Logos: Robust Sharding Blockchain With Fast Processing and Optimal Cross-Shard Overhead

Yizhong Liu[†♮], Boyu Zhao[†], Yuxuan Hu[†], Haojun Tan[†], Feiang Ran[†], Andi Liu[†], Zhuocheng Pan[†], Yuan Lu[§], Song Bian[†], Jianwei Liu[†*], Zhenyu Guan[†*]

[†]*School of Cyber Science and Technology, Beihang University,* [§]*Institute of Software, Chinese Academy of Sciences,*
[♮]*Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing,*

Email:{liuyizhong,boyuzhao,yuxuanhu,tanhaojun,ranfeiang,liuandi,
zhuochengpan,sbian,liujianwei,guanzhenyu}@buaa.edu.cn, luyuan@iscas.ac.cn

Abstract

Sharding blockchains improve scalability significantly by partitioning the network into shards. Due to the substantial fraction of cross-shard transactions (CSTXs) related to multiple shards, cross-shard transaction processing (CSTP) is critical to the system security and performance. However, existing CSTP methods suffer from limited robustness caused by invalid CSTXs flooding by malicious nodes and impose high overhead, especially in asynchronous networks.

We present Logos, a robust sharding blockchain with fast CSTP and optimal cross-shard overhead. Logos adopts a novel robust broadcast-transmission-agreement pattern. Each input shard only invokes a new designed broadcast primitive to generate input availability states. After the states are delivered to involved shards by an innovative parallel single-to-single transmission mechanism, valid CSTXs are committed via an agreement protocol while invalid ones are discarded. Logos is proven to achieve an optimal intra-shard overhead for valid CSTP and lower overhead for invalid CSTP. Besides, Logos achieves reliable transmission with optimal cross-shard overhead. Experiments conducted on 1000 AWS-EC2 nodes across 4 regions demonstrate that Logos realizes 50% latency compared to the baseline (Kronos, NDSS'25) and a peak throughput of 132.8 ktx/sec. Besides, the cross-shard network usage of Logos impressively remains only $1/210$ of Kronos. Under malicious flooding attacks, Logos maintains $2.86\times$ the throughput of Kronos, demonstrating strong robustness.

1 Introduction

Sharding blockchains are promising methods to enhance blockchain scalability [5, 26, 29, 34, 41, 48, 53, 54, 57].

Different from traditional blockchain systems, where all participants store full blockchain states and run global consensus protocols to process transactions, sharding blockchains divide nodes in the network into a few smaller groups called *shards*. Each shard handles only a subset of transactions and manages an isolated part of blockchain states. This division reduces the storage, computing, and communication overhead by decreasing the need for each node to interact with all other nodes or store the full states [37]. Sharding blockchains enable higher transaction throughput by scaling the number of participants and parallel shards. This approach advances blockchains into practical, distributed infrastructures of large-scale networks for emerging domains such as financial services [45], decentralized identity [42], and Web 3.0 [27, 32]. Notably, sharding blockchains have been already implemented in industry-grade systems such as Harmony [1], Nightshade [47], and Polkadot [2].

In sharding blockchain systems, *cross-shard transaction processing* (CSTP) requires coordination among multiple shards, significantly impacting system security and performance. Cross-shard transactions (CSTXs)—whose inputs or outputs are managed by different shards due to state isolation—come to dominate the workload as the shard number increases, surpassing 99% of all transactions in a 16-shard system [54]. A fundamental security goal of CSTP is *atomicity*, which guarantees that either every involved shard commits a valid CSTX or every shard rejects an invalid CSTX. Existing schemes [5, 13, 24, 25, 29, 54] achieve weak atomicity, relying on network timing assumptions or failing to resist attacks by malicious leaders or clients that may falsely compute or transmit messages. Specifically, CSTP proceeds in two phases: intra-shard (IS) consensus and cross-shard (CS) transmission. Each shard typically uses a Byzantine Fault Tolerance (BFT) protocol [4, 9] as the IS consensus to confirm input availability states of a CSTX, or commit a CSTX, where *IS-overhead* measures the BFT invocation times and corresponding communication and computation cost. Most current designs adopt a two-phase commit (2PC) [5, 24, 29] approach, incurring a $2k\mathcal{B}$ IS-overhead (k denotes the involved shard number and $\mathcal{B}$ denotes the cost of one BFT execution). For cross-shard transmission, shards exchange input availability states or CSTX commitment proofs and ensure *reliability* where honest nodes can eventually receive these cross-shard messages. *CS-overhead*, which denotes the total cross-shard

*Corresponding Author: Jianwei Liu and Zhenyu Guan

Table 1: Comparison between Logos and state-of-the-art BFT-based sharding blockchain systems.

Scheme	Applicability to Async. Networks	Atomicity*	Robustness$^{\lozenge}$	IS-Overhead† (valid CSTX)	IS-Overhead† (invalid CSTX)	CS-Overhead$^{\natural}$ for Reliability§ (for b transactions)
Omniledger [29]	✗	✔✗	✗	$2k\mathcal{B}$	$2k\mathcal{B}$	$O(n^2 \cdot b\lambda)$
Rapidchain [54]				$k\mathcal{B}$	$2k\mathcal{B}$	
Chainspace [5]				$2k\mathcal{B}$	$2k\mathcal{B}$	
ByShard [24]				$2k\mathcal{B}$	$2k\mathcal{B}$	
AHL [13]				$(2k+3)\mathcal{B}$	$(2k+3)\mathcal{B}$	
Kronos **[34]**	✔	✔	✗	$k\mathcal{B}$	$2x\mathcal{B}$	$O(n^2 \cdot \frac{b\lambda}{n})$
Logos	✔	✔	✔	$k\mathcal{B}$	$x\mathcal{C}^{*}$	$O(n \cdot \frac{b\lambda}{n})$

"✓", "✔", "✗" represent that a property is satisfied, partially satisfied, and not satisfied, respectively.

$\star$ Weak atomicity means that the protocols rely on honest leaders/clients or timing assumptions (Please refer to Definition 1 and Appendix A).

$\lozenge$ Robustness ensures stable CSTP capacity under malicious nodes flooding attacks by submitting a large number of invalid CSTXs (Please refer to Definition 2).

§ Reliability ensures all honest nodes in receiver shard can eventually receive the same cross-shard message delivered by the sender shard (Please refer to Definition 6).

† k: total number of involved shards in a CSTX. $x(<k)$: the number of input shards which has executed/locked inputs for an invalid CSTX (e.g., $k=4$, $x=2$).

$*$ $\mathcal{B}$: BFT execution cost. $\mathcal{C}$: Broadcast execution cost. $\mathcal{M}$: Agreement execution cost (e.g., MVBA). In asynchronous BFT, $\mathcal{B}=\mathcal{C}+\mathcal{M}$, $\mathcal{C}<\mathcal{B}$.

$\natural$ n: the number of nodes in each shard (n^2 implies the CS broadcast). b: tx batch size. λ: the security parameter of the hash function (the output bit length).

communication cost, scales as $O(n^2 b\lambda)$ (with n nodes per shard, b the batch size of transactions, and λ the security parameter of the hash function). More challenging still, in practical *asynchronous networks* where adversaries can delay messages arbitrarily and honest nodes face unpredictable latencies, ensuring atomicity and reliability is substantially more difficult. Notably, Kronos [34] designs an asynchronous sharding blockchain. It leverages an "execution-rollback" mechanism where input shards execute valid CSTXs directly via BFT, reducing IS-overhead to $k\mathcal{B}$. Also, a broadcast-style reliable cross-shard certification reduces CS-overhead to $O(nb\lambda)$.

However, there are still remaining security and performance issues. For invalid CSTXs, each input shard must perform an additional BFT for rollback, resulting in an IS-overhead of $2x\mathcal{B}$ (where x is the affected shard number). This high overhead enables malicious nodes to launch flooding attacks and exhaust computation and communication resources by submitting a large number of invalid CSTXs. The sharding blockchain should keep *robustness*, where the system can preserve stable processing capability for valid CSTXs under such attacks. Moreover, although Kronos employs erasure coding to mitigate transmission costs, its reliance on cross-shard broadcasting in an asynchronous setting still incurs significant CS-overhead. In real-world deployments, nodes in different shards are often located in geographically dispersed regions. Cross-shard transmission therefore requires maintaining costly and limited-bandwidth long links. Cross-shard broadcasting becomes a critical bottleneck to the practical deployment of sharding blockchains. Existing typical sharding blockchains are summarized in Table 1.

Our contributions. We propose Logos, a robust sharding blockchain that realizes fast CSTP and optimal cross-shard overhead, with the following contributions.

Robust and efficient CSTP pattern with optimized intra-shard overhead. We propose a *robust broadcast-transmission-agreement* CSTP pattern. All honest nodes in each input shard first determine the input availability states of CSTXs via the newly designed primitive *cross-state determine reliable broadcast* (csdRBC) instead of an entire BFT. After exchanging states via cross-shard transmission, valid CSTX can be consistently committed, while invalid CSTX can be rejected in the agreement phase. In particular, Logos ensures *atomicity* even in *asynchronous networks*, achieving an optimal IS-overhead of $k\mathcal{B}$ for valid CSTP and optimized IS-overhead $x\mathcal{C}$ for invalid CSTP ($\mathcal{C}$ denotes the cost of csdRBC and is lower than $\mathcal{B}$). Logos ensures *robustness* by preventing malicious nodes from degrading system processing capacity stability through flooding invalid CSTXs.

Reliable cross-shard transmission mechanism with optimal cross-shard overhead in asynchronous networks. We propose a *reliable parallel single-to-single transmission* mechanism. Each honest node in a sender shard only sends a message fragment generated by erasure coding to a *single* receiver, with a threshold signature and a Merkle proof as the certificate produced by the sender shard. The receiver shard then reconstructs the whole message, achieving *cross-shard reliability*. Notably, Logos is proven to realize an optimal CS-overhead of $O(b\lambda)$. Specifically, Logos reduces the cross-shard long link number required for message delivery from $O(n^2)$ in existing designs to $O(n)$, thus eliminating the bottleneck caused by high-overhead cross-shard broadcast in large-scale, real-world deployments of sharding blockchains.

Large-scale implementation of specific constructions. To demonstrate the practical performance of Logos, we implement it using an asynchronous BFT Speeding Dumbo and an MVBA protocol Speeding MVBA [23]. We conduct extensive experiments on Amazon EC2 instances distributed from 4 regions across the globe. The experiment results reveal that

Logos achieves a throughput of 132.8 ktx/sec with a network size of 1000 nodes. We compare Logos with Kronos adopting the same BFT protocol. Logos achieves with only 50% the latency compared to Kronos. Besides, cross-shard network usages of Logos impressively remain only $1/210$ of Kronos. Under malicious flooding attacks, Logos maintains $2.86\times$ throughput over Kronos, demonstrating strong robustness.

2 Challenges and Solutions

Challenge 1: On CSTP pattern—Realizing fast CSTP with *strong atomicity* and *robustness* in *asynchronous* networks. As shown in Figure 1, in 2PC, each input shard invokes two rounds of BFT to determine the input availability states of CSTXs and execute the CSTXs respectively, leading to an IS-overhead of $2k\mathcal{B}$. Kronos decreases the IS-overhead by executing the CSTX directly after one BFT. However, a rollback via a BFT is necessary to realize atomicity for invalid CSTXs (unhappy path in Kronos). In asynchronous networks, CSTP occurs constantly and progresses at different rates across shards, leading to frequent rollbacks. This incurs an IS-overhead of $2x\mathcal{B}$ and undermines robustness under malicious flooding invalid CSTXs. To achieve fast and robust CSTP, the rollback should be avoided. Each involved input shard should first determine the input availability of the CSTX that it manages. If all inputs are available, the transaction is committed as valid; if any input is unavailable, it is rejected as invalid. Meanwhile, the availability states should be certified by the entire shard, considering malicious nodes.

However, ensuring efficient availability determination and preserving atomicity in asynchronous networks is challenging. Trivially utilizing an entire BFT leads to a high IS-overhead of $2k\mathcal{B}$. Alternatively, if each honest node independently votes on CSTX input availability based on its local blockchain state, the resulting availability votes may conflict. The conflicts can be caused by the inconsistent view on current blockchain states or unawareness of conflicting transactions, which may happen in asynchronous networks. Consequently, it becomes difficult to assemble a consistent certificate of availability votes from an input shard, undermining the atomicity.

High-level technical overview of solution to Challenge 1. As shown in the right part of Figure 1, we propose the *broadcast-transmission-agreement* pattern to realize strong atomicity and robustness in asynchronous networks. We design a new *cross-state determine reliable broadcast* (csdRBC) to collect non-conflicting votes for input availability states in each shard with cost $\mathcal{C}$. We first add the state requirement to solve the conflict caused by inconsistent views on current blockchain states. The proposal on determining a CSTX is attached by the proposer's local block height and each node only starts to vote for the CSTX while the blockchain states are aligned. Besides, we solve the unawareness of conflicting CSTXs by restricting that each honest node only votes for one of the multiple conflicting CSTXs on the height h. Although

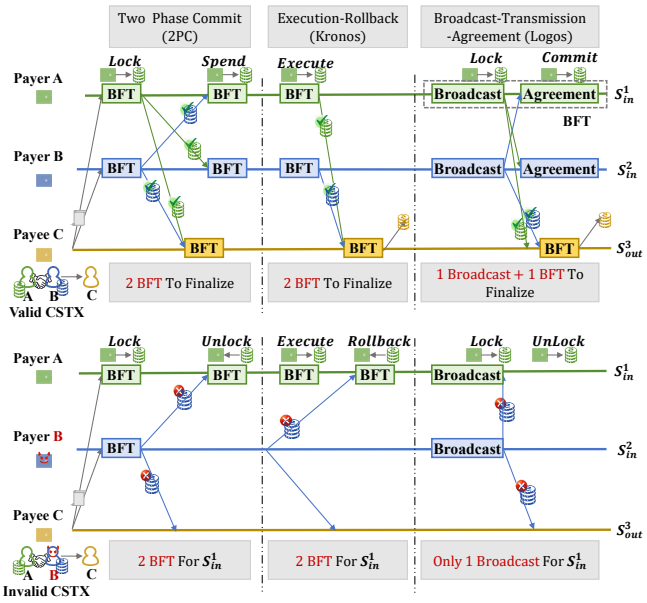


Figure 1: Challenge 1 and our solutions. CSTP workflows are shown from left to right for 2PC, Kronos, and Logos; the upper and lower parts illustrate the processing of valid CSTXs and invalid CSTXs, respectively.

none of the conflicting CSTXs can get $2f+1$ votes probably, all honest nodes can receive all conflicting CSTXs finally and solve the conflicts via common rules such as the hash order in the following csdRBC.

Once the conflicts are solved, we use the classical RBC-style collection to ensure all honest nodes can finally assemble a consistent certificate of availability votes including $2f+1$ signatures from at least $f+1$ honest nodes for non-conflicting CSTXs. Afterwards, each input shard exchanges input availability states and certificates via cross-shard transmission. For valid CSTXs, each input shard only needs to invoke an agreement protocol (e.g., MVBA) with a cost $\mathcal{M}$ to commit the transactions. The total cost for valid CSTP is only $\mathcal{C}+\mathcal{M}$, which is equal to a BFT cost in asynchronous networks. For invalid CSTXs, each input shard can directly discard them and finish the CSTP. So the total cost for invalid CSTP is only $x\mathcal{C}$, strengthening the robustness.

Challenge 2: On cross-shard transmission—Achieving asynchronous *reliability* and *optimal* cross-shard overhead. As shown in the left part of Figure 2, selecting a coordinator in each shard to deliver the message suffers from the loss of reliability in the asynchronous network. A malicious coordinator can refuse to deliver the message, leading to an indefinite wait. As shown in the middle part of Figure 2, the cross-shard broadcast can achieve asynchronous reliability but suffer from a high CS-overhead of $O(n^2 \cdot \frac{b\lambda}{n})$ bits (with erasure coding) and $O(n^2)$ long links. To optimize the overhead, the cross-shard broadcast should be avoided. Each sender only maintains

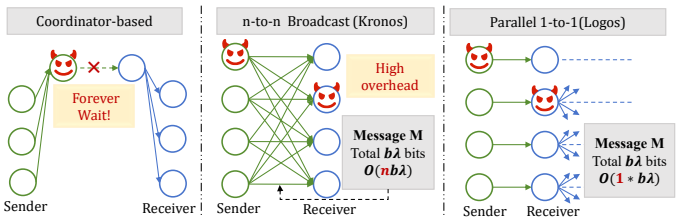


Figure 2: Challenge 2 and our solutions. CS transmission workflows are shown from left to right for coordinator-based, Kronos, and Logos, respectively.

some long links and sends some message fragments to part of the receivers and ensures all receivers can finally decode the original message from fragments in asynchronous networks.

However, ensuring all receivers can finally get the original message in asynchronous networks with an optimal overhead is challenging. Trivially letting each receiver get the message fragments from all senders will lead to a high CS-overhead of $O(n^2 \cdot \frac{b\lambda}{n})$ bits. Alternatively, if each receiver gets message fragments from only a small part of the senders, it cannot verify that the fragments are from honest senders and that the fragments are enough to reconstruct the original message. Consequently, it becomes difficult to make receivers get the least message fragments from senders and correctly reconstruct the original message, achieving asynchronous reliability and optimal CS-overhead simultaneously.

High-level technical overview of solution to Challenge 2. As shown in the right part of Figure 2, we propose the *reliable parallel single-to-single transmission* mechanism. All honest senders can generate the message fragments via erasure coding and the corresponding certificate jointly. The certificate contains a threshold signature for a Merkle root of the Merkle tree constructed by all fragments. In the proposed CSTP pattern, the input shards perform as the sender shards in the transmission after invoking csdRBC, so the message fragments and certificates can be elegantly generated in csdRBC. Hence, the fragments can be verified by the signature and the Merkle proof. After that, each sender only needs to deliver a single fragment with the certificate to a single receiver to realize the least message fragment transmission.

For the message fragments delivered, honest receivers can verify the incorrect fragments from malicious senders and prevent the transmission of f wrong fragments. Although f malicious receivers can also not deliver or send wrong fragments, at least $f+1$ correct fragments can be delivered by receivers since $n \geq 3f+1$ holds for each shards. Then all receivers use intra-shard broadcast to get at least $f+1$ correct fragments and reconstruct the message, achieving asynchronous reliability. Notably, the reconstruction can be embedded into the normal execution in asynchronous BFT without extra communication rounds.

3 Building Blocks and System Models

3.1 Building Blocks

Byzantine fault tolerance protocols. For most sharding blockchains, Byzantine fault tolerance protocols (BFT) usually act as intra-shard consensus to realize instant transaction confirmation. In a system of n ($n \geq 3f+1$) nodes in an asynchronous or partially synchronous network, a BFT tolerates up to f Byzantine (malicious) nodes (which can arbitrarily violate the protocol, crash, or vote twice on conflicting proposals). A BFT has the following properties [4, 12, 20, 55, 56].

- **Safety.** If two honest nodes commit proposals p and p' at the same height in their ledgers, it must hold that $p = p'$.
- **Liveness.** If a valid proposal p is submitted to the system, then each honest node will eventually commit p.

Multi-valued validated Byzantine agreement. Multi-valued validated Byzantine agreement (denoted as MVBA) is a kind of commonly used agreement protocol in asynchronous networks. Participating nodes can agree on a value satisfying a global and polynomial-time computable predicate $Q(\cdot)$ known by all of them. An MVBA protocol should satisfy the following properties [3, 8, 19, 40].

- **Termination.** If each honest node takes an input value v with $Q(v) = 1$ then each honest node will output a value v.
- **External-Validity.** If an honest node outputs a value v, then $Q(v)$ is valid for Q.
- **Agreement.** All honest nodes will decide the same value v as output.

Erasure coding. Erasure coding is a data redundancy technique used to enhance reliability and efficiency in distributed systems. By dividing the message into r blocks and adding redundant parity blocks (the total block number is n), it enables recovery from the loss of up to $n-r$ blocks [46]. We use a typical (r,n)-erasure coding scheme ($0 \leq r \leq n$) proposed by Reed-Solomon with the following algorithms.

- **Encode algorithm:** $\mathsf{ECEnc}(\phi, M, n, r) \rightarrow \{M_i\}_n$. A message M is encoded into n blocks $M_1, \cdots, M_n$ and can be reconstructed by r blocks. ϕ denotes the polynomial utilized for coding.
- **Decode algorithm:** $\mathsf{ECDec}(\phi, T) \rightarrow M$. T represents a list which contains r different and correct encoded blocks M_i. After decoding, the original message M can be retrieved.

Merkle tree. A Merkle tree is a data structure that uses hash functions to verify batched data integrity [43]. Each leaf node represents a data block D_i in the data block set $\{D_i\}$ while the nodes higher up are hashes of their children nodes and

the top is the tree root rt. The tree construction is denoted as $\mathsf{TreeCon}(\{\mathsf{Hash}(D_i)\}) \rightarrow (\mathsf{tree}, \mathsf{rt})$. To verify each D_i in the data block set $\{D_i\}$, a Merkle tree with root rt and a proof mproof_i from $\mathsf{Hash}(D_i)$ to rt need to be given. The proof generation for D_i is denoted as $\mathsf{MpfGen}(\mathsf{tree}, D_i) \rightarrow \mathsf{mproof}_i$.

Threshold signature schemes. We use threshold signatures to facilitate aggregation. (t,n)-threshold signature is a tuple of algorithms which involves n entities and at most $t-1$ in them can be corrupted [7, 15, 18]. Each node $\mathcal{P}_i$ can hold a key pair (pk_i, sk_i) initialized by distributed key generation (DKG) [16, 22] and sign on message M via function $\mathsf{ShareSig}(sk_i, M) \rightarrow sig_i$. Each signature share sig_i can be verified by $\mathsf{ShareVer}(M, (i, sig_i)) \rightarrow 0/1$ while t different shares can be aggregated into one signature via function $\mathsf{Combine}(\{sig_i\}) \rightarrow \sigma$. The aggregated signature σ can be verified by a total public key tpk via function $\mathsf{Verify}(\sigma, tpk, M) \rightarrow 0/1$. The threshold signature scheme should satisfy the *unforgeability*, i.e. any polynomial-time adversary cannot forge a valid σ or sig_i (without sk_i).

3.2 System Models

Adversary model. We consider a polynomial-time adversary $\mathcal{A}$ which can control up to $f < \frac{n}{3}$ malicious nodes in each shard S_i with n nodes, i.e., *honest majority* holds for each secure shard. Honest majority is commonly assumed by most sharding blockchains [29, 34, 54], where shard size is configured according to hypergeometric distribution. It requires sufficient candidates with a bounded malicious proportion less than 1/3 and a slightly larger shard size, where Logos can maintain high efficiency. More details of realizing *honest majority* in Logos is shown in Section 7.1. $\mathcal{A}$ can deviate from the protocol operation arbitrarily except for forging the signature of honest nodes.

Network model. The network model could be *asynchronous* or *partially synchronous*. Section 4 describe protocols in asynchronous networks, where messages between two honest nodes may be arbitrarily delayed or reordered, and only eventual reachability is guaranteed. Logos can adjust components for partially synchronous networks (Section 7.2). In partially synchronous networks, there exists an unknown but finite global stabilization time (GST). After GST, messages sent between two honest nodes arrive within a fixed delay.

Transaction model. The transaction models include the account model and the unspent transaction output (UTXO) model. We use the account model for the convenience of description and Logos is also applicable to the UTXO model (Section 7.3). In the account-based model, each shard is responsible for managing a set of accounts and their states. A transaction is structured as: $\mathsf{TX} = (\{input.account\}, \{output.account\}, val, id)$. In a cross-shard transaction $\mathsf{TX}_{\mathrm{cr}}$, $\{input.account\}$ and $\{output.account\}$ can contain multiple input and output accounts managed by different shards.

4 Logos

In this section, we provide the entire illustration of our proposed Logos, a robust sharding blockchain with fast processing and optimal cross-shard overhead.

4.1 Sharding Blockchain Formulation

4.1.1 Sharding Blockchain

In Logos, there are N nodes divided into m shards. Each shard S_i (where $i = 1, \cdots, m$) consists of n nodes $\{\mathcal{P}_j\}_{\{j=1,\cdots,n\}}$ and manages a subset of accounts. Logos operates in consecutive rounds to process and commit transactions. Transactions TX are divided into two types, including *intra-shard transactions* $\mathsf{TX}_{\mathrm{in}}$ (involving a single shard) and *cross-shard transactions* $\mathsf{TX}_{\mathrm{cr}}$ (involving multiple shards). For $\mathsf{TX}_{\mathrm{in}}$, involved shard S_i runs BFT to process it. For $\mathsf{TX}_{\mathrm{cr}}$, all involved shards $\{\mathsf{S}^{\mathrm{in}}, \mathsf{S}^{\mathrm{out}}\}$ (S^{in} for input shard; $\mathsf{S}^{\mathrm{out}}$ for output shard) process it via the proposed *robust broadcast-transmission-agreement* path. Only valid transactions will be finally committed on each involved shard ledger $\mathsf{S}_i.\log$ where each transaction TX corresponds to a specific position h. Invalid transactions, involving invalid signatures or unavailable inputs, will not be recorded by any shard. The cross-shard message exchange relies on the *parallel single-to-single transmission* mechanism.

4.1.2 Security Goals

We define the secure sharding blockchain protocol in Definition 1 with the following properties.

Definition 1 (Secure sharding blockchain)**.** *In a sharding blockchain protocol, each shard records the committed transactions on its shard ledger. The sharding blockchain is said to be secure if the following properties are satisfied.*

- *Consistency in a single shard. For any two honest nodes $\mathcal{P}_i, \mathcal{P}_j$ in the same shard, if $\mathcal{P}_i$ records a transaction* TX *on the ledger* $\mathsf{S}.\log$ *with height h, then $\mathcal{P}_j$ must record* TX *on the same height on its ledger.*
- *No conflict. For any two honest nodes $\mathcal{P}_i$ in S_1 and $\mathcal{P}_j$ in S_2, if $\mathcal{P}_i$ outputs $\mathsf{S}_1.\log$ and $\mathcal{P}_j$ outputs $\mathsf{S}_2.\log$, then it must holds that there does not exist $\mathsf{TX}_1 \neq \mathsf{TX}_2$ and TX_1 spends the same input with TX_2.*
- *Atomicity for cross-shard transactions. For any cross-shard transaction, it is either committed as valid by all involved shards, or rejected by each shard without any final state change if it is invalid.*
- *Liveness. If a valid transaction is submitted, it would be processed within κ rounds of communication, after which it is committed and recorded on relative shards' ledger.*

Robustness. Robustness (Definition 2) focuses on the system processing capacity stability. An adversary can consume system resources by launching flooding attacks, i.e., submitting a large number of invalid CSTXs to severely degrade the proportion of valid CSTXs that are processed.

Definition 2 (Robustness)**.** *A sharding blockchain system is said to satisfy robustness if it can preserve a stable capability to process transactions when subjected to flooding attacks of invalid CSTXs. Concretely, the processing overhead for an invalid CSTX is strictly less than that for a valid CSTX.*

4.1.3 Performance Metrics

To measure the practical performance of sharding blockchains, we utilize the following performance metrics.

Intra-shard overhead (*IS*-ω). *IS*-ω focuses on the total intra-shard execution overhead in CSTP, including the communication and computation cost. As most of the sharding blockchain protocols rely on pluggable BFT protocols as intra-shard consensus, we measure the *IS*-ω fairly by abstracting the BFT cost as $\mathcal{B}$. It is worth noting that asynchronous BFT protocols [19, 23, 39] typically consist of broadcast (e.g., RBC) with cost $\mathcal{C}$, and agreement (e.g., MVBA) with cost $\mathcal{M}$. The specific measurement method of *IS*-ω is given in Definition 3.

Definition 3 (Intra-shard overhead)**.** *Intra-shard overhead IS-ω refers to the overhead within all k involved shards during a single CSTP.*

Cross-shard communication overhead (*CS*-ω). *CS*-ω depends on the CSTP pattern and the size of transmitted cross-shard messages. As nodes in different shards are typically deployed in geographically dispersed locations, cross-shard message delivery happens frequently on the long-distance communication links, which incurs higher transmission costs than intra-shard ones. The specific measurement method of *CS*-ω is given in Definition 4.

Definition 4 (Cross-shard communication overhead)**.** *Cross-shard communication overhead CS-ω represents the total bits of messages transmitted in one cross-shard transmission.*

4.2 System Overview

System initialization. Each shard initializes threshold signature schemes among the shard participants with the threshold $T = n - f$ for proof generation. Each node $\mathcal{P}_i$ can get its individual secret key sk_i and corresponding public key pk_i.

Overview of Logos**.** Logos first leverages the request delivery method proposed by Kronos. Clients first submit their CSTXs to the corresponding output shards. Output shards deliver CSTXs to related input shards. Logos operates in three phases to process cross-shard transactions, including *input shard broadcast*, *cross-shard transmission*, and *related shard agreement*, as shown in Algorithm 1.

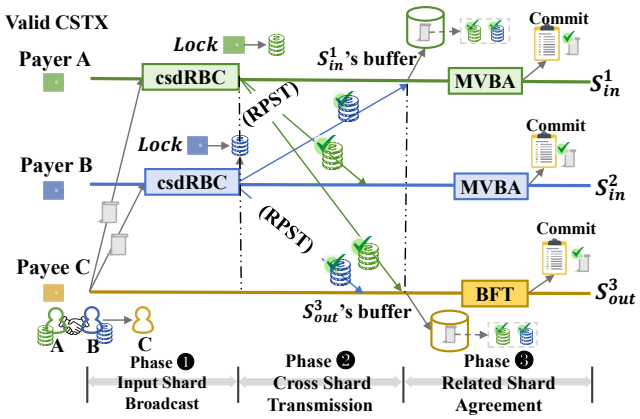


Figure 3: Valid CSTP in Logos.

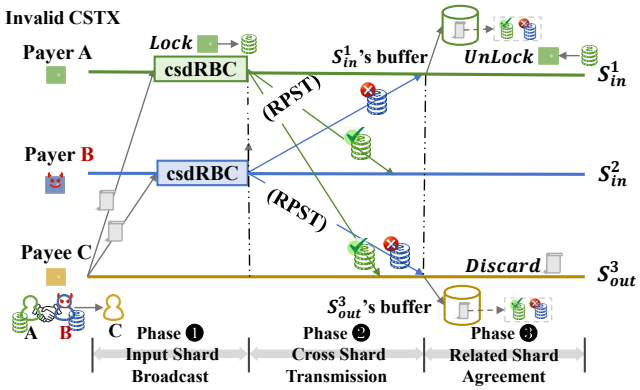


Figure 4: Invalid CSTP in Logos.

Valid CSTP is shown in Figure 3. Each input shard S_i^{in} first invokes the *cross-state determine reliable broadcast* (Algorithm 2, csdRBC) to lock the corresponding balance for valid transactions and generate input availability states with the certificate (Phase ❶ : Input Shard Broadcast). Then each input shard calls the *reliable parallel single-to-single transmission* (Algorithm 3, RPST) to send the input availability states with the certificate to all involved shards (Phase ❷ : Cross-Shard Transmission). Each involved shard uses a buffer to store state availability proofs for pending cross-shard transactions. The buffer state consistency is ensured by csdRBC and RPST since the proofs are generated and transmitted reliably. Finally, the input shards run MVBA and the output shards run BFT to commit $\mathsf{TX}_{\mathrm{cr}}$ on the ledgers (Phase ❸ : Related Shard Agreement).

Invalid CSTP is shown in Figure 4. Each input shard S_i^{in} first invokes csdRBC to generate input availability states (invalid CSTX for unavailability) with the certificate (Phase ❶ : Input Shard Broadcast). Then the input shard calls the RPST to send the unavailability states with the certificate to all involved shards (Phase ❷ : Cross-Shard Transmission). Finally, after receiving the unavailability states with the certificate, all

involved shards will discard the invalid transactions and input shards that have locked the balance will unlock the balance (Phase ❸ : Related Shard Agreement). Once transaction validity is confirmed, corresponding proofs will be removed from the buffer. Logos can be instantiated with different MVBA and asynchronous BFT protocols for flexibility.

Algorithm 1 Cross-Shard Transaction Processing

Phase ❶ : Input Shard Broadcast
▶ As an input shard S_i^{in} of TX_{cr}
1: **call** csdRBC($\{\mathsf{TX}_{\text{cr}}\}$) $\rightarrow \{\langle \mathsf{M}, \mathsf{tree}, \langle \mathsf{rt} \rangle_{\mathsf{S}_i^{\text{in}}} \rangle\}$ ▷ Algorithm 2
2: **for each** M **do**
3: **if** $flag$ is True **then** ▷ valid CSTX
4: **lock** the account balance needed in TX_{cr}
5: **put** TX_{cr} into TXQ_{wait}
6: **else** ▷ invalid CSTX
7: **discard** TX_{cr}

Phase ❷ : Cross-Shard Transmission
▶ As an input shard S_i^{in}
8: **call** RPST(M,tree,$\langle \mathsf{rt} \rangle_{\mathsf{S}_i^{\text{in}}}$) to send it to involved shards ▷ Algorithm 3
▶ As an involved shard S of TX_{cr}
9: **upon** receiving M from RPST of S_i^{in} **do**
10: **parse** $\mathsf{M} = \langle h_i, \mathsf{TX}_{\text{cr}}, flag \rangle$ ▷ input availability states

Phase ❸ : Related Shard Agreement
▶ As an input shard S_i^{in} of TX_{cr}
11: **for each** M **do**
12: **if** $flag$ is True **then**
13: **put** TX_{cr} into TXQ_{wait}
14: **else** ▷ invalid CSTX
15: **discard** TX_{cr}
16: **unlock** the account balance needed in TX_{cr}
17: **call** MVBA with the predicate $Q(\cdot)$ to commit TX_{cr} on S_i^{in}.log
where $Q(\cdot)$ is set to wait for all input flags of TX_{cr} are True to return 1
▶ As an output shard $\mathsf{S}_j^{\text{out}}$ of TX_{cr}
18: **for each** M **do**
19: **if** $flag$ is True **then**
20: **put** TX_{cr} into TXQ_{wait}
21: **else** ▷ invalid CSTX
22: **discard** TX_{cr}
23: **for each** TX_{cr} that all input flags of TX_{cr} are True **do** ▷ valid CSTX
24: **call** BFT to commit TX_{cr} on $\mathsf{S}_j^{\text{out}}$.log

4.3 Cross-Shard Transaction Processing

Next, we illustrate the details of the proposed robust CSTP pattern in Algorithm 1 stated in the shard's perspective (actually executed by each node in the shard). The valid and invalid CSTXs are processed with the following paths, respectively.

Valid CSTP. We first introduce the valid CSTX processing.

Phase ❶ : Input Shard Broadcast. For each input shard S_i^{in} of TX_{cr}, all nodes can perform as the sender and call csdRBC shown in Algorithm 2 to generate input availability states $\mathsf{M} = \langle h_i, \mathsf{TX}_{\text{cr}}, flag \rangle$ with the certificate $\langle \mathsf{rt} \rangle_{\mathsf{S}_i^{\text{in}}}$ (Line 1). $\langle \mathsf{rt} \rangle_{\mathsf{S}_i^{\text{in}}}$ is generated in csdRBC (Algorithm 2, Line 17) by aggregating $2f+1$ shares $\langle \mathsf{rt} \rangle_{sig_j}$ into one threshold signature and can be verified using the total public key of shard S_i^{in}. Afterwards, for TX_{cr} with available input ($flag$ is True), S_i^{in} locks the account balance needed in TX_{cr} and puts TX_{cr} into a valid transaction waiting queue TXQ_{wait}. TXQ_{wait} is used to collect all input availability states of TX_{cr} also from other involved shards (Lines 2-5).

Phase ❷ : Cross-Shard Transmission. Each S_i^{in} performs as a sender shard and invokes RPST (Algorithm 3) to transmit the input availability states M and its certificate to all other involved shards (Line 8). All other involved shards can finally get M (Lines 9-10).

Phase ❸ : Related Shard Agreement. For each involved input shard, after receiving M, TX_{cr} with available input is recorded into TXQ_{wait} (Lines 11-13). S_i^{in} calls MVBA and commit valid TX_{cr} on the shard ledger S_i^{in}.log. MVBA has a predicate $Q(\cdot)$ to verify if the transaction inputs are all available, which means TX_{cr} is also output as available by the local csdRBC (Line 17). Meanwhile, for each output shard, after reconstructing M, TX_{cr} receiving all available inputs is also recorded into TXQ_{wait} (Lines 18-20). Then, $\mathsf{S}_j^{\text{out}}$ calls BFT to commit valid TX_{cr} on the ledger $\mathsf{S}_j^{\text{out}}$.log and change the payee account state to finalize the TX_{cr} (Lines 23-24).

Invalid CSTP. Then we give details of invalid CSTX processing, where the differences mainly lie in the agreement phase.

Phase ❶ : Input Shard Broadcast. For each input shard S_i^{in} of TX_{cr}, it first calls csdRBC shown in Algorithm 2 to generate input availability states $\mathsf{M} = \langle h_i, \mathsf{TX}_{\text{cr}}, flag \rangle$ ($flag$ is False) with the certificate $\langle \mathsf{rt} \rangle_{\mathsf{S}_i^{\text{in}}}$ (Line 1). After csdRBC, all honest nodes in S_i^{in} can ensure that TX_{cr} has the unavailable input. Then S_i^{in} discards the invalid transactions (Lines 6-7).

Phase ❷ : Cross-Shard Transmission. Each S_i^{in} performs as a sender shard and calls RPST (Algorithm 3) to transmit the input availability states M with its certificate to all other involved shards (Line 8). According to Algorithm 3, all other shards can finally receive M (Lines 9-10).

Phase ❸ : Related Shard Agreement. After receiving unavailable input of TX_{cr}, each input shard S_i^{in} unlocks the account balance for locked available input of TX_{cr} (Lines 14-16). All output shards discard the TX_{cr} (Lines 21-22).

4.4 Cross-State Determine Reliable Broadcast

4.4.1 Overview and Definition

One key idea of constructing Logos is to determine the transaction validity and generate input availability states with the certificate in each shard via csdRBC. Therefore, we design the csdRBC formalized as follow.

In csdRBC, each node $\mathcal{P}_j$ can perform as the proposer on the height h and there is a global predicate Judge$(\cdot)$ to validate the proposer's input. $\mathcal{P}_j$ will broadcast an input availability state M including the height h, the transaction TX_{cr}, and the validation tag $flag = \text{Judge}(\cdot)$ for TX_{cr} to all nodes. If there are two conflicting M and M′ (with the same height h and conflicting TX_{cr}), each node would not output or output only one tuple $(\mathsf{M}, \mathsf{cert})$. The certificate cert should be the valid proof

to verify M including the threshold signature. Besides the above syntax, csdRBC should be secure as per Definition 5.

Definition 5 (Security of csdRBC)**.** *The cross-state determine reliable broadcast should satisfy the following properties.*

- *One-State Agreement. For any two honest nodes, if one outputs* M *and another outputs* M' *from proposer* $\mathcal{P}_j$ *on height h, then* $\mathsf{M} = \mathsf{M}'$ *holds.*
- *Totality. If one honest node outputs the input availability states* M *from proposer* $\mathcal{P}_j$*, then every honest node outputs* M *finally.*
- *Cross-State Non-Conflicting Validity. If two honest proposer input conflicting* M *and* M'*, then at most one of the* M *and* M' *can be outputted.*

4.4.2 csdRBC Construction

Algorithm 2 Cross-State Determine Reliable Broadcast (csdRBC) with proposers $\{\mathcal{P}_j\}_{j=1}^{n}$ and predicate Judge

Step ①: Preparation and Transaction Proposal
▶ As a node $\mathcal{P}_j$ ▷ $\mathcal{P}_j$ perform as a proposer
1: **input** $\mathsf{M} = \langle h_j, \mathsf{TX}_{\text{cr}}, flag \leftarrow \mathsf{Judge}(\mathsf{TX}_{\text{cr}}, \mathcal{P}_j.\log)\rangle$
▷ M is an input availability state for TX_{cr}
2: **broadcast** $\langle \textsc{Propose}, \mathsf{M}\rangle$

Step ②: State Determination and Non-Conflicting Voting
▶ As a node $\mathcal{P}_i$ ▷ $\mathcal{P}_i$ perform as a receiver
3: **upon** receiving $\langle \textsc{Propose}, \mathsf{M}\rangle$ from the proposer $\mathcal{P}_j$ **do**
4: **wait** until $h_i = \mathsf{M}.h_j$ ▷ TX_{cr} is determined under the newest state
5: **if** $\mathsf{Judge}(\mathsf{M}.\mathsf{TX}_{\text{cr}}, \mathcal{P}_i.\log) = \mathsf{M}.flag$ and not having sent $\langle \textsc{Echo}, \mathsf{M}'\rangle$ with conflicting TX'_{cr} at height h_i **then**
6: **broadcast** $\langle \textsc{Echo}, \mathsf{M}\rangle$
7: **else**
8: **discard** $\langle \textsc{Propose}, \mathsf{M}\rangle$

Step ③: Input Availability Certificate Generation
9: **upon** receiving $2f+1$ valid $\langle \textsc{Echo}, \mathsf{M}\rangle$ **do**
10: **call** $\mathsf{ECEnc}(\phi, \mathsf{M}, N, f+1) \rightarrow \{[\mathsf{M}]_i\}_n$
11: **call** $\mathsf{TreeCon}(\{\mathsf{Hash}([\mathsf{M}]_i)\}) \rightarrow (\mathsf{tree}, \mathsf{rt})$
12: **call** $\mathsf{ShareSig}(sk_i, \mathsf{rt}) \rightarrow sig_i$
13: **broadcast** $\langle \textsc{Ready}, \mathsf{M}, \langle \mathsf{rt}\rangle_{sig_i}\rangle$
14: **upon** receiving $f+1$ valid $\langle \textsc{Ready}, \mathsf{M}, \langle \mathsf{rt}\rangle_{sig_j}\rangle$ with the same M and not having sent $\langle \textsc{Ready}, \mathsf{M}, \langle \mathsf{rt}\rangle_{sig_i}\rangle$ **do**
15: **broadcast** $\langle \textsc{Ready}, \mathsf{M}, \langle \mathsf{rt}\rangle_{sig_i}\rangle$
16: **upon** receiving $2f+1$ valid $\langle \textsc{Ready}, \mathsf{M}, \langle \mathsf{rt}\rangle_{sig_j}\rangle$ with the same M **do**
17: **call** $\mathsf{Combine}(\{sig_j\}) \rightarrow \langle \mathsf{rt}\rangle_{\mathsf{S}_s}$ ▷ input availability states certificate
18: **output** $\langle \mathsf{M}, \mathsf{cert} \leftarrow (\mathsf{tree}, \langle \mathsf{rt}\rangle_{\mathsf{S}})\rangle$

19: **function** $\mathsf{Judge}(\mathsf{TX}, \mathcal{P}_i.\log) \rightarrow flag$ ▷ judge the input availability
20: **if** TX has valid input based on $\mathcal{P}_i.\log$ **then set** $flag = \mathsf{True}$
▷ All involved accounts have enough balance for TX
21: **else set** $flag = \mathsf{False}$

Next, we give the detailed construction of csdRBC in Algorithm 2. Input shards of TX_{cr} invoke csdRBC, including the following steps.

Step ①: Preparation and Transaction Proposal. csdRBC uses the function $\mathsf{Judge}(\mathsf{TX}_{\text{cr}}, \mathcal{P}_j.\log)$ as the predicate. Each node $\mathcal{P}_j$ uses this function to judge the transaction TX_{cr} input availability based on its local ledger. If all involved accounts have enough balance to pay for TX_{cr}, $flag$ is set to True. Else, $flag$ is set to False (Lines 19-21). Then $\mathcal{P}_j$ proposes the input availability state M including its local block height h_j, the transaction TX_{cr}, and the validation tag $flag$. Then $\mathcal{P}_j$ broadcasts the PROPOSE message to the shard S (Lines 1-2).

Step ②: State Determination and Non-Conflicting Voting. After receiving the PROPOSE message from a proposer $\mathcal{P}_j$, each node $\mathcal{P}_i$ votes for the proposal via broadcasting a ECHO message if the following conditions are fulfilled. Firstly, by checking the height h, $\mathcal{P}_i$ verifies whether the proposal from $\mathcal{P}_j$ is constructed based on the same state as $\mathcal{P}_i$'s. If the proposal is based on a newer state, $\mathcal{P}_i$ will wait until the state is updated (Lines 3-4). While the states are aligned, $\mathcal{P}_i$ determines the input availability of TX_{cr} by the $\mathsf{Judge}(\cdot)$ function, checks the correctness of $flag$ in the proposal, and votes for TX_{cr} (Lines 5-6). Moreover, if $\mathcal{P}_i$ has voted for a transaction TX'_{cr} which conflicts with TX_{cr} (TX_{cr} and TX'_{cr} spend the same input), $\mathcal{P}_i$ will not vote for TX_{cr} (Lines 7-8).

Step ③: Input Availability Certificate Generation. After receiving $2f+1$ valid ECHO messages on the same TX_{cr}, each node $\mathcal{P}_i$ makes sure that at least $f+1$ honest nodes has voted for TX_{cr}. Then each $\mathcal{P}_i$ uses erasure coding and Merkle tree to generate the threshold signature share $\langle \mathsf{rt}\rangle_{sig_i}$ for the proposed input availability states M (Lines 9-13). Each $\mathcal{P}_i$ broadcasts the READY message with its proof. After the RBC-style broadcast, all honest nodes can finally receive $2f+1$ signature shares and aggregate them into $\langle \mathsf{rt}\rangle_{\mathsf{S}}$ as the input availability certificate (Lines 14-18).

Regarding consistent state determination. Note that a malicious client can submit conflicting transactions with the intention of performing a double-spending attack. If multiple conflicting CSTXs are proposed at the same height h, honest nodes may vote for different CSTXs due to the unawareness of conflicting CSTXs caused by the asynchronous network delay. As honest node can only vote for one of conflicting CSTXs, it is possible that neither of the conflicting CSTXs gets $2f+1$ votes. However, all conflicting CSTXs can be finally received by at least one honest node via csdRBC. Therefore, this honest node will broadcast the conflicting transactions and all honest nodes can finally receive and solve the conflict via specific voting rules such as the hash order in one of the later rounds. For example, for the received two conflicting transactions TX_{cr} and TX'_{cr}, if $\mathsf{Hash}(\mathsf{TX}_{\text{cr}}) < \mathsf{Hash}(\mathsf{TX}'_{\text{cr}})$, then each honest node considers that TX_{cr} has an available input. For further realizing fair processing for multiple conflicting transactions, some randomization method such as random beacon or pseudorandom function can be introduced to prevent certain transactions consistently getting priority. For example, each honest node determines the validity by computing $\mathsf{Hash}(\mathsf{TX}_{\text{cr}}, r)$ while r is a common random number generated unbiasedly and unpredictably.

4.5 Parallel Single-to-Single Transmission

4.5.1 Overview and Definition

Another critical component of Logos is that each input shard (abstracted as sender shard) invokes RPST to send the same input availability states with the certificate from the output of csdRBC to all involved shards (abstracted as receiver shard).

In Logos, each node $\mathcal{P}_i$ in the sender shard takes input availability state with the certificate $\langle \mathsf{M}, \mathsf{tree}, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ outputted by csdRBC as the input. Each node $\mathcal{P}'_i$ in the receiver shard would output M. Besides the above syntax, RPST should satisfy reliability as per Definition 6. Notably, RPST can transmit any identical messages with certificates between two groups, not limited to input availability states in Logos.

Definition 6 (Cross-shard reliability)**.** *All honest nodes in the receiver shard will output the same* M *if each honest node* $\mathcal{P}_i$ *in the sender shard inputs the same* $\langle \mathsf{M}, \mathsf{tree}, \langle \mathsf{rt} \rangle_{\mathsf{S}_s} \rangle$.

4.5.2 RPST Construction

Next, we introduce the details of the *reliable parallel single-to-single transmission* RPST in Algorithm 3 and Figure 5.

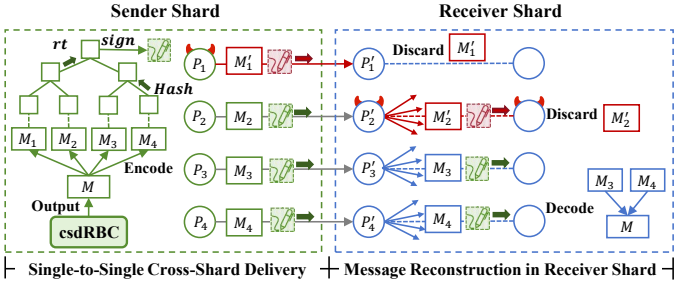


Figure 5: Reliable parallel single-to-single transmission.

Step ④: Single-to-Single Cross-Shard Delivery. The sender shard takes an input availability state M with the certificate $\langle \mathsf{rt} \rangle_{\mathsf{S_s}}$ outputted by csdRBC as the input of RPST (Line 1). Each sender node $\mathcal{P}_i$ gets the fragment $[\mathsf{M}]_i$ through erasure coding $\mathsf{ECEnc}(\phi, \mathsf{M}, N, f+1)$ and generates a Merkle tree for fragments (Line 2-3). During the cross-shard transmission, the sender node $\mathcal{P}_i$ in $\mathsf{S_s}$ only needs to deliver the fragment $[\mathsf{M}]_i$, corresponding Merkle proof mproof_i, and the certificate $\langle \mathsf{rt} \rangle_{\mathsf{S_s}}$ to nodes $\mathcal{P}'_i$ in the receiver shard. The certificate means that at least $2f+1$ senders (at least $f+1$ honest nodes) have authenticated M and each fragment $\{[\mathsf{M}]_i\}_n$ (Line 4).

Step ⑤: Message Reconstruction in Receiver Shard. After receiving the fragment $[\mathsf{M}]_i$ from the sender node $\mathcal{P}_i$, $\mathcal{P}'_i$ verifies the validity of the certificate and the Merkle proof. Then $\mathcal{P}'_i$ broadcasts verified fragment $[\mathsf{M}]_i$ to nodes in $\mathsf{S_r}$ (Lines 5-7). Here, $\mathcal{P}'_i$ may receive nothing or wrong messages from a malicious sender $\mathcal{P}_i$. As a result, an honest receiver $\mathcal{P}'_i$ won't broadcast $[\mathsf{M}]'_i$ received from $\mathcal{P}_i$ (Lines 8-9). After receiving $f+1$ valid fragments, every receiver can reconstruct the input availability states M by the decoding operation (Lines 10-11). The reconstruction in the receiver shard can be embedded into the asynchronous BFT without introducing additional communication rounds, since most asynchronous BFT inherently relies on broadcast to ensure message delivery.

Realizing asynchronous reliability via RPST. As shown in Figure 5, after executing csdRBC, all honest nodes can output the same M with the certificate $\langle \mathsf{rt} \rangle_{\mathsf{S_s}}$ and run RPST. In the worst case, f malicious senders and f malicious receivers try to deviate from the protocol by sending the wrong fragments or not sending anything. However, all wrong fragments can be verified and discarded and at least $f+1$ correct fragments $[\mathsf{M}]_i$ delivered on $f+1$ honest (sender, receiver) pairs can be finally received in asynchronous network via broadcast in receiver shard. Therefore, in RPST, the original message can be decoded correctly by all honest receivers without the need of timeout and recovery mechanisms.

Algorithm 3 Reliable Parallel Single-to-Single Transmission (RPST) with sender shard $\mathsf{S_s}$ and receiver shard $\mathsf{S_r}$

Step ④: Single-to-Single Cross-Shard Message Delivery
▶ As a node $\mathcal{P}_i$ in $\mathsf{S_s}$
1: **input** $\langle \mathsf{M}, \mathsf{tree}, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ ▷ the output of csdRBC in $\mathsf{S_s}$
2: **get** $[\mathsf{M}]_i$ from $\mathsf{ECEnc}(\phi, \mathsf{M}, N, f+1)$ ▷ Line 10, Algorithm 2
3: **call** $\mathsf{MpfGen}(\mathsf{tree}, [\mathsf{M}]_i) \rightarrow \mathsf{mproof}_i$
4: **send** $\langle [\mathsf{M}]_i, \mathsf{mproof}_i, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ to $\mathcal{P}'_i$ in $\mathsf{S_r}$

Step ⑤: Message Reconstruction in Receiver Shard
▶ As a node $\mathcal{P}'_i$ in $\mathsf{S_r}$
5: **upon** receiving $\langle [\mathsf{M}]_i, \mathsf{mproof}_i, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ from a sender $\mathcal{P}_i$ in $\mathsf{S_s}$ **do**
6: **if** $\langle \mathsf{rt} \rangle_{\mathsf{S_s}}$ is valid and mproof_i is valid from $\mathsf{Hash}([\mathsf{M}]_i)$ to rt **then**
7: **send** $\langle [\mathsf{M}]_i, \mathsf{mproof}_i, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ to all $\mathcal{P}'_j$ in $\mathsf{S_r}$
8: **else**
9: **discard** $[\mathsf{M}]_i$
10: **upon** receiving $f+1$ valid $\langle [\mathsf{M}]_j, \mathsf{mproof}_j, \langle \mathsf{rt} \rangle_{\mathsf{S_s}} \rangle$ with the same rt **do**
11: **output** $\mathsf{M} \leftarrow \mathsf{ECDec}(\phi, \{[\mathsf{M}]_j\}_{f+1})$

5 System Analysis

5.1 Security Analysis

According to Algorithm 1, the security of Logos relies on the composition of of csdRBC, RPST, MVBA and BFT. Due to the page limitations, we provide lemmas that Logos satisfy the properties and its associated guarantees. Please see Appendix D for detailed proofs.

Lemma 1 (Consistency in a single shard)**.** *Let* $n \geq 3f+1$ *hold in each* S_i*, then* Logos *satisfies consistency in a single shard as per Definition 1.*

Proof of Lemma 1. The consistency property is ensured by the *safety* of BFT for intra-shard transactions and the execution of cross-shard transactions in output shards. The consistency property also relies on the *agreement* of MVBA for the execution of cross-shard transactions in input shards. □

Lemma 2 (No conflict)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies no conflict as per Definition 1.*

Proof of Lemma 2. The no conflict property is ensured by the *cross-state non-conflicting validity* property and the conflicting solving rules of csdRBC in each shard. The no conflict property also relies on the *cross-shard reliability* of RPST for solving conflicts in multiple shards. □

Lemma 3 (Atomicity for cross-shard transactions)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies atomicity for cross-shard transactions as per Definition 1.*

Proof of Lemma 3. The atomicity property is ensured by the *one-state agreement* and *totality* of csdRBC, *cross-shard reliability* of RPST for all CSTXs. For valid CSTXs, the atomicity further relies on the security of BFT in output shards and the security of MVBA in input shards. □

Lemma 4 (Liveness)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies liveness as per Definition 1 with κ rounds.*

Proof of Lemma 4. The liveness is ensured by the csdRBC, the *cross-shard reliability* of RPST, the *liveness* of BFT, and the *termination* of MVBA over the no conflict property. □

Theorem 1 (Logos is a secure sharding blockchain)**.** *Let $n \geq 3f+1$ hold in each S_i, then* Logos *is a secure sharding blockchain as per Definition 1.*

Proof of Theorem 1. By combining Lemma 1, 2, 3 and 4, Theorem 1 holds for Logos. □

5.2 Complexity Analysis

In this section, we analyze the intra-shard and cross-shard overhead for transaction processing in Logos. Additionally, we delve into the lower bound of intra-shard and cross-shard overhead while ensuring secure transaction processing and give a more detailed comparison with Kronos in Table 2.

Table 2: Comparison between Logos and Kronos.

Scheme		Kronos [34]	Logos
CSTX Processing Method		Execution-Rollback	Broadcast-Transmission-Agreement
IS Overhead	Valid CSTX	$k\mathcal{B}$	$k\mathcal{B}$
	Invalid CSTX	$2x\mathcal{B}$	$x\mathcal{C}$
Liveness Parameter*	Valid CSTX	$2\kappa_{\mathsf{BFT}}+2$	$\kappa_{\mathsf{BFT}}+5$
	Invalid CSTX	$2\kappa_{\mathsf{BFT}}+1$	5
CS Transmission Pattern		$O(n)$-to-$O(n)$ Broadcast	Single-to-Single Transmission
CS Overhead	Message Bits$^\aleph$	$O(nb\lambda+n^2\lambda)$	$O(b\lambda+n\lambda)^\aleph$
	Long links	$O(n^2)$	$O(n)$

* Liveness Parameter: the communication rounds of CSTP. κ_{BFT}: specific round cost in a BFT.

$^\aleph$ For $b > n$, the CS transmission bits of Logos are $O(b\lambda)$.

5.2.1 Intra-Shard Overhead

Most of sharding blockchain protocols rely on BFT to process transactions in each shard and the BFT execution is the main intra-shard overhead. We use $\mathcal{B}$ to denote a BFT cost. Moreover, in our Logos, we decouple the BFT execution into broadcast and agreement in the input shard. In addition to $\mathcal{B}$, we also use $\mathcal{C}$ and $\mathcal{M}$ to denote the broadcast and agreement cost. Since most existing asynchronous BFT [19, 21, 23, 40] adopts the broadcast-agreement design, we assume $\mathcal{B} = \mathcal{C} + \mathcal{M}$. According to the following analysis, Logos has optimal intra-shard overhead for valid transactions and optimized intra-shard overhead for invalid transactions.

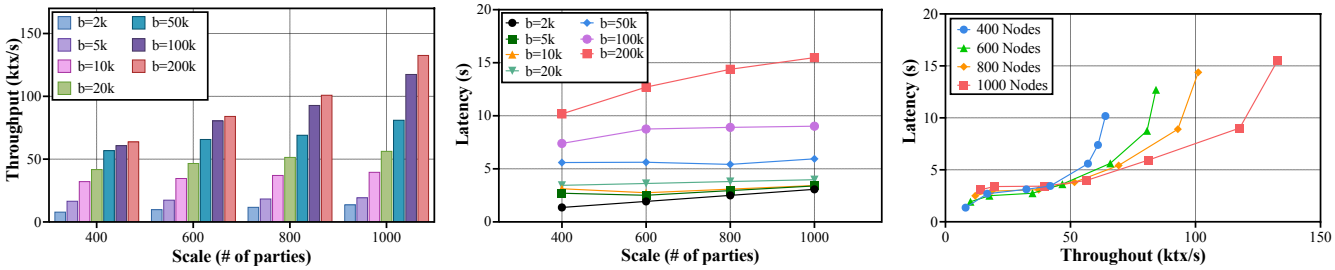
Theorem 2 (Optimal intra-shard overhead)**.** Logos *commits a k-shard-involved transaction* TX *through executing* BFT *k times totally, realizing the lower bound of intra-shard overhead IS-$\omega = k\mathcal{B}$, which is optimal for a secure sharding blockchain as per Definition 3.*

Proof of Theorem 2. The transaction processing needs state update on UTXO/accounts with finalization on the shard ledger log in each involved shard. In blockchain, secure updating is ensured only through consensus, requiring at least one time of BFT in each shard. If a transaction TX is committed in fewer BFT rounds than the total shard number k, there must be at least one involved shard S that fails to achieve consensus on TX, leading to the loss of atomicity. Therefore, $k\mathcal{B}$ is the lower bound of *IS*-ω for valid transaction processing. In Logos, BFT protocols can be decoupled into a broadcast phase and an agreement phase. For the valid CSTP, each input shard needs one round of broadcast and one round of agreement, leading to a total IS-overhead of $\mathcal{B}$. Overall, Logos has an optimal *IS*-$\omega = k\mathcal{B}$ for valid CSTP. □

Comparison between Kronos **and** Logos **for intra-shard overhead of Invalid CSTP.** Compared to Kronos, Logos has a much lower *IS*-ω in invalid CSTP. In Kronos, when the unhappy path in invalid CSTP happens in an input shard S_i^{in}, an extra entire BFT is necessary to rollback the TX_{cr}. It leads to a high *IS*-$\omega = 2x\mathcal{B}$. (x denotes the executed valid inputs in invalid CSTP). More specifically, the BFT execution can be decoupled into broadcast and agreement. So *IS*-ω for invalid transaction processing in Kronos equals $2x\mathcal{C} + 2x\mathcal{M}$. However, in Logos, the main intra-shard cost on invalid transaction processing is the execution of csdRBC. Therefore, *IS*-ω for invalid transaction processing is only *IS*-$\omega = x\mathcal{C}$.

5.2.2 Cross-Shard Complexity Analysis

For a cross-shard transmission, we analyze the transmission bits and cross-shard long links. The message transmission bits directly affect the transmission delay, while the maintenance of cross-shard long links consumes the CPU and memory resources, which influence the scalability. Logos has optimal cross-shard overhead while realizing cross-shard reliability.



(a) Peak throughput (b) Latency (c) Latency-throughput trade-off

Figure 6: Performance of Logos in the WAN setting.

Definition 7 (Cross-shard long links)**.** *Cross-shard long links represent the number of total long links between two shards for cross-shard transmission.*

Theorem 3 (Optimal cross-shard communication overhead)**.** Logos *realizes optimal CS-*$\omega = O(b\lambda)$ *as per Definition 4 and least cross-shard long links* $O(n)$ *as per Definition 7 while achieving cross-shard reliability.*

Proof of Theorem 3. Communication overhead can be reduced by minimizing the exchanged message number or decreasing each message size. Logos delves into reducing message size from two aspects. To minimize the exchanged message number, we first analyze the cross-shard reliability conditions. Kronos intuitively uses $O(n)$-to-$O(n)$ cross-shard broadcast to ensure reliability. However, we observe that the cross-shard reliability only needs $f+1$ non-overlapping honest (sender, receiver) pairs if the honest receiver can verify the message from the sender and prevent wrong message transmission. To achieve $f+1$ non-overlapping honest (sender, receiver) pairs, Logos replaces $O(n)$-to-$O(n)$ broadcast to RPST to reduce the cross-shard message exchange number and the cross-shard long links from $O(n^2)$ to $O(n)$.

Furthermore, to decrease each message size, Logos uses erasure coding and each $\mathcal{P}_i$ only sends one $O(b\lambda/n)$-sized message fragment (with b transactions and each transaction with security parameter λ). For the fragment certification and commitment compression, Logos adopts the Merkle tree and n message fragments are constructed to a Merkle tree with n leaf nodes. Each sender sends the message fragment with a hash path with the size of $O(log n\lambda)$. Therefore, the cross-shard overhead $CS\text{-}\omega = n*(O(b\lambda/n)+O(log n\lambda)) = O(b\lambda + nlog n\lambda)$. The vector commitment [28, 50] can be also utilized as in Kronos to decrease the overhead to $CS\text{-}\omega = O(b\lambda + n\lambda)$. Since b is generally larger than n, $CS-\omega$ can be reduced to $O(b\lambda)$, linear to the batch size. $CS\text{-}\omega = O(b\lambda)$ means the total message bits transmitted across shards have the same order as the targeted message on the information quantity $b\lambda$, indicating the optimality according to information theory [6]. □

6 Performance Evaluation

We implement a prototype of Logos and deploy it across four AWS regions (Virginia, Hong Kong, Tokyo, and London), involving up to 1000 nodes, to evaluate its practical performance. The evaluation include the performance of Logos in a realistic wide-area network (Section 6.1) and the comparison to the existing SOTA sharding protocol Kronos (Section 6.2).

Implementation details. We program Logos and Kronos in Golang. Each shard adopts Speeding Dumbo and Speeding MVBA (the well-known efficient asynchronous BFT and MVBA) [23]. More details of Speeding MVBA are shown in Appendix B. We use SHA-256, BLS signature, and Reed-Solomon code to instantiate hash function, threshold signature, and erasure coding, respectively.

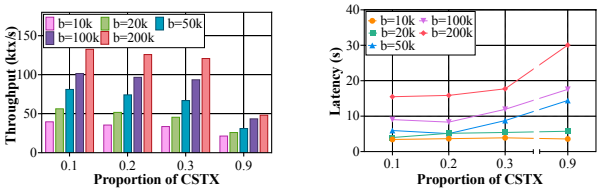
Deployment on Amazon EC2. We deploy Logos among Amazon EC2 c5.xlarge instances, which are equipped with 4vCPUs and 8GB main memory. Each node is allocated 1vCPU at 3GHz and 2GB RAM, closely to real-world settings. The performances are evaluated with varying scales at up to $N = 1000$ nodes. The transaction length is 128 bytes, which approximates the size of most transfer transactions in Ethereum.

6.1 Overall Performance of Logos

Throughput and latency under different networks. We first measure the throughput (i.e., the number of transactions processed per second). We vary the batch sizes (i.e., the number of transactions proposed in each round) from $b = 2$k to 200k for evaluation with 50 nodes per shard. As illustrated in Figure 6a, Logos demonstrates scalability, showcasing an increasing throughput as network scales and achieving a peak throughput of 132.8 ktx/sec with $N = 1000$, $b = 200$k. Figure 6b illustrates Logos's latency across varying network sizes. The latency remains below 3.43 sec for network scales $N \leq 1000$ and batch sizes $b \leq 10$k. It demonstrates the effectiveness of Logos for latency-sensitive applications, even in large-scale networks.

Throughput-latency trade-off. Figure 6c illustrates the throughput-latency trade-off of Logos. The latency stays be-

low 10.18 sec, with the throughput reaching 64.15 ktx/sec in a medium-scale network ($N = 400$). In large-scale networks ($N = 1000$), the latency remains limited as the throughput reaches a sizable 117.70 ktx/sec. This trade-off underscores the applicability of Logos in large-scale distributed networks requiring both throughput and latency.



(a) Throughput (b) Latency

Figure 7: Performance under different CSTX proportions.

Throughput and latency under different CSTX proportion. Figure 7a and Figure 7b illustrate the throughput and latency of Logos under different CSTX proportion. Although throughput decreases and latency increases as the CSTX proportion rises, the throughput can still be maintained at 120.91 ktps, while the latency remains within 17.72 sec. Even under 90% CSTX proportion, the throughput can maintained at 43.5ktps.

6.2 Comparison with Existing Solutions

Comparison with Kronos under different shards. We compare the performance of Logos with the baseline protocol of Kronos [34] (NDSS'25), a secure asynchronous sharding blockchain. For fair comparison, we set that each shard has 50 nodes in both Logos and Kronos and the shards vary from 10 to 20. Figure 8a, Figure 8b, Figure 8c depict the throughput, latency, and throughput-latency trade-off of Logos in comparison to Kronos, respectively. Overall, Logos outperforms Kronos in all cases. Notably, the throughput of Logos exceeds $1.87\times$ that of Kronos and realizes only 50% of the total latency. Additionally, Logos exhibits a significant advantage in peak throughput with low latency making it applicable in both latency-sensitive and throughput-critical scenarios.

Comparison with Kronos under different shard size. We also compare the performance of Logos and Kronos in different shard sizes from 50 to 150 with 6 shards. Figure 8d depicts the throughput-latency trade-off results. Overall, although the throughput decreases and latency increases as the shard size grows, Logos still outperforms Kronos.

Cross-shard communication overhead in real-world networks. We compare the practical CS-overhead of Logos with Kronos by checking the cross-shard network usage under different shard size from 50 to 150. The batch sizes are set as 10k, 50k, and 100k. Figure 8e depicts that the network usages of Logos remain nearly constant while the network usages of Kronos increase vastly with the growth of shard sizes. Under the largest shard size and batch size, the network usage of Logos remains only $2.3 * 10^2$ MB, with an impressive ratio of $1/210$ compared to $4.83 * 10^4$ MB of Kronos for processing 600k cross-shard transactions, illustrating low CS-overhead. Logos reduces cross-shard long-links and network consumption, thereby enhancing the system's practicality.

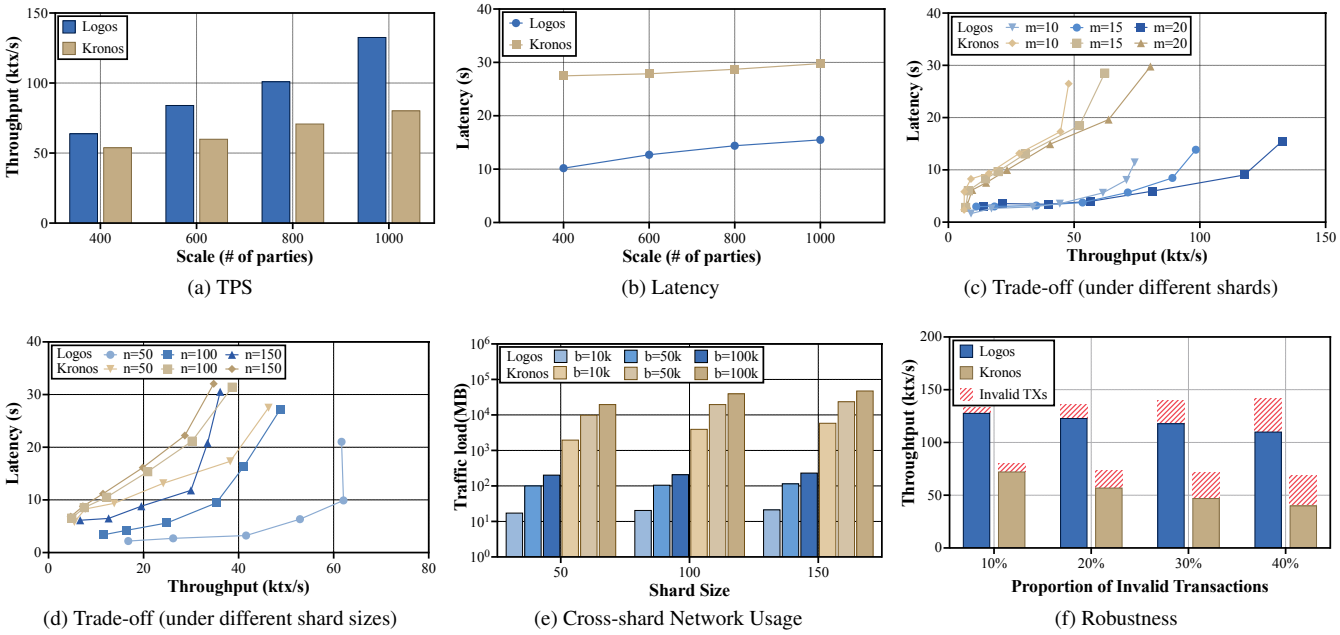
Robustness against malicious flooding. Figure 8f illustrates the throughput of Logos and Kronos under different invalid CSTX proportions. Kronos demonstrates a more obvious decline in throughput for valid transactions and total throughput as the proportion of invalid transactions rises due to the execution of expensive rollback operations within the input shards. In contrast, Logos has a more stable capacity under malicious nodes flooding invalid transactions, achieving $2.86\times$ throughput improvement compared to Kronos when the proportion of invalid CSTXs reaches 40%. Besides, the proportion of valid transactions successfully processed remains 78% while the total throughput slightly rises since processing invalid transactions is faster than processing valid transactions in Logos, showcasing strong *robustness* as per Definition 2.

7 Discussion

7.1 Realizing Honest Majority

In Logos, each shard is required to satisfy $n \geq 3f+1$ ("honest majority" or "secure shard" in some related works). While some prominent existing works such as Monoxide [49] and Manifoldchain [10] adopt the Proof-of-Work (PoW) which only require a global honest majority, it still remains a fundamental challenge for many sharding protocols (adopting BFT) to ensure an honest majority in every shard. Here, we discuss some common methods to realize a secure shard configuration in both permissionless and permissioned environments.

Permissionless setting. The configuration in the permissionless can rely on the reference committee and the a Proof-of-Work (PoW) puzzle [44]. Specifically, each prospective participant solves a PoW challenge, i.e. to find a *nonce* that satisfies $H(nonce, \eta, str) < D$ during the system initialization [29]. Here, η is a public randomness, str is the hash of the previous block, and D is the pre-defined difficulty target. The computed solution is then submitted to a reference committee for identity registration and assignment. $\mathcal{P}_i$ is assigned to c-th of total m shards via a pseudorandom permutation (PRP) function. Specifically, $c \equiv \mathsf{PRP}(\eta, i) \bmod m$. Configuring such secure shards that satisfy $f < n/3$ with a bounded malicious node F (e.g., $N/5$) follows a hypergeometric distribution [17]. The probability for one shard corruption and the overall system failure probability with m shards can be computed as follows. Under our experiment settings of shard size $n = 50$ and shard number $m = 20$, the system failure probability is negligible (less than 10^{-5}).



(a) TPS (b) Latency (c) Trade-off (under different shards)

(d) Trade-off (under different shard sizes) (e) Cross-shard Network Usage (f) Robustness

Figure 8: Comparison of Logos and the baseline Kronos.

$$\Pr[f \geq n/3] = \sum_{f=\lfloor \frac{n+1}{3} \rfloor}^{n} \frac{\binom{F}{f}\binom{N-F}{n-f}}{\binom{N}{n}}$$

$$\Pr[\text{fail}] = 1 - (1 - \Pr[f \geq n/3])^m$$

Permissioned setting. In the permissioned setting, a trusted *certification authority* (CA) can authenticate and assign the participants. The permissioned setting allows for smaller shard sizes while preserving security, which in turn enhances parallelism and improves overall transaction throughput.

Rotating committee to ensure long-term security. During long-term system operation, each shard committee needs to be rotated to prevent shard corruption. The rotation is performed with three steps including reconfiguration, DKG and matching nodes. The reconfiguration process is similar to the initial configuration. After reconfiguration, all new committee members are elected. In permissioned environments, CA can reassign nodes to shards based on predefined policies or performance metrics. In the permissionless setting, nodes jointly run a public randomness generation protocol [14] to generate a new randomness η', then compute PoW solutions based on η' for new allocations. After the reconfiguration, the new committee jointly operates DKG to generate key pairs (pk, sk) of threshold signature for each member, spending only some seconds [51,52]. Next, the public keys of committee members are broadcast across all shards. Then members can be ordered by $\mathsf{Hash}(pk)$ and matched with corresponding nodes in other shards to realize shard coordination. Meanwhile, each node requests the historical states from previous members of its newly assigned shard to resume transaction processing [36].

7.2 Partially Synchronous Implementation

Here we describe the Logos construction under the partially synchronous network in Algorithm 4. The broadcast and transmission phases are the same as the construction in Algorithm 1 (Lines 1-2). For involved shards, invalid CSTP is also processed in the same way as Algorithm 1 (Line 7 and Line 10). For valid CSTXs, the agreement phase in the input shards could be simplified. After each honest node in input shard S_i^{in} receives all available inputs, each honest node broadcasts the FINALIZATION message for TX_{cr} (Lines 3-4). After receiving $2f+1$ FINALIZATION messages, each honest node can make sure that at least $f+1$ honest nodes have received all available inputs. Therefore, each honest node can commit TX_{cr} on the ledger (Lines 5-6). For each output shard, it invokes BFT in partial synchronous networks [9] to commit TX_{cr} (Lines 8-9).

Algorithm 4 Logos In Partially Synchronous Network

▶ As an input shard S_i^{in} of TX_{cr}
1: **invoke** Algorithm 1 Lines 1-7 to finish Phase ❶
2: **invoke** Algorithm 1 Lines 8-10 to finish Phase ❷

Phase ❸ Related Shard Agreement
▶ As an input shard S_i^{in} of TX_{cr}
3: **for each** TX_{cr} that all input flags of TX_{cr} are true **do** ▷ valid CSTX
4: **broadcast** ⟨FINALIZATION, TX_{cr}⟩
5: **upon** receiving $2f+1$ valid ⟨FINALIZATION, TX_{cr}⟩ **do**
6: **commit** TX_{cr} on S_i^{in}.log finally
7: **invoke** Algorithm 1 Lines 14-16 to process invalid TX_{cr}
▶ As an output shard $\mathsf{S}_j^{\text{out}}$ of $\{\mathsf{TX}_{\text{cr}}\}$
8: **for each** TX_{cr} that all input flags of TX_{cr} are true **do** ▷ valid CSTX
9: **call** BFT in partial synchronous network [9] to commit TX_{cr}
10: **invoke** Algorithm 1 Lines 21-22 to process invalid TX_{cr}

7.3 Application on UTXO model

The CSTP pattern of Logos is also applicable to the UTXO model. In UTXO model systems, each shard maintains a UTXO list. A CSTX can be formalized as $\mathsf{TX}_{\text{cr}} = (\mathbf{I}, \mathbf{O}, id)$. $\mathbf{I} = \{I_1, I_2, \cdots\}$ indicates the transaction input set, where each I_i in $\mathbf{I}$ consists of the belonging shard S_i^{in} and $utxo_i$. $\mathbf{O} = \{O_1, O_2, \cdots\}$ indicates the transaction output set, where each O_j in $\mathbf{O}$ includes output shard $\mathsf{S}_j^{\text{out}}$ and the output value v_j. In CSTP, each input shard invokes csdRBC to determine whether $utxo_i$ is in the input shard's UTXO list and invokes RPST to transmit the states to all involved shards. For valid CSTP, after receiving all input availability states, each input shard finally commits TX_{cr} on its ledger and deletes $utxo_i$ from UTXO lists. Each output shard generates a new $utxo_j$ from TX_{cr} and adds $utxo_j$ into its UTXO lists. For invalid CSTP, each input shard unlocks the $utxo_i$ and each output shard discards TX_{cr}.

7.4 Regarding Asymmetric Committee Size

The RPST is also applicable to the asymmetric committee with different size n_s (sender shard with f_s malicious nodes) and n_r (receiver shard with f_r malicious nodes) by fine-tuning the encoding parameters and the communication pattern. Each sender node can encode the message M into $n_{all} = \mathsf{LCM}(n_s, n_r)$ fragments $[\mathsf{M}]_{k=1,\cdots,n_{all}}$ where LCM denotes the least common multiple. Followed the analysis of RPST, there are at most $n_{all}/n_s * f_1$ wrong fragments from malicious senders and $n_{all}/n_r * f_2$ wrong fragments from malicious receivers. Therefore, the parity fragments are $\frac{n_{all}}{n_s} * f_1 + \frac{n_{all}}{n_r} * f_2$. The k-th fragment $[\mathsf{M}]_k$ is transmitted from sender $\mathcal{P}_{\lfloor \frac{k}{n_r} \rfloor}$ to receiver $\mathcal{P}'_{\lfloor \frac{k}{n_s} \rfloor}$ with the certificate.

8 Conclusion

We present Logos, a robust sharding blockchain that achieves fast transaction processing and optimal cross-shard overhead. Logos ensures strong atomicity and robustness by preventing malicious nodes from causing state inconsistencies or undermining the system stability. The proposed reliable parallel single-to-single transmission mechanism enables arbitrary message delivery while reducing cross-shard complexity. Logos can serve as an asynchronous sharding blockchain framework, supporting integration with newly designed RBC and MVBA protocols to enhance both system security and performance. This makes it suitable for large-scale, real-world applications such as finance, digital currency, and Web 3.0 identity management.

Ethical Considerations

We develop Logos, a robust sharding blockchain with fast processing and optimal cross-shard overhead, with the aim of improving security while reducing latency and bandwidth. We carefully consider the ethical implications, striving to promote security, performance, and social benefit while minimizing potential negative outcomes.

Stakeholders. Immediate technical beneficiaries include the engineers, operators, and end-users who rely on blockchain and other distributed environments. The research community gains reproducible benchmarks and an open-source reference implementation. Commercial competitors and open-source maintainers may experience positive or negative market effects. Society at large could be affected in a positive way as the more efficient solutions for legitimate users.

Ethical principles. We dive into four Menlo principles to decide how to treat stakeholders.

- **Beneficence.** We recognize that lower latency and network usage can significantly reduce operational costs and improve operational efficiency, bringing broader access for organizations that rely on blockchains or distributed systems. Although high efficiency also leads to large-scale illegal services, comparable solutions are already widely available and we have found no evidence of such misuse. We believe that our protocols have more positive benefits than negative outcomes due to the higher throughput, strong robustness, lower latency, and less cross-shard network usage. Our solutions can enhance the security and efficiency of distributed systems and promote the fairness of cross-region access. We also prepare the mitigation mechanisms to minimize potential risks, including the long-term monitoring on real-world deployments, the prompt remediation on any identified problems, the transparent disclosure process, and the continuous improvements on our protocols.

- **Respect for persons.** No human subjects or real-world user data were involved. Nonetheless, we consider the well-being of our researchers. We prioritize team well-being by preventing harm, fostering a supportive environment, and promoting balance and diversity.

- **Justice**. To ensure that performance gains are shared equitably, all code will be released on GitHub under the permissive CC BY4.0 license together with step-by-step build scripts, preventing inequitable costs for outside researchers.

- **Respect for law and public interest.** We check the legal risks associated with computer misuse statutes and privacy or intellectual property regulations. Our research adheres to all legal frameworks and data privacy principles, employing only safe experiments. Our work advances the public interest through open-source release and a responsible-disclosure process. Because the study contacts no third party systems and exposes no personal data, it is lawful in every jurisdiction where the authors reside. Publishing the artifacts allows anyone to replicate the experiments, audit our claims, and contribute fixes.

Decision. We decide to complete and publish the research. While acknowledging the potential for misuse to conceal malicious activities, we contend that Logos's benefits in enabling large-scale distributed trust infrastructure with high performance vastly outweigh this risk. Our optimizations push the sharding blockchain more practical in large-scale, real-world applications against adversaries. Because no human subjects are involved, autonomy and privacy rights remain untouched. Legal due diligence confirms compliance with misuse and IP rules, and full transparency serves the public interest by enabling independent audit and continuous improvement. Taken together, the principles of beneficence, respect for persons, justice, and respect for law all support the same conclusion: proceeding with this research and its publication is an ethical course, with an ongoing obligation to monitor, disclose, and remediate any unforeseen harms.

Open Science

We are committed to upholding the principles of open science to ensure the transparency of our research. All research codes and necessary documentation have been published at: https://zenodo.org/records/17855927

Acknowledgment

This paper is supported by the National Key R&D Program of China (2025ZD0808500), the National Natural Science Foundation of China (T2425023, U21B2021, U22B2008, U21A20467, U2241213, 62202027, 62172025, 62472015), Beijing Natural Science Foundation (L251003), Zhejiang Provincial Natural Science Foundation of China under Grant No. LMS25F020014, Open Research Fund of The State Key Laboratory of Blockchain and Data Security, Zhejiang University, the Fundamental Research Funds for the Central Universities, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

References

[1] Harmony: Technical whitepaper, 2018. https://harmony.one/whitepaper.pdf.

[2] Polkadot lightpaper:an introduction to polkadot, 2020. https://assets.polkadot.network/Polkadot-lightpaper.pdf.

[3] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *PODC '19*, pages 337–346. ACM, 2019.

[4] Ittai Abraham and Gilad Stern. Information theoretic hotstuff. In *OPODIS '20*, volume 184, pages 11:1–11:16, 2020.

[5] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. In *NDSS '18*. ISOC, 2018.

[6] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *PODC '22*, pages 399–417. ACM, 2022.

[7] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In *CCS'22*, pages 193–207. ACM, 2022.

[8] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Crypto '21*, pages 524–541. Springer, 2001.

[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[10] Chunjiang Che, Songze Li, and Xuechao Wang. Manifoldchain: Maximizing blockchain throughput via bandwidth-clustered sharding. In *NDSS'25*. ISOC, 2025.

[11] Feng Cheng, Jiang Xiao, Cunyang Liu, Shijie Zhang, Yifan Zhou, Bo Li, Baochun Li, and Hai Jin. Shardag: Scaling dag-based blockchains via adaptive sharding. In *ICDE '24*, pages 2068–2081. IEEE, 2024.

[12] Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. Parbft: Faster asynchronous bft consensus with a parallel optimistic path. In *CCS'23*, pages 504–518, 2023.

[13] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *SIGMOD '19*, pages 123–140. ACM, 2019.

[14] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *SP' 22*, pages 2502–2517. IEEE, 2022.

[15] Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-cdh. In *CRYPTO '24*, volume 14926, pages 251–284. Springer, 2024.

[16] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *USENIX Security '23*, pages 5359–5376. USENIX Association, 2023.

[17] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *CCS '22*, pages 683–696. ACM, 2022.

[18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *SP' 19*, pages 1051–1066. IEEE, 2019.

[19] Sisi Duan, Xin Wang, and Haibin Zhang. FIN: practical signature-free asynchronous common subset in constant time. In *CCS '23*, pages 815–829. ACM, 2023.

[20] Sisi Duan, Haibin Zhang, Xiao Sui, Baohan Huang, Changchun Mu, Gang Di, and Xiaoyun Wang. Dashing and star: Byzantine fault tolerance with weak certificates. In *EuroSys '24*, pages 250–264. ACM, 2024.

[21] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous BFT consensus with throughput-oblivious latency. In *CCS '22*, pages 1187–1201. ACM, 2022.

[22] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In *Eurocrypt '24*, pages 370–400. Springer, 2024.

[23] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous BFT closer to practice. In *NDSS '22*. ISOC, 2022.

[24] Jelle Hellings and Mohammad Sadoghi. Byshard: sharding in a byzantine environment. *VLDB J.*, 32(6):1343–1367, 2023.

[25] Zicong Hong, Song Guo, and Peng Li. Scaling blockchain via layered sharding. *IEEE J. Sel. Areas Commun.*, 40(12):3575–3588, 2022.

[26] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In *INFOCOM '22*, pages 1968–1977. IEEE, 2022.

[27] Huawei Huang, Zhaokang Yin, Qinglin Yang, Taotao Li, Xiaofei Luo, Zhou Lu, and Zibin Zheng. Scalability and security of blockchain-empowered metaverse: A survey. *IEEE Open J. Comput. Soc.*, 5:648–659, 2024.

[28] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT '10*, pages 177–194. Springer, 2010.

[29] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *SP '18*, pages 583–598. IEEE, 2018.

[30] Mingzhe Li, You Lin, Jin Zhang, and Wei Wang. Cochain: High concurrency blockchain sharding via consensus on consensus. In *INFOCOM '23*, pages 1–10. IEEE, 2023.

[31] Pengze Li, Mingxuan Song, Mingzhe Xing, Zhen Xiao, Qiuyu Ding, Shengjie Guan, and Jieyi Long. SPRING: improving the throughput of sharding blockchain via deep reinforcement learning based state placement. In *WWW '24*, pages 2836–2846. ACM, 2024.

[32] Yijing Lin, Zhipeng Gao, Hongyang Du, Dusit Niyato, Jiawen Kang, Ruilong Deng, and Xuemin Sherman Shen. A unified blockchain-semantic framework for wireless edge intelligence enabled web 3.0. *IEEE Wirel. Commun.*, 31(2):126–133, 2024.

[33] Andi Liu, Yizhong Liu, Qianhong Wu, Boyu Zhao, Dongyu Li, Yuan Lu, Rongxing Lu, and Willy Susilo. Cherubim: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains. *IEEE Trans. Inf. Forensics Secur.*, 19:3178–3193, 2024.

[34] Yizhong Liu, Andi Liu, Yuan Lu, Zhuocheng Pan, Yinuo Li, Jianwei Liu, Song Bian, and Mauro Conti. Kronos: A secure and generic sharding blockchain consensus with optimized overhead. In *NDSS '25*. ISOC, 2025.

[35] Yizhong Liu, Andi Liu, Zhuocheng Pan, Yuxuan Hu, Jianwei Liu, Song Bian, Yuan Lu, Zhenyu Guan, Dawei Li, and Meikang Qiu. Realizing corrupted-shard tolerance: A sharding blockchain with preserving global resilience. In *CCS'25*, pages 2099–2113. ACM, 2025.

[36] Yizhong Liu, Jianwei Liu, Yiming Hei, Wei Tan, and Qianhong Wu. A secure shard reconfiguration protocol for sharding blockchains without a randomness. In *Trustcom '20*, pages 1012–1019. IEEE, 2020.

[37] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, et al. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Comput. Sci. Rev.*, 46:100513, 2022.

[38] Yizhong Liu, Boyu Zhao, Xun Lin, Zhenyu Guan, Dawei Li, Jianwei Liu, Qianhong Wu, Willy Susilo, and Robert H. Deng. Quantum-resistant sharding blockchain and its application in secure data transmission. *IEEE J. Sel. Areas Commun.*, 43(8):2732–2746, 2025.

[39] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined BFT. In *CCS '22*, pages 2159–2173. ACM, 2022.

[40] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC '20*, pages 129–138. ACM, 2020.

[41] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *CCS '16*, pages 17–30. ACM, 2016.

[42] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *SP '21*, pages 1348–1366. IEEE, 2021.

[43] Ralph C. Merkle. One way hash functions and DES. In *CRYPTO '89*, volume 435, pages 428–446. Springer, 1989.

[44] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[45] Li Quan. Distributed transparent data layer for next generation blockchains. In *WWW '24*, pages 1178–1181. ACM, 2024.

[46] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *Comput. Commun. Rev.*, 27(2):24–36, 1997.

[47] Alex Skidanov and Illia Polosukhin. Nightshade: Near protocol sharding design, 2019. https://nearprotocol.com/downloads/Nightshade.pdf.

[48] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *NSDI '19*, pages 95–112. USENIX Association, 2019.

[49] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *NSDI'19*, volume 2019, pages 95–112, 2019.

[50] Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. Balanceproofs: Maintainable vector commitments with fast aggregation. In *USENIX Security '23*, pages 4409–4426. USENIX Association, 2023.

[51] Zhaoyang Xie, Haibin Zhang, Shengli Liu, Sisi Duan, and Liehuang Zhu. Practical constant-time asynchronous distributed key generation with improved efficiency. *IEEE Transactions on Information Forensics and Security*, 2025.

[52] Xinxin Xing, Yizhong Liu, Boyang Liao, Jianwei Liu, Bin Hu, Xun Lin, Yuan Lu, and Tianwei Zhang. Gossamer: Lightweight and linear-communication asynchronous (dynamic proactive) secret sharing and the applications. *IACR Cryptol. ePrint Arch.*, page 1516, 2025.

[53] Yibin Xu, Jingyi Zheng, Boris Düdder, Tijs Slaats, and Yongluan Zhou. A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance. In *NDSS '24*. ISOC, 2024.

[54] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *CCS '18*, pages 931–948. ACM, 2018.

[55] Haibin Zhang and Sisi Duan. PACE: fully parallelizable BFT from reproposable byzantine agreement. In *CCS '22*, pages 3151–3164. ACM, 2022.

[56] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft. In *USENIX Security '23*, pages 5341–5357, 2023.

[57] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, Zicong Hong, and Aniket Kate. Front-running attack in sharded blockchains and fair cross-shard consensus. In *NDSS '24*. ISOC, 2024.

A Related Works

Numerous works [5,11,17,25,26,29–31,33–35,38,48,53,57] have studied sharding blockchains, and existing approaches to improving CSTP can be classified as follows.

Kronos. Kronos [34] is the state-of-the-art generic sharding blockchain consensus that realizes strong atomicity and optimized overhead even in asynchronous networks. Briefly, Kronos processes CSTXs through the following phases: request delivery, available input spending (or unavailable input proving for invalid CSTXs), reliable cross-shard certification, and valid request finalization (or invalid request rejection).

Clients first submit their CSTXs, signed by all involved clients, to the corresponding output shards. Output shards conduct a formal examination and forward examined CSTXs to their input shards, preventing malicious clients from selectively sending requests to only a subset of involved shards while neglecting others. After receiving a CSTX, each input shard verifies the input availability and spends available inputs to the output shard *buffer* through BFT. Processed CSTXs are then certified to the corresponding output shards in batch using a $O(n)$-to-$O(n)$ reliable cross-shard batch certification method. The buffer is securely maintained by all parties in the output shard using a threshold signature, ensuring that no input is transferred until all required inputs are received.

Once all inputs of a CSTX are received, the CSTX is deemed valid, and honest nodes in the output shard sign to the validity of this CSTX. The input funds are transferred to the payee from the buffer with these honest nodes' signatures. Invalid transactions are certified to all involved shards in the unavailable input proving phase. In the happy path, invalid requests are comprehensively rejected through cross-shard broadcast. In unhappy paths, where an input has already been spent by the input shard before receiving the invalidity proof, this shard performs a rollback through another round of BFT, known as the "execution-rollback" method in Figure 1.

Other well-known CSTP patterns. Kokoris-Kogias et al. [29], Al-Bassam et al. [5], and Zhang et al. [57] adopt two-phase commit (2PC) to process CSTXs, which includes a prepare phase and a commit phase. Each phase is finished through a BFT in the involved shards, leading to the IS-Overhead of $2k\mathcal{B}$ as analyzed in Section 2. Zamani et al. [54] split CSTX commitment into two individual commitments, which are finished by input shards and output shards successively. Wang et al. [48] propose Monoxide and utilizes the relay transaction mechanism to convert CSTX into ISTXs and decrease the IS-overhead. Monoxide is based on PoW and cannot process invalid multi-input transactions. Li et al. [30] also adopt this method to process CSTXs.

Introducing special shard structures. Hong et al. [25] introduce bridge shards that store the full ledger records of multiple shards. Therefore, the CSTXs involving these shards can be processed and validated by them internally, where the bridge shard serves as the coordinator. The bridge structure limits full state sharding and struggles to accommodate randomly distributed CSTXs without appropriate bridge shards. David et al. [17] exploit a large control chain to monitor all shard operations. Under the supervision of the control chain, shards are initially configured with small sizes and scaled up if the control chain detects a consensus halt (i.e., the shard cannot run BFT securely). This protocol adopts 2PC, where either the clients or the shard leaders interacting with the control chain can act as the cross-shard coordinators. Xu et al. [53] employ a two-layer sharding structure: the network is partitioned into large control shards, each further divided into several smaller process shards. Nodes only participate in one process shard to execute CSTP, under the regulation of the upper control shard. CSTXs are committed using the splitting mechanism, similar to that of Zamani et al. [54].

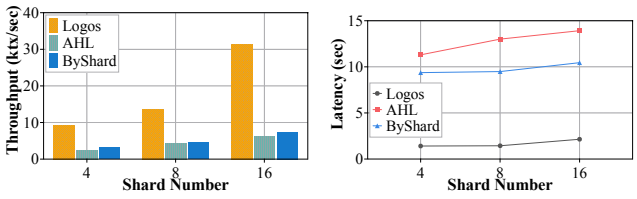
B MVBA used in Logos

Here we introduce the Speeding MVBA used in our Logos proposed by Guo et al. [23]. Speeding MVBA uses a compact structure that significantly reduces communication rounds while maintaining agreement and termination. Unlike traditional MVBA designs such as AMS19 [3], which rely on a three-phase "key-lock-commit" broadcast using four provable broadcasts (PBs), Speeding MVBA compresses this into a two-stage protocol by constructing a Strong Provable Broadcast (SPB) from just two sequential PBs. This allows the protocol to produce both lock and finish proofs with fewer messages. Additionally, it introduces a novel two-phase "Yes-No" voting mechanism that replaces the complex key-lock while ensuring consensus even in adversarial settings.

In optimistic cases, a shortcut mechanism allows honest nodes to immediately terminate, reducing the protocol execution into only 6 rounds. Even in the worst case, the protocol only needs 12 rounds to terminate. Speeding MVBA has significantly fewer rounds compared to other works, which need dozens of rounds, dramatically improving latency and throughput in asynchronous BFT systems.

C Supplementary Experiments

Comparison with other sharding blockchains. We further compare Logos with other state-of-the-art sharding systems, AHL [13] and Byshard [24]. Each shard comprises 50 nodes. Figure 9a and Figure 9b depict the throughput and the latency respectively. Overall, Logos outperforms the two schemes by about 4.2-5.1× throughput with a time cost below 20%.



(a) Throughput (b) Latency

Figure 9: Comparison of Logos and other sharding blockchains.

D Detailed Security Analysis

D.1 Analysis of csdRBC

Lemma 5. *Let $n \geq 3f+1$ hold, the* csdRBC *presented in Algorithm 2 is secure as per Definition 5.*

Proof of Lemma 5. We prove the lemma as follows.

One-State Agreement. In csdRBC, honest node could vote for multiple input availability states with non-conflicting $\mathsf{TX}_{\mathrm{cr}}$ and vote for only one of input availability states with conflicting $\mathsf{TX}_{\mathrm{cr}}$ at the height h. For any two honest nodes, if one outputs the input availability states M and another outputs M' for proposer $\mathcal{P}_j$, then both M and M' receive at least $2f+1$ ECHO message votes. That is to say, when $n \geq 3f+1$ holds, if M does not equal to M', then at least one honest node votes for M and M' at the same height h, leading to a contradiction. Therefore, csdRBC satisfies *one-state agreement*.

Totality. In csdRBC, if one honest node $\mathcal{P}_i$ outputs the input availability states M, $\mathcal{P}_i$ must receive at least $2f+1$ READY messages for M which means that at least $f+1$ honest nodes send READY messages for M. Then, all honest nodes can receive $f+1$ READY messages from these honest nodes and broadcast READY messages. Finally, all honest nodes can receive $2f+1$ READY messages and output M. Therefore, csdRBC satisfies *totality*.

Cross-State Non-Conflicting Validity. In csdRBC, according to Lines 5-6 of Algorithm 2, each honest node $\mathcal{P}_i$ can only vote once for conflicting transactions. If one honest proposer $\mathcal{P}_j$ proposes an input availability state M, and another honest proposer $\mathcal{P}'_j$ proposes a conflicting M′, then only one of M and M′ can receive at least $f+1$ honest votes. Although f malicious nodes can double-vote, at most one message can receive more than $2f+1$ votes and finally be outputted. Therefore, csdRBC satisfies *cross-state non-conflicting validity*. □

D.2 Analysis of RPST

Lemma 6. *Let $n \geq 3f+1$ hold in both sender shard S_s and receiver shard S_r, then* RPST *presented in Algorithm 3 is reliable as per Definition 6.*

Proof of Lemma 6. In RPST, if each node $\mathcal{P}_i$ in the sender shard inputs the same $\langle \mathsf{M}, \mathsf{tree}, \langle \mathsf{rt} \rangle_{\mathsf{S}_s} \rangle$, then all honest nodes in S_s will send the correct fragment with the certificate $\langle [\mathsf{M}]_i, \mathsf{mproof}_i, \langle \mathsf{rt} \rangle_{\mathsf{S}_s} \rangle$. In the worst case, at most f wrong fragment will be delivered to f honest receivers by f malicious senders in Step ④. However, according to the *unforgeability* of signatures and the *collision resistance* of the hash function, f honest receivers can verify the certificate and discard the wrong fragments. Besides, f malicious receivers try to broadcast wrong message fragments in Step ⑤. However, according to the *unforgeability* of signatures and the *collision resistance* of the hash function, these f wrong fragment will be discarded by all honest receivers. Finally, at least $f+1$ correct fragments $[\mathsf{M}]_i$ delivered between honest (sender, receiver) pairs can be received and M can be decoded and outputted correctly by all honest receivers. □

D.3 Analysis of Logos

Lemma 1 (Consistency in a single shard)**.** *Let $n \geq 3f+1$ hold in each S_i, then* Logos *satisfies consistency in a single shard as per Definition 1.*

Proof of Lemma 1. For intra-shard transactions, the consistency is ensured by the BFT *safety* property. We analyze the consistency for CSTP in a single shard as follows.

Consistency for CSTP in input shards. In Logos, each input shard S^{in} only commits the valid CSTXs on $\mathsf{S}^{\mathsf{in}}.\mathsf{log}$ in the agreement phase by invoking MVBA. If an honest node $\mathcal{P}_i$ commits $\mathsf{TX}_{\mathsf{cr}}$ on the height h, then for any other honest node $\mathcal{P}_j$ in S^{in} that commits $\mathsf{TX}'_{\mathsf{cr}}$ on the same height h on ledger $\mathsf{S}^{\mathsf{in}}.\mathsf{log}$, it must hold that $\mathsf{TX}_{\mathsf{cr}} = \mathsf{TX}'_{\mathsf{cr}}$ according to the *agreement* property of MVBA.

Consistency for CSTP in output shards. In Logos, output shard commits valid CSTXs in the agreement phase by invoking BFT. If an honest node $\mathcal{P}_i$ commits $\mathsf{TX}_{\mathsf{cr}}$ on the height h, then for any other honest node $\mathcal{P}_j$ in $\mathsf{S}^{\mathsf{out}}$ that commits $\mathsf{TX}'_{\mathsf{cr}}$ on the same height h of the ledger $\mathsf{S}^{\mathsf{out}}.\mathsf{log}$, it must hold that $\mathsf{TX}_{\mathsf{cr}} = \mathsf{TX}'_{\mathsf{cr}}$ according to the *safety* property of BFT.

Finally, consistency in a single shard holds for Logos. □

Lemma 2 (No conflict)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies no conflict as per Definition 1.*

Proof of Lemma 2. We analyze this property as follows.

No conflict in one shard. In Logos, if two honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ in one shard propose two conflicting CSTXs $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$ with the same input, they first invoke the csdRBC to output the input availability states M and M′ for $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$. However, according to *cross-state non-conflicting validity* property of csdRBC, at most one of M and M′ can be outputted. If neither of them is outputted, then in the following rounds, all honest nodes in two different shards can finally receive $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$. Each shard can solve the conflict. If one of them (assume it is $\mathsf{TX}_{\mathsf{cr}}$) is outputted as having available inputs and committed as a valid CSTX on the shard ledger finally, then the state change occurs, which means $\mathsf{TX}'_{\mathsf{cr}}$ will be outputted as having unavailable input. Finally, no conflicting transactions will be committed on the shard ledger.

No conflict in multiple shards. In Logos, if two honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ in two different shards propose two conflicting transactions $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$ with the same input, they first invoke the csdRBC in there own shard to output $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$. Then all honest nodes in each input shard $\mathsf{S}^{\mathsf{in}}_i$ send the result to all other involved shards $\{\mathsf{S}^{\mathsf{in}}, \mathsf{S}^{\mathsf{out}}\}$ by invoking RPST. According to the *reliability* of RPST, all honest nodes in two different shards can receive $\mathsf{TX}_{\mathsf{cr}}$ and $\mathsf{TX}'_{\mathsf{cr}}$. Then each shard can solve the conflict. If one of them (assume it is $\mathsf{TX}_{\mathsf{cr}}$) is outputted as having available inputs and committed as a valid CSTX on the two shard ledgers finally, then the state change occurs, which means $\mathsf{TX}'_{\mathsf{cr}}$ will be outputted as having unavailable input. Finally, only $\mathsf{TX}_{\mathsf{cr}}$ can be considered as valid in two shards and be committed on both ledgers.

Therefore, no conflict holds for Logos. □

Lemma 3 (Atomicity for cross-shard transactions)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies atomicity for cross-shard transactions as per Definition 1.*

Proof of Lemma 3. We analyze these properties from valid and invalid transactions as follows.

Atomicity for valid CSTXs. For a valid CSTX $\mathsf{TX}_{\mathsf{cr}}$, according to Algorithm 1, all honest nodes in each input shard $\mathsf{S}^{\mathsf{in}}_i$ first invoke csdRBC. By the *one-state agreement* and *totality* properties of csdRBC, all honest nodes can finally output the input availability states with the certificate and lock the account balance for $\mathsf{TX}_{\mathsf{cr}}$. Then all honest nodes in each input

shard $\mathsf{S}_i^{\mathsf{in}}$ send the available input states to all other involved shards $\{\mathsf{S}^{\mathsf{in}}, \mathsf{S}^{\mathsf{out}}\}$ by invoking RPST. According to the *reliability* of RPST, all honest nodes in all involved shards can receive all available input states of $\mathsf{TX}_{\mathrm{cr}}$. Then input shards run MVBA to commit $\mathsf{TX}_{\mathrm{cr}}$. By the *termination* and the *agreement* properties of MVBA, all input shards will finally commit $\mathsf{TX}_{\mathrm{cr}}$ on the shard ledger and change the state. Meanwhile, the output shards invoke BFT to commit $\mathsf{TX}_{\mathrm{cr}}$. According to the *safety* and *liveness* properties of BFT, the output shards will finally commit valid $\mathsf{TX}_{\mathrm{cr}}$ on the shard ledger.

Atomicity for invalid CSTXs. For an invalid CSTX $\mathsf{TX}_{\mathrm{cr}}$, according to Algorithm 1, all honest nodes in each input shard $\mathsf{S}_i^{\mathsf{in}}$ first invoke csdRBC. By the *one-state agreement* and *totality* properties of csdRBC, all honest nodes can finally output the input availability states with the certificate. Then all honest nodes in each input shard $\mathsf{S}_i^{\mathsf{in}}$ send the input availability states of $\mathsf{TX}_{\mathrm{cr}}$ to all other involved shards $\{\mathsf{S}^{\mathsf{in}}, \mathsf{S}^{\mathsf{out}}\}$ by invoking RPST. Similarly, by the *reliability* of RPST, all honest nodes in all other involved shards can receive the unavailable input states of $\mathsf{TX}_{\mathrm{cr}}$. After that, all honest nodes in all input shards unlock the account balance for available inputs and no state changes happen in input shards. Meanwhile, all honest nodes in all output shards discard $\mathsf{TX}_{\mathrm{cr}}$ directly and no state changes happen in output shards.

Hence, atomicity holds for Logos. □

Lemma 4 (Liveness)**.** *Let $n \geq 3f+1$ hold in each S_i,* Logos *satisfies liveness as per Definition 1 with κ rounds.*

Proof of Lemma 4. Liveness for intra-shard transactions is ensured by the BFT *liveness* property with $\kappa = \kappa_{\mathsf{BFT}}$. We prove this theorem for CSTP by analyzing the liveness parameter for non-conflicting transactions since all conflicts between CSTXs can be solved by the *no conflict* property proven in Lemma 2.

Liveness for valid CSTP. For a valid CSTX $\mathsf{TX}_{\mathrm{cr}}$, each input shard S^{in} first runs one round csdRBC with $\kappa = 3$ (Step ①-③ in Algorithm 2), and one round RPST with $\kappa = 2$ (Step ④-⑤ in Algorithm 3). After that, each input shard S^{in} runs the MVBA with $\kappa = \kappa_{\mathsf{MVBA}}$ and each output shard $\mathsf{S}^{\mathsf{out}}$ runs BFT with liveness parameter $\kappa = \kappa_{\mathsf{BFT}}$. In asynchronous networks, $\kappa_{\mathsf{BFT}} > \kappa_{\mathsf{MVBA}}$ [23]. Therefore, the liveness parameter κ holds that $\kappa = \kappa_{\mathsf{BFT}} + 5$.

Liveness for invalid CSTP. For an invalid CSTX $\mathsf{TX}_{\mathrm{cr}}$, each input shard S^{in} rejects it with one round csdRBC with $\kappa = 3$, and one round RPST with $\kappa = 2$. Then all shards discard $\mathsf{TX}_{\mathrm{cr}}$. Therefore, the liveness parameter κ holds that $\kappa = 5$. □