Lightning Network [15] and Ethereum's Raiden Network [16]. The key properties of PCs we value here are the fast off-chain state updates and instant dispute settlements. Leveraging PCs for cross-shard transactions is non-trivial and poses three design challenges.

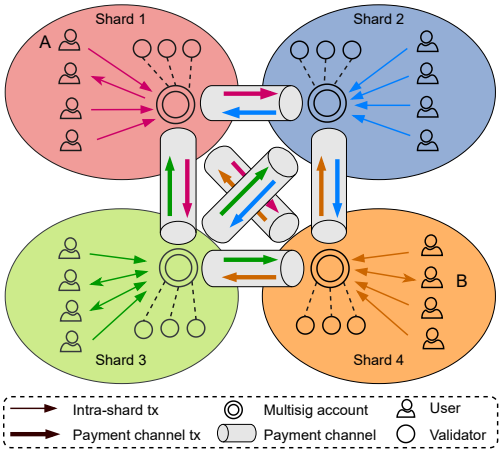


Fig. 1: Example of cross-shard transaction processing in ShardBridge. Every payment channel connects every pair of shards. A transaction from sender A to receiver B is split into three transactions: from A to Shard 1's Multisig account, from Shard 1's Multisig account to Shard 4's Multisig account, and from Shard 4's Multisig account to receiver B.

Challenges: (1) Constructing one PC between two shards requires two blockchain accounts to represent them, respectively, which are not provided in any current sharding protocol designs; **(2)** Updating balances on a PC between two shards requires a balance proof including consents from the shards to prevent single-point failures, which is not provided by the current balance proof [16] designed for PC; **(3)** Achieving high transaction throughput and low confirmation latency relies on seamless integration of PC into the sharded blockchain.

To address these challenges, we propose ShardBridge, a sharding protocol that boosts the performance of the sharded blockchain by leveraging secure PCs. To deploy PCs in ShardBridge, we first define how to construct a Multisig account in a shard and PCs for shards. Next, we explore how ShardBridge processes cross-shard transactions. For each cross-shard transaction, ShardBridge creates three new transactions: two special intra-shard transactions and one PC transaction, as shown in Fig. 1. ShardBridge reduces latency by eliminating cross-shard communication through balance updates on PCs between shards. We also design a rollback

scheme to ensure atomicity by rolling back failed cross-shard transactions. The main contributions of this paper are:

- We propose a novel protocol ShardBridge, leveraging PCs for efficient cross-shard transaction processing. It achieves a constant communication complexity between the shards.
- We utilize the threshold signature scheme to sign messages in batches to further improve the cross-shard transaction throughput. To enable this process, we modify the original block and the PBFT consensus algorithm.
- We take advantage of the modified PBFT consensus to generate signatures for the balance proof in PC balance update, thereby eliminating the need to generate signatures separately for the balance proof.
- We design a rollback scheme for ShardBridge to ensure the atomicity of the cross-shard transactions.
- Evaluation results demonstrate that ShardBridge achieves higher throughput and lower latency compared to two state-of-the-art cross-shard protocols.

Organization. §II overviews the background. §III describes the system and threat model. §IV illustrates the design. §V shows the security analysis. §VI provides evaluation results. §VII concludes this paper.

II. Background

A. Transaction Model

Distributed ledgers maintain system state through an ordered chain of transaction blocks. Two transaction models are widely used: the account model and the Unspent Transaction Output (UTXO) model. The representative blockchain for the account model (similar to a bank account) is Ethereum [17]. In the account model, the blockchain state is recorded as the balance of each account, and transactions typically involve one sender account and one receiver account. In the UTXO model, one or more UTXOs might be used to cover the total cost and the transaction fee. More than one UTXO can also be generated as new outputs. The UTXO model has been widely used in Bitcoin, Zcash, and Dogechain [18–20].

B. Payment Channel

Two blockchain accounts are required to open a PC, which involves a funding transaction by depositing funds on-chain through a smart contract [16] or a message exchange scheme in the Lightning Network [15]. Both parties of a PC sign the initial state to set the starting point of this channel. Then, they exchange encrypted messages to update the state of the PC. For example, the balance on a PC can be updated by sending the *balance proof* [16]. The balance proof includes the total sum of all token transfers sent to a participant up to a specific point, digitally signed by the sender. A strictly monotonic nonce is included in the balance proof to prevent replay attacks. Next, to close the PC, a final state agreed by both parties is recorded on-chain, after which the locked deposits are returned to the participants. Only the final result will be recorded on the blockchain. Any dispute can be resolved on-chain by responding within a challenge time window [16]. The malicious party will be penalized by forfeiting all its deposits.

Protocol	Model	Message Complexity	Storage
Elastico [3]	UTXO	$\Omega(m^2/b+\nu)$	x
OmniLedger [9]	Account/UTXO	$\Omega(m^2/b+\nu)$	x/k
RapidChain [27]	UTXO	$O(m^2/b+m\log\nu)$	x/k
Monoxide [2]	Account	$O(m^2/b+m+k)$	x/k
BrokerChain [10]	Account	$O(m^2/b+\nu)$	x/k
ShardBridge	Account	$O(m^2/b)$	x/k
ShardBridge	UTXO	$O(m^2/b+k)$	x/k

TABLE I: Comparison of different sharding protocols, where x denotes the amount of storage required for each validator to process the transactions, k denotes the number of shards, m denotes the number of validators in the shard, b denotes the block size, and ν denotes the total number of validators.

In this paper, we focus solely on PCs and leave the extension of state channels for future work.

C. Threshold Signature Scheme

The threshold signature scheme involves multiple parties working together to generate a joint signature. In a t-of-m threshold signature scheme (TSS) [21], m denotes the total number of signers, and t denotes that any group of t players can jointly sign, while any smaller group cannot. The TSS scheme provides nonforgeability, robustness, anonymity, and privacy [22, 23], and has also been well-studied in blockchain systems [24–26]. The four main functions of the TSS are:

- **Key Generation:** Generate a private key for each validator and a joint public key given a security parameter.
- **Partial Signature Generation:** Generate a partial signature for a validator using its private key and a message.
- **Signature Combination:** All the partial signatures are combined to generate a joint signature.
- **Verification:** The joint public key, the message, and the joint signature can be used to verify the correctness of the joint signature. True is returned if the verification passes; otherwise, false is returned.

D. Related Work

We summarize several recent sharding protocols shown in Table I. Elastico [3] is the first partial sharding proposal on a permissionless blockchain. Elastico suffers from inefficiencies in handling cross-shard transactions by requiring a final committee to store the entire blockchain network for transaction verifications. OmniLedger [9] introduces state blocks to enable fast bootstrapping and reduce storage costs, employing a two-phase commit mechanism for cross-shard transaction processing. RapidChain [27] operates on a synchronous UTXO-based network. Monoxide [2] utilizes a relay mechanism in the account model by creating relay transactions for the original cross-shard transactions. The proof of work is used as the intra-shard consensus mechanism, which gives probabilistic finality. BrokerChain [10] uses account segmentation, using brokers to handle cross-shard transactions. More recently, the Ethereum community has been working to implement Danksharding [28]. In our protocol, cross-shard messaging is facilitated through PCs, reducing the message complexity to $O(1)$ for the account model, and $O(k)$ for the UTXO model.

III. System Model and Threat Model

In this section, we describe the system model and threat model for constructing PCs in the sharded system.

A. System Model

Assume there are ν validators that are connected partially synchronously in the blockchain network. The validators in the whole blockchain network are divided into a set $\mathcal{S}$ of k *regular* shards $\{S_1, S_2, \ldots, S_k\}$ and a *identity* shard [3, 9]. The *regular* shards are for processing transactions, and the *identity* shard is for storing the identities of all validators. In the following, we refer to regular shards as shards when the context is clear. In this paper, we focus on the account model [17] for accounting and will discuss the adaptation to the UTXO model [18] in §IV-E. In the account model, each blockchain user has an account to send and receive cryptocurrency payments and a public-private key pair for signing transactions through the ECDSA algorithm [29]. To enable multi-sender and multi-receiver transactions, specific smart contracts need to be deployed on-chain. We leave this application to our future work. To validate transactions, each shard serves as a committee composed of m validators responsible for processing transactions for a designated group of accounts. Each validator in S_i possesses a partial private key to generate its partial signature within the TSS scheme [22]. For each shard $S_i \in \mathcal{S}$, let L_i denote the *consensus leader* in S_i, let $\mathcal{V}_i$ denote the set of validators in S_i excluding L_i, and let pk_i denote the joint public key of S_i under TSS.

B. Threat Model

Both honest and malicious validators can exist in ShardBridge. The malicious validators can collude with other malicious validators and not follow ShardBridge, such as sending incorrect, inconsistent messages, or being unresponsive. Similar to existing sharding and security works [3, 9, 13, 30–35], we account for a probabilistic polynomial-time Byzantine adversary that can corrupt fewer than $f < \frac{m}{3}$ of the validators at any given time, where f is the number of malicious validators in a shard. Additionally, malicious adversaries are slowly adaptive, which means it takes time to corrupt validators [9, 13, 27, 36]. In other words, the set of malicious and honest validators in a shard remains the same during each epoch, which might be changed only between epochs. The threshold t of the t-of-m TSS is set to $t = 2f + 1$ to match the least number of commits received in intra-shard consensus, which indicates that at least $2f+1$ partial signatures are needed to generate a valid threshold signature in a shard. $\mathcal{F} \cdot \nu$ denotes validators controlled by Byzantine adversaries in the whole blockchain network, where $\mathcal{F}$ is the fraction of total malicious validators in the system and $\mathcal{F} < \frac{1}{3}$. In ShardBridge, leaders can be malicious and refuse to forward cross-shard transactions to other shards.

IV. Design of ShardBridge

In this section, we propose ShardBridge to handle the cross-shard transactions in the account-based blockchain.

A. Overview of ShardBridge

ShardBridge processes transactions in repeated fixed time intervals (*epochs*). Each epoch shown in Fig. 2 has two phases: reconfiguration and consensus. The reconfiguration begins by assigning validators to shards and creating a Multisig account for each shard. The PC is then constructed for each pair of shards. Finally, account states are partitioned. The reconfiguration consists of four stages as follows:

(1) *Validator Assignment:* A proof of work (PoW) puzzle hard-coded by the *identity* shard is generated at the start of each epoch, similar to [3, 9, 27]. Each validator must solve this puzzle to prevent Sybil attacks when joining the system. Next, each validator commits its solution to the *identity* shard to record its identity on-chain. An identity uniquely identifies a validator through its public key, IP address, and a valid proof-of-work solution for the identity shard. The entire blockchain network is partitioned into shards, each containing a set of validators to validate incoming transactions in parallel [9, 36, 37]. Within each shard, validators take turns serving as the *consensus leader* for that shard, following a predefined sequence determined by their ID and the current view of the consensus [7, 38]. Consensus leaders can also be selected either through a round-robin fashion [8] or via a RANDAO mechanism as in Ethereum [17]. The leader of a shard, henceforth, refers to the consensus leader.

(2) *Multisig Account Creation:* To create PCs, each shard needs a Multisig account constructed with the consent of a fraction of the validators within that shard. The details of the fraction of the validators in a shard who contribute to the PCs will be elaborated in §IV-F. Multisig wallets are designed for a group, such as the Ethereum-compatible Multisig wallet Gnosis Safe [39], Argent [40], and TOTALSIG [41], etc.

(3) *PC Construction:* The PC construction completes in two steps as shown in Fig. 3(a). In Step 1, each validator deposits certain tokens from their blockchain account into the Multisig account. The leaders of every two shards can open a channel using the two respective Multisig accounts through, for example, the *ChannelOpened* function in Raiden Network [16]. The *settle_timeout* field in the *ChannelOpened* is set as the number of blocks to be mined in an *epoch*. In Step 2, each leader deposits tokens as the shard's balance on the PC from their respective Multisig account through, for example, the function *setTotalDeposit* in Raiden Network [16]. PCs are then constructed between every two shards. When an *epoch* ends, validators are reassigned to shards, and they need to reconstruct PCs. If the capacity of the PC is not enough to support all cross-shard transactions, the system offloads the cross-shard transaction processing to the common two-phase commit (2PC) protocol as in [3, 9].

(4) *Account Partition:* All shards partition blockchain accounts into non-overlapping k groups using Metis [42] in a decentralized manner, as adopted by existing works [10, 43], a classic balanced graph edge partition algorithm designed to minimize the probability of cross-shard transaction existence. Then, these accounts are mapped to k shards, and each shard is responsible for maintaining its group of accounts. The assigned validators within each shard begin the bootstrapping process [9] to crawl the entire distributed ledger and maintain the consistency of the state information. This step constructs UTXOs or gathers account data to maintain a consistent state

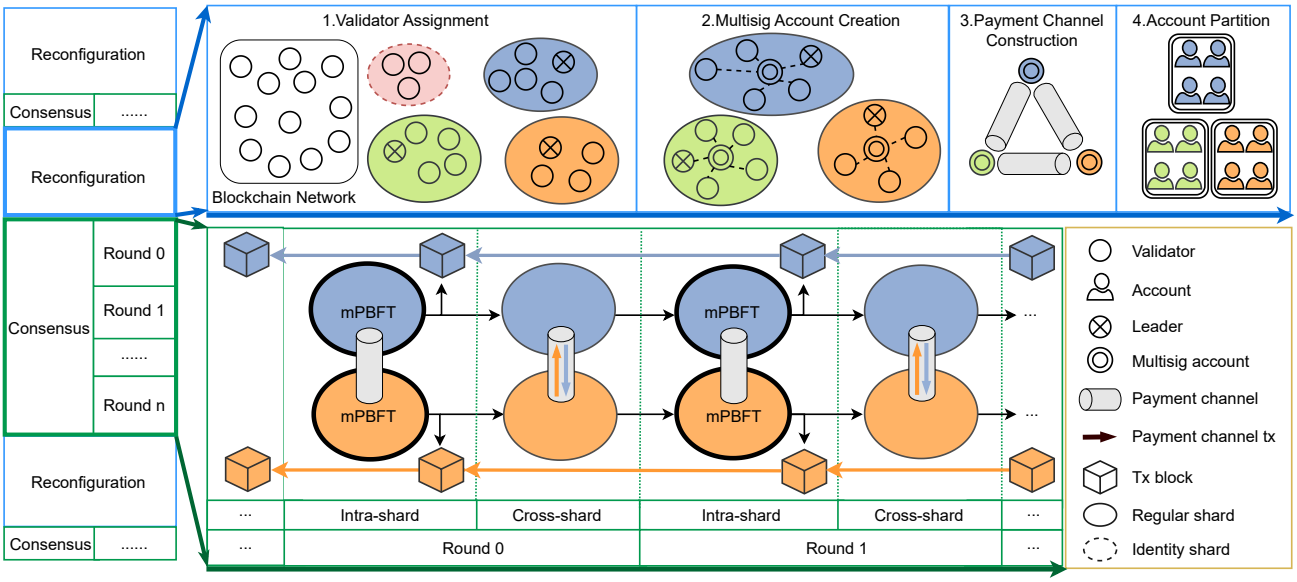
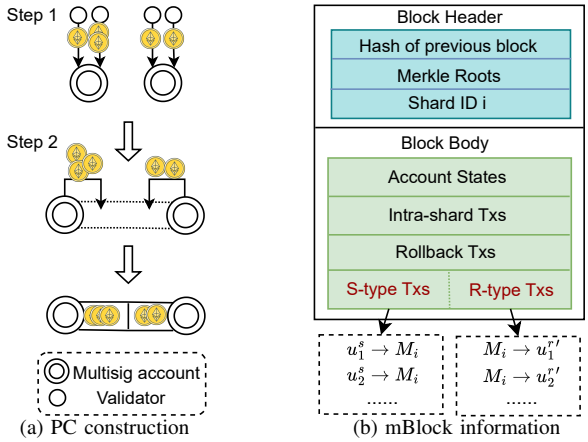


Fig. 2: Overview of ShardBridge



(a) PC construction (b) mBlock information

Fig. 3: PC construction and mBlock information

as accounts move across shards.

The consensus phase consists of multiple rounds, each with two stages: intra-shard and cross-shard consensus.

(1) *Intra-shard consensus:* We modify the PBFT, referred to as the modified PBFT (mPBFT), for shards to process the intra-shard transactions. We also modify the block, referred to as the modified block (mBlock), for better supporting mPBFT. Details will be explained in §IV-B.

(2) *Cross-shard consensus:* Each cross-shard transaction involves a *sender's shard* and a *receiver's shard*, based on where the sender and receiver accounts belong. Leaders handle each cross-shard transaction by creating two special intra-shard transactions (one in the sender's shard and the other in the receiver's shard) and one PC transaction. For the PC transaction, the sender of a balance proof is the leader in the sender's shard. The joint signature serves as the signature in the balance proof.

B. mBlock and mPBFT

We adapt the original block and PBFT to integrate with TSS to manage two special intra-shard transactions for cross-shard transactions. Each mBlock S_i has a header and a body as shown in Fig. 3(b). The header now includes the index of the shard ID i, and the body contains intra-shard, rollback, *S-type*, and *R-type* transactions. The S-type and R-type transactions are two special intra-shard transactions converted from cross-shard transactions. Intra-shard transactions should have both the sender and the receiver belonging to S_i. Rollback transactions ensure the cross-shard transaction atomicity and will be explained in §IV-D. An S-type transaction is a transaction whose sender is an account belonging to S_i and whose receiver is the Multisig account M_i. An R-type transaction is a transaction whose sender is M_i and whose receiver is an account belonging to S_i. PBFT is selected due to its wide application in many deployed blockchains [4–6]. Our mPBFT has the following stages:

(1) *Pre-prepare*: The leader L_i in S_i proposes an mBlock as shown in Fig. 3(b). L_i then sends the pre-prepare message including the mBlock to $\mathcal{V}_i$ in S_i. Upon $\mathcal{V}_i$ receiving the pre-prepare message, $\mathcal{V}_i$ checks the legitimacy of the message and then broadcasts their respective prepare message in S_i.

(2) *Prepare*: Upon receiving $2f+1$ prepare messages from other validators during the prepare phase, any honest validator in $\mathcal{V}_i$ verifies the validity of the mBlock by ensuring each transaction's sender has sufficient balance for the payment and that the sender's signature matches the transaction. If valid, honest validators in S_i sign the mBlock using their partial private key to produce partial signatures, as introduced in §II-C. Then, they broadcast their commit messages, including their partial signatures, to other validators in S_i.

(3) *Commit*: Any honest validator who receives at least $2f+1$ commits from other validators in S_i (including itself) verifies if these commits are consistent. If valid, they confirm the mBlock and construct a joint signature using partial signatures from these commits, as described in §II-C. Once L_i collects $2f+1$ commits and constructs the joint signature σ_i, it shares σ_i with leaders in other shards for processing cross-shard transactions.

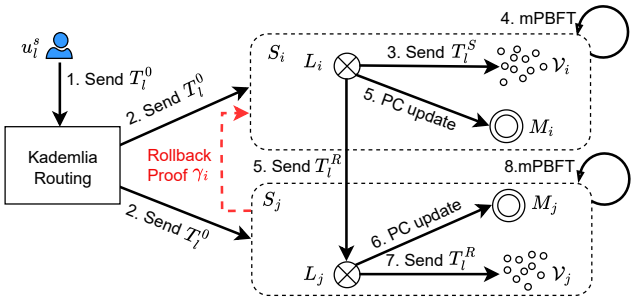


Fig. 4: Cross-shard transaction processing

C. Processing Cross-shard Transactions

ShardBridge handles the cross-shard transactions by creating three new transactions: an **S-type transaction**, **a PC transaction**, and an **R-type transaction**. We use Fig. 4 as an example to illustrate the details of handling cross-shard transactions:

(1) The sender u_l^s in S_i sends a cross-shard transaction to u_l^r in S_j. Let $T_l^0{:=}(u_l^s, u_l^r, a, \sigma_s, H_i^0, \tau)$ denote this transaction, where l denotes the transaction index, a denotes the transaction amount, σ_s denotes the signature of u_l^s (signed using ECDSA [29]), H_i^0 denotes the block height in S_i when T_l^0 is initiated, and τ denotes the locking duration for a being locked in S_i, which is the number of future blocks starting from H_i^0. A strictly monotonic nonce is included in T_l^0 to prevent replay attacks [10].

(2) T_l^0 initiated by u_l^s is routed to S_i and S_j shards (e.g. via Kademlia routing protocol in many similar works [26, 27, 30]).

(3) Leader L_i in S_i creates the S-type transaction T_l^S for T_l^0, denoted as $T_l^S{:=}(T_l^0, u_l^s, M_i)$, where u_l^s is the sender and M_i is the receiver. Next, L_i sends T_l^S to other validators $\mathcal{V}_i$.

(4) L_i proposes T_l^S together with other S-type, R-type, Rollback, and intra-shard transactions as an mBlock of S_i. Then, $\mathcal{V}_i$ verifies if the proposed mBlock is valid through mPBFT. The verification includes checking if u_l^s has enough balance to initiate T_l^S and if σ_s is valid. If all transactions in the mBlock pass such verification, each validator in $\mathcal{V}_i$ signs the proposed mBlock and broadcasts its commit including its partial signature (§II-C).

(5) L_i collects the partial signatures from $\mathcal{V}_i$ and combines them as a joint signature σ_i through the TSS. Now, the mPBFT process finishes, and L_i appends this new mBlock to the blockchain of S_i with H_i^S ($H_i^S > H_i^0$), where H_i^S denotes the current height of S_i when the S-type transaction T_l^S is committed. L_i then creates the R-type transaction denoted as $T_l^R := (T_l^S, M_j, u_l^r, H_i^S, \sigma_i)$, where the sender account is M_j, and the receiver is u_l^r. At the same time, L_i updates the balance of M_i on the PC. L_i packs T_l^R with other R-type transactions whose receivers are also in S_j. Then, L_i sends these transactions to L_j as a balance proof for L_j to update the balance of M_j on PC. For ease of illustration, we use the single transaction T_l^R to represent the balance proof here.

(6) After receiving the balance proof including T_l^R, L_j verifies if T_l^R is received within $H_j^R{<}\frac{2}{3}\tau + H_i^0$ and if σ_i is valid using pk_i, where H_j^R is the current block height in S_j when L_j receives T_l^R. A transaction is valid only if it passes both verifications. After L_j finishes the verification process, it increases the balance of M_j on the PC by the net amount of valid transactions in the balance proof.

(7) L_j sends T_l^R to the set of validators $\mathcal{V}_j$ in S_j.

(8) L_j packs T_l^R with other transactions as an mBlock, which are then committed in S_j.

If S_j is unable to initiate mPBFT on T_l^R, then S_j will initiate a rollback mechanism to ensure the atomicity of cross-shard transactions, as detailed in the following subsection.

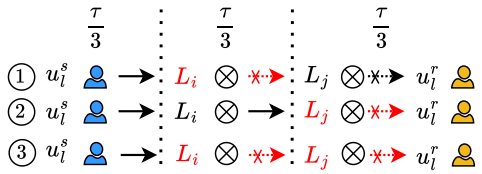


Fig. 5: Three potential failure cases

D. Atomicity Guarantee

We still use the example in Fig. 4 to explain the rollback scheme. Each validator in S_j knows the lock duration τ for T_l^0 from Step 1. Thus, it is easy for S_j to detect if T_l^R is received on time. Generally, updating a PC is much quicker than performing mPBFT consensus since the update is handled by the leaders and occurs off-chain. However, to allow enough time for PC update, we equally divide τ among the two intra-shard consensus processes and a PC update as shown in Fig. 5. Note that this time allocation can be adjusted according to the system specification. To provide a succinct overview, we summarize all three possible failure cases as follows:

Case 1 (Failure of L_i): L_i fails to forward T_l^R to L_j. PC is not updated, and T_l^R is not transmitted to S_j.

Case 2 (Failure of L_j): L_j receives T_l^R but fails to commit T_l^R. PC is updated, but $\mathcal{V}_j$ does not receive T_l^R.

Case 3 (Collusion between L_i and L_j): L_i and L_j collude. The PC remains unchanged, and T_l^R is not transmitted to S_j.

In Case 1 and Case 3, balances on PC are not updated. To address Case 1, any honest validator v_j in S_j who observes that T_l^R has not been transmitted within $H_i^0 + \frac{2}{3}\tau$ can initiate a rollback proof γ_i. v_j groups T_l^0 with other failed cross-shard transactions whose sender's shard is also S_i and uses them to generate the set of Rollback transactions for the rollback proof. M_i is the Multisig account to pay for all Rollback transactions. A rollback proof γ_i is denoted as $\gamma_i := (\mathcal{R}_i, H_j, \sigma_j)$. H_j denotes the current block height of S_j, and $\mathcal{R}_i = \{((M_i \rightarrow u_1^s), T_1^0), ((M_i \rightarrow u_2^s), T_2^0), \ldots\}$ denotes the set of Rollback transactions. v_j signs $\mathcal{R}_i$ and broadcasts $\mathcal{R}_i$ to other validators in S_j. Other validators also sign $\mathcal{R}_i$ to generate their partial signatures and broadcast them. If v_j receives at least $2f + 1$ partial signatures, it combines the partial signatures and generates the joint signature σ_j. v_j then broadcasts γ_i to S_i. If S_i receives γ_i, any honest validator in S_i can check if σ_j is valid using pk_j and if $H_j{\geq}H_i^0{+}\frac{2}{3}\tau$. If both are valid, L_i is faulty. Any honest validator in S_i can initiate a view change process to re-elect a new leader L_i' for S_i [7, 8] through an mPBFT process. Note that the joint signature is not required in this leader re-election process. Finally, L_i' proposes an mBlock containing $\mathcal{R}_i$ with other transactions and appends it to the blockchain of S_i after a new mPBFT consensus.

In Case 2, the PC state must also be rolled back. The rollback proof for this case is also denoted as $\gamma_i := (\mathcal{R}_i, H_j, \sigma_j)$. Any honest validator in S_j who observes that $H_j \geq H_i^0 + \frac{2}{3}\tau$ initiates an mPBFT to generate the joint signature σ_j on $\mathcal{R}_i$. A new leader L_j' for S_j is reselected through an extra intra-shard consensus. L_j' decreases the balance of M_j on PC by the sum of the transaction values in $\mathcal{R}_i$. L_j' then broadcasts γ_i to S_i. Any honest validator in S_i initiates a re-election process to select a new mPBFT leader L_i' for S_i. L_i' then verifies if σ_j is valid and if $H_j \geq H_i^0 + \frac{2}{3}\tau$. If valid, L_i' increases the balance of M_i on PC by the sum of the transaction values in $\mathcal{R}_i$. Finally, L_i' commits $\mathcal{R}_i$ to the blockchain of S_i through a new mPBFT consensus and collects the respective joint signature, etc.

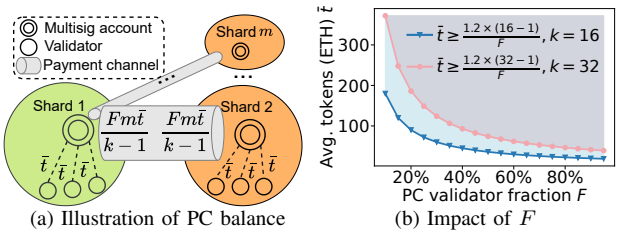
In Case 3, L_i and L_j cannot both benefit from the collusion, as they are on opposite sides of the PC. The total balance of PC remains the same as when the channel was initially opened if no new deposits are made [16]. If Case 3 occurs, the rollback scheme is the same as in Case 1.

E. Adaptation to UTXO Model

Aside from the account model, ShardBridge can also adapt to the UTXO model for accounting [18]. All inputs to a transaction in the UTXO model must be unspent tokens from the outputs of previous transactions. The key differences are as follows: In the reconfiguration phase, each shard can create its Multisig account through the scripts of Bitcoin (P2SH, P2MS), or BlueWallet [18, 44, 45]. To adapt to the UTXO model, we use the PC in Lightning Network [15] for ShardBridge. To assign UTXOs to shards, we use the method in [46] to reduce the cross-shard transaction percentage based on the past transaction pattern. In the consensus phase, we still use mPBFT and mBlock. Only that mBlock now records the UTXOs as the state of a shard, instead of the accounts. A cross-shard transaction may involve multiple input shards and output shards, where the input shards are those containing the input UTXOs of this transaction, and the output shards are those containing the output UTXOs of this transaction. To process such transactions in ShardBridge, the leader of each input shard must create an S-type transaction from the original input UTXO to the newly generated UTXO with the Multisig account of its shard as the output address. The input UTXOs are locked in the input shards. Next, each leader must create corresponding R-type transactions using the UTXOs of Multisig accounts of the output shards as input and send these transactions to the output shards. Once the leaders of all related output shards receive the R-type transactions, they start to commit them in their respective shards through mPBFT. As long as one input shard refuses to create or send the R-type transaction, the transaction is considered failed, and the rollback scheme will be initiated to ensure atomicity.

F. Scalability Analysis

The scalability of ShardBridge relies on validators collaborating to create and deposit into Multisig accounts. The average tokens contributed by validators correlate with the number of shards and the number of participating validators. As shown in Fig. 6(a), assume the average tokens contributed by validators



(a) Illustration of PC balance (b) Impact of F

Fig. 6: Scalability analysis of PC in ShardBridge

is $\bar{t}$, shard size is m, shard number is k, the average amount of tokens transferred in on-chain transactions is a (average transaction tokens transferred on Ethereum is 1.2 ETH [47] as of 04/05/2025), and the fraction of validators contribute to the PC construction is F. Since each Multisig is connected to $k-1$ other shards, the balance on each side of the PC should be $\frac{Fm\bar{t}}{k-1}$. The balance on one side of the PC should support transactions from all validators, requiring $\frac{Fm\bar{t}}{k-1} \geq am$, which gives $\bar{t} \geq \frac{a(k-1)}{F}$. Fig. 6(b) shows $\bar{t}$ with respect to F, when k=16 and k=32. As more validators join a PC, $\bar{t}$ decreases. When $F=\frac{2}{3}$ (worst case), $\bar{t}$ is at around 1.6 ETH/shard and 1.7 ETH/shard, when k=16 and k=32, respectively. This is an economic improvement over Ethereum [28], where each validator assigned for a slot must stake 32 ETH/shard for consensus (Ethereum network is not sharded). ShardBridge offloads the cross-shard transaction processing to the 2PC protocol if there is insufficient capacity in PCs as introduced in §IV-A. In future work, we plan to optimize and incentivize the token amount deposited from validators in each shard, such as introducing a fee model that rewards validators based on the cross-shard transactions they process. This should further reduce $\bar{t}$, as the profit gained by validators also increases.

V. Security Analysis

A. Shard-Size Security

We prove the safety and liveness of ShardBridge through the following theorems. Similar to other sharding consensus protocols [9, 13, 30, 36], we first define the safety of ShardBridge as that honest validators agree on the same valid block in each round. Liveness in ShardBridge ensures each proposed mBlock is eventually committed or aborted.

Theorem 1. ShardBridge *achieves safety if each shard has no more than* $f < \frac{m}{3}$ *malicious validators.*

Proof. A shard is collectively honest only when it has less than $\frac{1}{3}m$ malicious validators, and the mPBFT consensus protocol requires a shard size of $m = 3f + 1$ to provide a valid consensus. The mPBFT for generating mBlocks guarantees the validity of the blocks proposed by each shard. Similar to [27, 30], the shard security is modeled as a random sampling problem using the cumulative hypergeometric distribution: $\Pr[X \geq \lfloor \frac{m}{3} \rfloor] = \sum_{i=\lfloor \frac{m}{3} \rfloor}^{m} \frac{\binom{\mathcal{F}\cdot\nu}{i}\binom{\nu-\mathcal{F}\cdot\nu}{m-i}}{\binom{\nu}{m}}$, where X represents a random variable denoting the number of corrupt validators in the shard, ν is the total number of validators in the network, $\mathcal{F}$ is the fraction of validators controlled by a Byzantine adversary in the network. Each shard needs to maintain a reasonable number of validators to keep its failure probability

negligible. i.e. $\Pr[X \geq \lfloor \frac{m}{3} \rfloor] \approx 6.82e^{-6}$ when $\nu = 600$, $m = 100$ and $\mathcal{F} \cdot \nu = 100$, a shard failure in 401.7 years with 24h epochs. In addition, the joint signature produced from mPBFT cannot be forged or modified. This is because the TSS ensures the unforgeability of the partial signatures from the majority of the parties [24, 25]. Thus, honest validators in ShardBridge should always agree on the newly generated mblocks, which ensures the shard security. □

Theorem 2. ShardBridge *achieves liveness if there is no more than* $f < \frac{m}{3}$ *malicious validators in each shard.*

Proof. All system messages are secured against forgery through digital signatures, and validators are connected by a partially synchronous network. If an honest leader proposes a valid mBlock, it can be downloaded by the honest validators by broadcasting the message in the process of mPBFT. Based on Theorem 1, each shard agrees on the same block in each round. If the receiver shard does not receive the R-type transactions on time, at least one validator will initiate the rollback scheme to roll back false transactions. Thus, no malicious validators can indefinitely prevent the consensus. □

B. Epoch Security

The security of a single shard does not fully reflect the failure probability of the whole sharding system. The system is considered to have failed if at least one shard fails. According to the cumulative hypergeometric distribution function similar to [13, 30], we formulate the upper bound of the probability for the system failure as: $Pr[Failure_s] = 1 - \left[1 - \sum_{i=\lfloor m/3 \rfloor}^{m} \frac{\binom{\mathcal{F}\cdot\nu}{i}\binom{\nu-\mathcal{F}\cdot\nu}{m-i}}{\binom{\nu}{m}}\right]^k < 2^{-\lambda}$. The system failure can be very negligible by adjusting the shard size m and the number of shards k, where λ is the security parameter for epochs in ShardBridge (detailed in §VI-B).

C. Shard Leader Randomness

Shard leaders manage mPBFT consensus and share joint signatures for cross-shard transactions. In ShardBridge, an honest leader is not always guaranteed at the start of each epoch. While a dishonest leader might be selected, the mutual consensus remains unaffected, since the shard is collectively honest as proven in Theorem 1. The threat model specifies that the adversary in each shard can control less than $\frac{1}{3}$ of its validators. The probability of an adversary controlling i consecutive leaders without excluding the previous leader is bounded by $Pr[X = i] = \left(\frac{1}{3}\right)^i < 2^{-l}$, where l is the security parameter for leader selection, i.e. for $l = 5$, the adversary has to control at least 4 consecutive leader selections.

D. Payment Channel Security

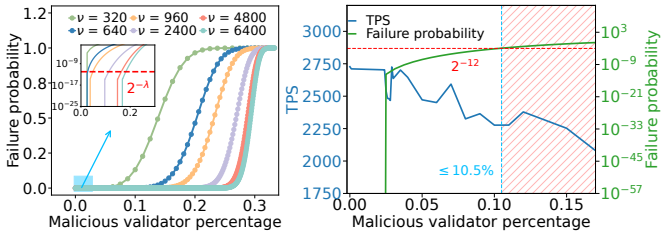
If any cases described in §IV-D occur, it can eventually reduce the throughput of the shards and trigger the rollback scheme. However, leaders are economically disincentivized from exhibiting such malicious behaviors for the following reasons: **First**, even though leaders can be malicious in ShardBridge, the motivation of leaders embezzling the received payment for the Multisig account is minimal. Tokens in the Multisig account can only be redeemed with the consent of all validators who created that account. This means that the leaders are not able to use the money after embezzlement as long as there is a majority of honest validators in the shards, which is proved in Theorem 1. **Second**, if either leader of the PC refuses to update the balance, the most recent balance proof is recorded on-chain once the PC is closed. The malicious party will forfeit its share of the tokens held in the Multisig account. Thus, PC adds an extra layer of security by connecting the shards in ShardBridge. **Third**, existing works [9, 10] incentivize an individual intermediary to handle cross-shard transactions and require them to have accounts across different shards. In contrast, ShardBridge requires validators within the same shard to collectively create a Multisig account to represent their shard and contribute tokens to this account. Therefore, the Multisig accounts in ShardBridge hold a larger amount of tokens compared to single brokers as in [10]. In addition, since PCs significantly accelerate the confirmation of cross-shard transactions, which ensures a faster return on the intra-shard fee, each validator is self-motivated to create the Multisig for their shards. Thus, validators in ShardBridge are incentivized to manage a higher volume of cross-shard transactions, which reduces malicious behaviors and enhances security.

VI. Performance Evaluation

A. Implementation and Evaluation Setup

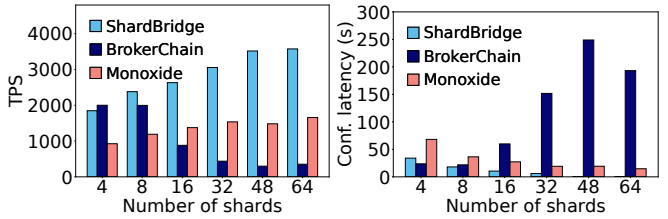
To evaluate the performance of ShardBridge, we implemented a prototype of ShardBridge based on an open-sourced blockchain testbed BlockEmulator [48] in Go language. Each hardware for simulation is equipped with 9-core processors, 32GB of RAM, and a 10 Gbps network link. The TSS scheme was implemented using the TCRSA library [49]. The validators in each shard are connected via a P2P network. Each validator runs the same protocol to process the incoming transactions without a third party. We sampled 1,000,000 Ethereum transactions from block height 1,000,000 to block height 1,999,999 in chronological order [50]. We ran 30 times for each setting to average out the network noise.

We used the following evaluation metrics to assess ShardBridge: transactions per second (TPS), confirmation latency (s) (CL), queue size, and injection size. TPS is the transactions processed divided by the processing time. CL is the average time required for a transaction to be confirmed. Note that high TPS does not necessarily result in low CL, or vice versa. Because TPS is how many transactions the system handles in a given time frame, whereas CL is about how fast each transaction is processed. The queue size represents the number of transactions in the transaction pool, reflecting how quickly transactions can be accumulated. The injection size represents the number of transactions injected into the transaction pool, reflecting how quickly transactions can be injected. We examine the impact of the following base parameters (default values in parentheses): (1) number of shards (k=16), (2) network size (ν=160), and (3) transaction size (200,000), the total



(a) Impact of network size ν (b) Impact of malicious validator percentage

Fig. 7: (a) Analysis of failure probability for varying malicious validator percentage across different ν from 320 to 6400, (b) TPS and failure probability for varying malicious validator percentage with $k = 16$.



(a) Number of shards vs. TPS (b) Number of shards vs. CL

Fig. 8: Impact of number of shards

transaction volume sent. We compare ShardBridge with two state-of-the-art sharding protocols:

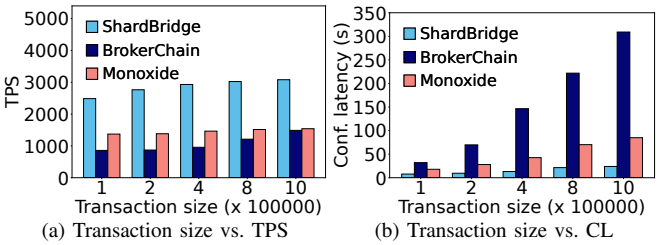
- Monoxide [2]: The sender shard derives and forwards the Relay transactions for cross-shard transactions through a dual-stage transaction handling mechanism.
- BrokerChain [10, 51]: A predetermined number of brokers who have accounts in multiple shards handle the cross-shard transactions for all shards. Each cross-shard transaction is converted into two types of transactions (Type1 and Type2).

B. Evaluation Results

We first analyze how malicious validator percentages affect ShardBridge, then examine parameter impacts, and finally evaluate queue and injection sizes across the three protocols.

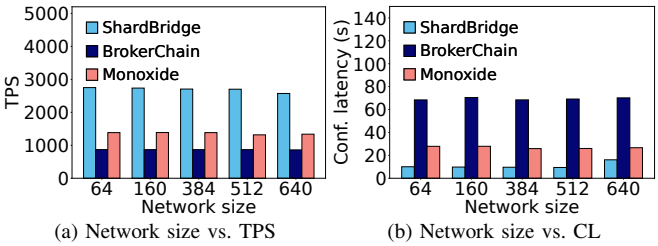
Impact of Malicious Node: Fig. 7 exhibits the impact of malicious validator percentage. Fig. 7(a) illustrates the failure probability of ShardBridge with varying malicious validator percentages. As the network size increases, the failure probability decreases. To maintain the system failure bounded by $2^{-\lambda}$ (introduced in §V-B) with a certain percentage of malicious validators, a large network size is necessary to ensure a negligible failure probability. Fig. 7(b) illustrates the performance of the ShardBridge in terms of TPS and failure probability with varying malicious validator percentages. In the implementation, a malicious validator actively participates when elected as a leader by generating and broadcasting false blocks to disrupt the consensus. As the malicious validator percentage increases, the TPS of ShardBridge decreases and the failure probability increases. This is because the false blocks proposed by malicious validators are aborted through the mPBFT consensus. We set the security parameter λ as 12 in Fig. 7(b), which indicates a system failure probability smaller than 2^{-12}, i.e., one failure in ∼157.2 years for two-week epochs. To satisfy $Pr[Failure_s] < 2^{-12}$, the malicious validator percentage needs to remain below 10.5%.

Impact of Number of Shards k: Fig. 8 illustrates the performance of the three protocols in terms of TPS and CL



(a) Transaction size vs. TPS (b) Transaction size vs. CL

Fig. 9: Impact of transaction size



(a) Network size vs. TPS (b) Network size vs. CL

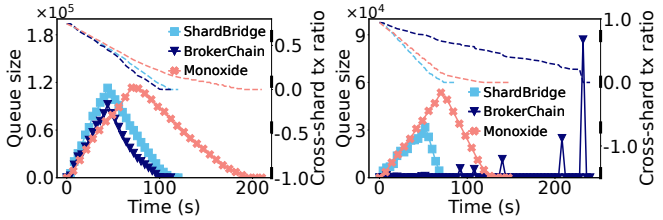
Fig. 10: Impact of network size, the shard size can be computed by $m = \nu/k$

with respect to k. ShardBridge outperforms the other two protocols as k grows for both metrics. In Fig. 8(a), the TPS initially increases almost linearly with k, but then slowly converges once k exceeds 32 for both ShardBridge and Monoxide. In Fig. 8(b), the CL decreases with k for both ShardBridge and Monoxide. As k increases, more shards can process transactions in parallel, resulting in a positive effect on TPS and CL. The convergence of TPS is because TPS is constrained by block size and the injection speed, and thus cannot increase indefinitely. We observe a reverse trend of TPS and CL for BrokerChain. The key difference between BrokerChain and the other two protocols is that it employs brokers to handle the cross-shard transactions. The brokers soon become the bottleneck of processing transactions when the number of cross-shard transactions exceeds their processing ability, which occurs as k increases.

Impact of Transaction Size: Fig. 9 shows the performance of the three protocols with respect to transaction size. ShardBridge performs better than the other two protocols in both metrics. Additionally, both TPS and CL increase with the transaction size for all three protocols. The increase of TPS is intuitive as more transactions result in more blocks filled with transactions, given the same time frame. The increase in CL is due to the growing congestion in the transaction pool, resulting in a longer transaction waiting time.

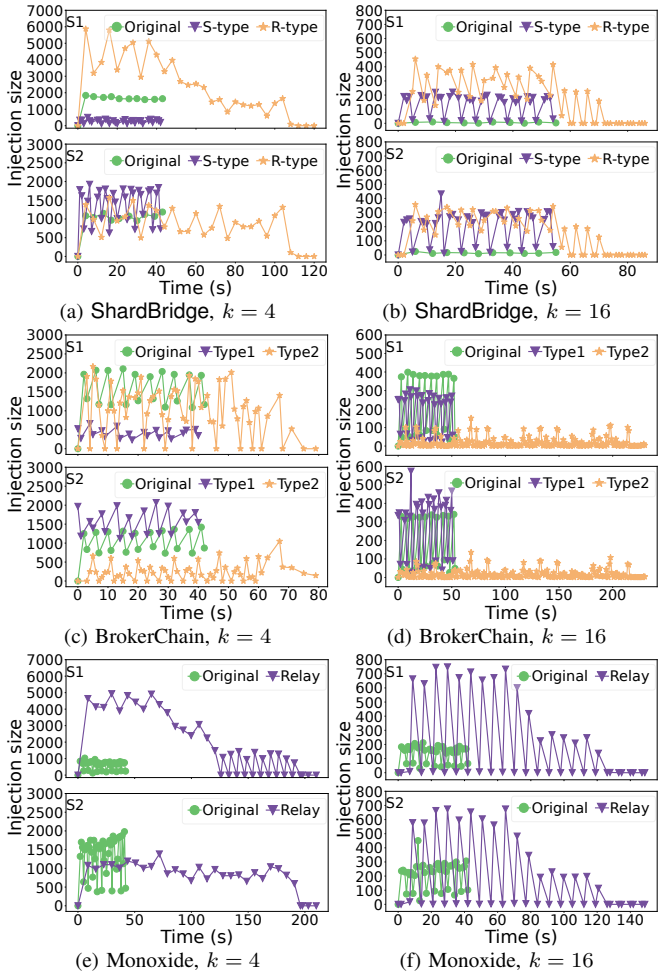
Impact of Network Size ν: Fig. 10 shows the performance of the three protocols concerning ν. We fixed k to 16 and increased m in the implementation. We observe that ShardBridge outperforms the other two protocols as ν increases for both metrics. In Fig. 10(a), the TPS slightly decreases with ν in all three protocols. As ν grows, the increased number of validators in each shard requires more time to reach consensus, leading to a decrease in TPS. In Fig. 10(b), CL increases with network size for all three protocols. This is because a larger network size causes packing transactions to take longer to be confirmed by validators, leading to a longer latency.

Queue Size and Cross-shard Transaction Ratio: Fig. 11 shows the queue size and cross-shard transaction ratio of three



(a) $k = 4$ (b) $k = 16$

Fig. 11: Comparison results for Time(s) vs. Queue size and Cross-shard transaction ratio with (a) $k = 4$ and (b) $k = 16$



(a) ShardBridge, $k = 4$ (b) ShardBridge, $k = 16$

(c) BrokerChain, $k = 4$ (d) BrokerChain, $k = 16$

(e) Monoxide, $k = 4$ (f) Monoxide, $k = 16$

Fig. 12: Comparison results for Time(s) vs. Injection size

protocols relative to the time elapsed with two k values. All three protocols show decreasing cross-shard ratios over time, with ShardBridge declining faster, indicating faster cross-shard handling. In Fig. 11(a), the queue size initially increases over time and then decreases until it reaches zero for all three protocols. The queue size of BrokerChain decreases faster than the other two protocols, resulting in quicker queue clearance. This aligns with Fig. 8(a), where BrokerChain shows a relatively higher TPS when k=4. In Fig. 11(b), the queue size of ShardBridge accumulates slowly and drops faster than Monoxide. Note that BrokerChain does not accumulate transactions early and shows a peak later than the other two protocols, because it needs a longer processing time of cross-shard transactions for brokers.

Injection Size: Fig. 12 illustrates how injection size changes over time for the three protocols. We observe that original transactions are injected within approximately 50 seconds for all three protocols, while S-type, Type2, and Relay transactions are injected throughout the entire processing duration. This is because the latter transactions are created after the original transactions are processed, and transactions are processed in the order they are received. Fig. 12(a), Fig. 12(c) and Fig. 12(e) show the injection size change for S_1 and S_2 when $k = 4$. We observe that S_1 and S_2 have different injection sizes of the original transactions for all three protocols. This is because the distribution of the original transactions among different shards is determined by the transaction input and the partition result. Fig. 12(b), Fig. 12(d) and Fig. 12(f) show the injection size change for S_1 and S_2 when k=16. We observe that the number of original transactions in both shards decreases for all three protocols. Such decreases occur because the workload distributed to each shard is reduced as k increases. Comparing k=16 to k=4, we observe that the total time to inject R-type transactions decreases for ShardBridge, Relay transactions decrease for Monoxide, but Type2 transactions increase for BrokerChain. BrokerChain takes a longer time to inject different types of transactions compared to the other two protocols when k=16, and it takes longer to process the transactions, resulting in a lower TPS for BrokerChain. However, ShardBridge utilizes PC to reduce communication for broadcasting the R-type transactions, and thus uses less time to finish transaction injection compared to Monoxide.

VII. Conclusion

In this paper, we developed a cross-shard protocol ShardBridge to handle cross-shard transactions in a sharded blockchain. We first studied PC construction between shards to eliminate extra cross-shard communication. We also designed mPBFT with a threshold signature scheme for batch signing, enhancing cross-shard transaction efficiency. We further designed a rollback scheme to ensure the atomicity and security of the ShardBridge. Extensive evaluations demonstrate that ShardBridge outperforms other sharding protocols in transaction throughput and confirmation latency.

Validator incentivization: Validators in ShardBridge are naturally incentivized to form PCs for their shards to ensure themselves a faster return of intra-shard transaction fees, such as by introducing a reward model for participating validators in the PCs. We aim to further optimize and incentivize the deposit requirements for validators to construct the Multisig account of their respective shard. This optimization aims to lower the average deposit per validator, as validator rewards are expected to grow with the number of shards.

Smart contract abstraction: ShardBridge focuses on PCs for facilitating payments. We believe this is sufficient for payment conduct in our current design. As future work, we will extend our protocol with state channels to support smart contract execution and enable applications like multi-sender and multi-receiver transactions.

REFERENCES

[1] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 123–140.

[2] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *USENIX NSDI*, 2019, pp. 95–112.

[3] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *SIGSAC*. ACM, 2016, pp. 17–30.

[4] Hyperledger Fabric Team, "Hyperledger Fabric: What is Hyperledger Fabric?" 2022. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2

[5] Z. Team, "ZILLIQA 2.0 Whitepaper," 2025. [Online]. Available: https://www.zilliqa.com/

[6] Tendermint Team, "Tendermint: BFT Consensus for Blockchains," 2024. [Online]. Available: https://tendermint.com/

[7] M. Du, Q. Chen, and X. Ma, "Mbft: A new consensus algorithm for consortium blockchain," *IEEE Access*, vol. 8, pp. 87 665–87 675, 2020.

[8] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: A scalable and decentralized trust infrastructure," in *IEEE/IFIP DSN*, 2019, pp. 568–580.

[9] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and 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.

[10] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding," in *IEEE INFOCOM*, 2022, pp. 1968–1977.

[11] J. Zhao, J. Yu, and J. K. Liu, "Consolidating hash power in blockchain shards with a forest," in *Information Security and Cryptology: 15th International Conference, Inscrypt 2019, Nanjing, China, December 6–8, 2019, Revised Selected Papers 15*. Springer, 2020, pp. 309–322.

[12] T. Rajabi, A. A. Khalil, M. H. Manshaei, M. A. Rahman, M. Dakhilalian, M. Ngouen, M. Jadliwala, and A. S. Uluagac, "Feasibility analysis for sybil attacks in shard-based permissionless blockchains," *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 4, pp. 1–21, 2023.

[13] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *IEEE INFOCOM*, 2021, pp. 1–10.

[14] S. Jiang, J. Cao, C. L. Tung, Y. Wang, and S. Wang, "Sharon: Secure and efficient cross-shard transaction processing via shard rotation," in *IEEE INFOCOM*, 2024, pp. 2418–2427.

[15] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[16] E. Foundation, "What is the raiden network," 2019. [Online]. Available: https://raiden.network

[17] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

[19] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox *et al.*, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, vol. 4, no. 220, p. 32, 2016.

[20] B. Markus and J. Palmer, "Dogechain whitepaper," 2024. [Online]. Available: https://dogechain.dog/DogechainWP.pdf

[21] Y. D. Y. Frankel and Y. Desmedt, "Threshold cryptosystems," in *CRYPTO*, vol. 89, 1998, pp. 307–315.

[22] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1179–1194.

[23] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *IEEE SP*, 2019, pp. 1051–1066.

[24] H. Yu and H. Wang, "Elliptic curve threshold signature scheme for blockchain," *Journal of Information Security and Applications*, vol. 70, p. 103345, 2022.

[25] C. Stathakopoulou and C. Cachin, "Threshold signatures for blockchain systems," *Swiss Federal Institute of Technology*, vol. 30, p. 1, 2017.

[26] J. Xu, Y. Ming, Z. Wu, C. Wang, and X. Jia, "X-shard: Optimistic cross-shard transaction processing for sharding-based blockchains," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[27] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 931–948.

[28] Ethereum Foundation, "Ethereum roadmap: Danksharding," 2024. [Online]. Available: https://ethereum.org/en/roadmap/danksharding

[29] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.

[30] M. Li, Y. Lin, J. Zhang, and W. Wang, "Cochain: High concurrency blockchain sharding via consensus on consensus," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[31] F. Cheng, J. Xiao, C. Liu, S. Zhang, Y. Zhou, B. Li, B. Li, and 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.

[32] J. XU, "xshard," 2024. [Online]. Available: https://github.com/myl7/xshard/

[33] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[34] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer pbft consensus for blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1146–1160, 2020.

[35] S.-E. Huang, S. Pettie, and L. Zhu, "Byzantine agreement in polynomial time with near-optimal resilience," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, 2022, pp. 502–514.

[36] W. Chen, D. Xia, Z. Cai, H.-N. Dai, J. Zhang, Z. Hong, J. Liang, and Z. Zheng, "Porygon: Scaling blockchain via 3d parallelism," in *IEEE ICDE)*, 2024, pp. 1944–1957.

[37] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, 1999, pp. 120–130.

[38] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi, "An in-depth look of bft consensus in blockchain: Challenges and opportunities," in *Proceedings of the 20th international middleware conference tutorials*, 2019, pp. 6–10.

[39] F. Ulfo, "Multisig transactions with gnosis safe," April 2020. [Online]. Available: https://medium.com/gauntlet-networks/multisig-transactions-with-gnosis-safe-f5dbe67c1c2d

[40] D. Nnam, "Introducing argent multisig," March 2020. [Online]. Available: https://www.argent.xyz/blog/introducing-argent-multisig/

[41] "Totalsig blog," 2024. [Online]. Available: https://www.totalsig.com

[42] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[43] A. Mizrahi and O. Rottenstreich, "Blockchain state sharding with space-aware representations," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1571–1583, 2020.

[44] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," *Designs, Codes and Cryptography*, vol. 87, no. 9, pp. 2139–2164, 2019.

[45] BlueWallet, "Bluewallet multisig wallet," 2024. [Online]. Available: https://bluewallet.io/multisig-wallet/

[46] L. N. Nguyen, T. D. Nguyen, T. N. Dinh, and M. T. Thai, "Optchain: optimal transactions placement for scalable blockchain sharding," in *IEEE ICDCS*, 2019, pp. 525–535.

[47] C. Community, 2024. [Online]. Available: https://cryptoquant.com/community/discover

[48] HuangLab-SYSU, "Block emulator," 2024. [Online]. Available: https://github.com/HuangLab-SYSU/block-emulator

[49] Niclabs, "Tcrsa," 2024. [Online]. Available: https://github.com/niclabs

[50] X. Team, "Xblock," 2024. [Online]. Available: https://xblock.pro/xblock-eth.html

[51] H. Huang, G. Ye, Q. Chen, Z. Yin, X. Luo, J. Lin, T. Li, Q. Yang, and Z. Zheng, "Blockemulator: An emulator enabling to test blockchain sharding protocols," *arXiv preprint arXiv:2311.03612*, 2023.