Lipper: Leveraging Safety and Liveness for Sharded Blockchain under Time-Varying Adversarial Population

The author list is suppressed for double-blind reviews

***Abstract*—Sharding plays a crucial role in improving the transaction throughput of vote-based blockchains. However, existing protocols adopt a pessimistic approach, always attempting to withstand a fixed upper-bound number of adversarial nodes. This kind of approach fails to exploit opportunities to maximize transaction throughput when fewer adversaries are actually present at runtime. This paper presents Lipper, a novel sharding protocol that can achieve a trade-off between maximizing transaction throughput and defending against the runtime number of adversaries. More specifically, Lipper's consensus derivation always guarantees the network's safety without sacrificing the transaction throughput as long as the number of adversaries is under the safety threshold. On the other hand, Lipper can gracefully adapt the transaction throughput according to the number of adversaries attacking the liveness of the network. When there are more (or fewer) adversaries, Lipper will spend more (or less) effort achieving liveness, lowering (or increasing) the transaction throughput. This nice property is attributed to Lipper's novel multi-phased consensus derivation protocol. Extensive experimental results demonstrate that, compared to the state-of-the-art approaches, Lipper achieves superior transaction throughput and robustness in the presence of a time-varying adversary population.**

***Index Terms*—Blockchain, Sharding, Scalability, Adversary nodes, Adaptive sharding protocol**

I. Introduction

Blockchain sharding is a promising technique for enhancing the scalability of vote-based blockchain systems by partitioning the network into smaller units known as shards. The primary objective of sharding is to increase parallelism and reduce consensus overhead within each shard, thereby improving overall efficiency [3], [5]–[7], [9]–[13], [16], [19], [20].

The security of a blockchain is characterized by two crucial properties: liveness and safety. Liveness refers to the shard's ability to deliver new messages to all participating peers eventually. Safety pertains to achieving consensus on the order of output messages. The safety threshold (S) and liveness threshold (L) represent the maximum permissible ratio of adversarial participants within a shard to ensure safety and liveness, respectively.

A fundamental trade-off exists between parallelism and security in blockchain sharding. Given the corruption level of the whole network, the probability that a shard containing a certain ratio of adversarial nodes increases inversely with the shard size. In other words, ensuring that the shard is secure with a high probability requires the shard size not be too small. Consequently, sharding solutions like [19] have to use relatively large shard sizes, limited by the safety and liveness thresholds. These solutions primarily guarantee security with equal liveness and safety thresholds (e.g., $L, S < 50\%$ in a synchronous system), assuming an equal adversary's interest in attacking both safety and liveness. Unfortunately, this approach leads to limited parallelism due to the large shard size, which is suboptimal, especially when the runtime adversarial behavior is less severe than the worst-case scenario.

Recent proposals [5], [17], [18] focus on enabling smaller shards by adjusting the values of S and L within each shard at runtime. Instead of using large shards, these approaches allow for a higher S value for smaller shards, thus keeping the probability of safety violations low. However, a shard with a higher S would be more susceptible to attacks on liveness. For instance, even if the ratio of corruption is lower than S, an adversary can still disrupt liveness by disagreeing or remaining silent regarding all proposals. It has been shown that $S < 1 - L$ and $S < 1 - 2L$ hold in a synchronous and a partially synchronous system, respectively [5].

As depicted in Figure 1, the approaches presented in [5], [18] initiate with a small shard size with a high S and low L, assuming that the ratio of adversary targeting liveness is low. This maximizes the system's parallelism. When the ratio of nodes attacking liveness exceeds L at runtime, shards can be enlarged to increase L and decrease S, thus restoring the system's liveness. This can be achieved either by reconstructing the shards entirely, changing the shard memberships of all nodes [18], or by performing local shard adjustments, leading to the formation of overlapping shards [5].

In practical scenarios, the adversarial population is time-varying. For instance, an otherwise honest node may be temporarily irresponsive during voting due to poor network connectivity, causing it to fall out of sync and miss an epoch. However, once the node is fully synchronized, it will resume its normal behavior. In addition, in some epochs, some adversarial nodes may intentionally halt the blockchain's progress by refusing to participate or voting for adversarial blocks. But, in other epochs, they may behave normally. These dynamic behaviors result in a constantly changing and unpredictably adversarial population against liveness. Under such situations, the aforementioned approaches of adjusting shard sizes and memberships can lead to frequent and costly shard membership adjustments and the extra overhead of overlapping shards. These vulnerabilities undermine the effectiveness of these approaches.

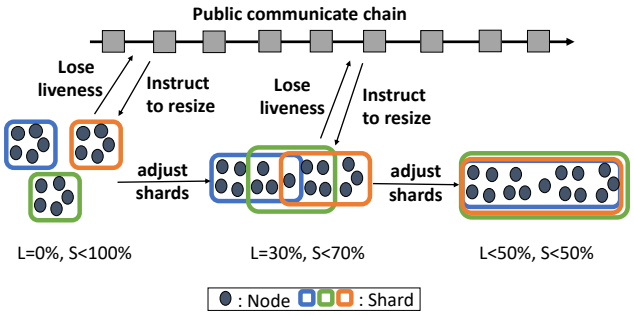

Fig. 1. A typical structure of a sharding approach that incorporates liveness and safety thresholds. As the liveness threshold L increases, the size of shards also increases. In this system, three shards are present. The increased shard size either leads to a reduction in the number of shards or results in overlapping shards.

In this paper, we address the limitations of existing approaches and present Lipper, a novel sharded blockchain protocol that adapt its behavior according to a time-varying adversarial population while minimizing overhead. Lipper achieves high parallelism by allowing small shards with a high S and a low L. One of its salient features is that if the adversary ratio in a shard surpasses L, it restores liveness without necessitating global or local shard reconstructions. Instead, it employs a novel two-layer voting mechanism, which uses an additional voting process to revive the stalled shards. As the overhead of the additional voting is proportional to the number of stalled shards, the system's transaction throughput adjusts gracefully according to the runtime adversarial population.

Furthermore, we implement a prototype of Lipper and experimentally and analytically evaluate its performance compared to two state-of-the-art approaches, namely Gearbox [5] and Rapidchain [19]. Our experiments and analysis demonstrate that Lipper outperforms both approaches significantly regarding transaction throughput and storage requirements.

The remainder of this paper is structured as follows. Section II provides an overview of Lipper. Afterwards, Section III describes the system model and threat model. Section IV presents the details of Lipper about bootstrapping, blockchain structure, consensus stage, cross-shard transaction and determination of the parameters. Section V reports the evaluation results for our prototype. Section VI analyses related work. Section VII concludes the paper.

II. Overview

Lipper is a two-layer, multi-phased sharding protocol designed to address the limitations of existing approaches. It dynamically adjusts the liveness and safety thresholds and adapts to the time-varying adversarial population at runtime, unlike classical protocols such as Rapidchain [19] and Omniledger [11] that assume a constant upper-bounded adversarial population. Contemporary methods like Gearbox [5] assume a static adversarial population below the upper-bounded limit. However, Gearbox's default one-way scaling from smaller to larger shard sizes can lead to overlapping shards. Enabling two-way scaling in Gearbox results in frequent readjustment of shard memberships, which deteriorates the transaction throughput of the system. In contrast, Lipper enables two-direction scaling, automatically adjusting the transaction throughput based on the time-varying adversarial population. It effectively avoids adjusting either shard size or the number of shards and also prevents overlapping shard memberships.

Lipper consists of two layers: the work shards (first layer) and the committee shards (second layer). The work shards, with $S < 100\%$ and $L = 0\%$, contain a small number of nodes and govern a range of transactions (wallet accounts). The committee shards, with $S = L < 50\%$, govern multiple work shards. Shards of the same layer do not share overlapping members. Each node is randomly assigned to exactly one work shard and the committee shard that governs it.

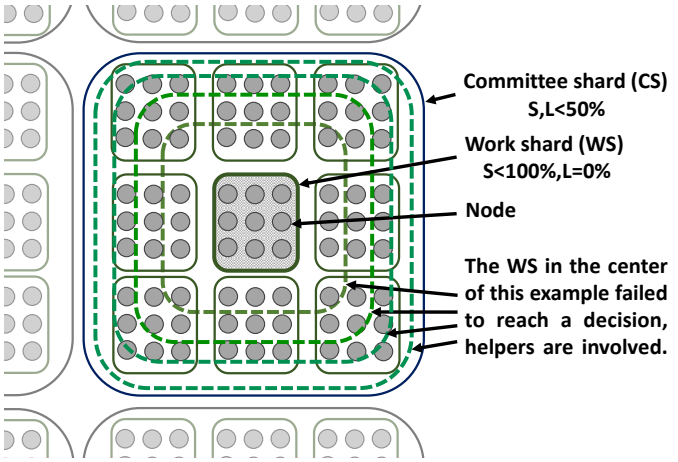

Fig. 2. The structure of Lipper. When a work shard (WS) cannot reach a consensus decision for a block, additional nodes (a.k.a. helpers) will proactively sync the block and the relevant data to participate in the voting process. S is adjusted after new nodes participate in the voting.

Lipper operates through a multi-phased protocol with five phases, as shown in Figure 2. Each work block goes through at least one phase and at most all five phases. At each phase $i, i \in [1, 5]$, more than $S < (100 - (i-1) \times \frac{50}{5-1})\%$ of the involved nodes must agree on the same verdict in order to establish consensus for the block. The safety threshold of all work shards is $S < 100\%$, indicating that a work block will be accepted only if there is a unanimous vote among the nodes in the work shard. If the work block does not reach a unanimous agreement in the first phase, the voting process advances to the second phase with $i = 2$. In this phase, the safety threshold is adjusted to $S < 83.3\%$, and additional helper nodes join the voting process. If a consensus decision still cannot be made, the voting process continues to advance phases with new helpers added and S decreased.

The number of phases needed to reach a consensus in Lipper depends on the number of runtime adversarial populations. Lipper is resilient to the dynamic decrease or increase in the adversarial population, enabling higher or lower transaction throughput. As the adversarial population decreases or increases, fewer or more phases will be needed, resulting in a

lower or higher cost for nodes and higher or lower transaction throughput. Lipper achieves its highest transaction throughput when no adversaries are attacking liveness because there are no failed work blocks that require helpers.

Unlike the previous approach by Gearbox [5], which dissolves shards and forms completely new shards with larger sizes, Lipper does not dissolve any shards. The helpers for each phase of a shard are determined based on the randomly assigned node index during the system bootstrap. In other words, the helpers for a shard remain fixed. To enable quick synchronization, each shard in Lipper operates as a state-block-state style blockchain, where blocks update the states. The helper nodes only sync the latest confirmed state and the latest block, and they can evaluate the correctness of the block based on the confirmed state. In the worst case, a decision can be made in the last phase when $L, S < 50\%$, indicating that the majority of nodes in the shard are honest.

III. System and threat model

A. System Model

Lipper operates in a N-node network with $\lfloor N/N_c \rfloor$ committee shards and $\lfloor N/N_w \rfloor$ work shards where the size of committee shard and work shard denoted as N_c and N_w. Each node, denoted as $node_i$ is in exactly one work shard and the committee shard that governs this work shard. Each $node_i$ employs a public-private key pair (pK_{node_i}, sK_{node_i}), which the public key pK_{node_i} serves as a unique identifier for the node within the Lipper network. These interconnected nodes utilize a standardized synchronous peer-to-peer network architecture. We assume that nodes can establish identities through any Sybil-attack-resistant mechanism.

The messages are disseminated across the network via a synchronous gossip protocol [8]. This protocol guarantees a well-defined upper bound, denoted as Δ, on the maximum delay of message propagation. It is important to note that the gossip protocol does not necessarily maintain the order of message delivery. Moreover, Lipper adopts the account/balance model to represent the state of the ledger.

B. Threat Model

Lipper is engineered to withstand adversarial nodes, tolerating a maximum of $f \leq \lfloor (N-1)/3 \rfloor$ adversaries at any time. This tolerance level is critical for handling various incidents such as node disconnections, internal failures, network jitters, and actions from probabilistic polynomial-time Byzantine adversaries. These adversaries are capable of collusion and engaging in protocol violations including denial of service, tampering, forgery, and message interception.

Drawing on established practices from other sharding systems like [13], [11], and [19], Lipper adopts the concept of a slowly-adaptive adversary. This approach implies that the set of adversarial and honest nodes remains fixed during each epoch, only changing between epochs. Such an assumption allows Lipper to maintain robust security measures over each epoch's duration, adapting its defenses in the transition to subsequent epochs. This strategy ensures the resilience and integrity of the network, even in the face of evolving adversarial tactics.

IV. Lipper protocol

A. Bootstrapping

Lipper employs a bootstrapping design that assigns nodes to shards randomly and unbiasedly, which consists of Randomness Generation and Nodes Assignments.

Randomness Generation: A randomness without bias allows the nodes to be assigned into shards without manipulation or prediction by any node. Various methods for randomness generation exist [2], [3], [14], which satisfy the public-verifiable, bias-resistant and unpredictable requirements. These methods are also employed in other sharding systems [11], [19]. It is important to note that the randomness generation process can be considered a separate module within a sharding system and is not the main focus of our work. Therefore, we do not delve into further details about this process. Here, we choose [4] utilizing randomness.

Nodes Assignments: A predefined list of N nodes is initially shared among participants. A random sequence C is then collectively generated using a discussed randomness generation method. This sequence has a length of N, corresponding to the number of nodes in the system. Using the publicly available sequence C, each node can compute its ws-id (work shard ID) and cs-id (committee shard ID), indicating their respective shard assignments. Furthermore, nodes can determine their index within the committee shard, which determines their role as a helper for the corresponding work shard. It is important to note that the publicly available sequence allows each node to compute not only its own IDs but also the IDs of other nodes in the system.

In Lipper, we adopt a fixed sharding approach similar to the ones used in Rapidchain [19] and Omniledger [11]. However, to accommodate changes like adding new nodes or removing existing ones, the system may undergo periodic restarts every few days, in line with the initial design philosophy. Additionally, inspired by Rapidchain, we consider the possibility of rotating nodes between shards at regular intervals, possibly every few epochs. This rotation strategy is a precautionary measure to prevent an accumulation of adversarial nodes in any single shard, assuming adversaries are capable of such infiltration.

B. Blockchain Structure

In Lipper, there is a blockchain for each work shard (WS) and a blockchain for each committee shard (CS). Each node in the network is responsible for synchronizing and maintaining both the blockchain and the blockchain of the WS and CS where it belongs to.

Blockchain in Work Shard: In contrast to traditional block-connected chains, our blockchain in WS employs a State-block-State structure. More specifically, a State is linked after each WS Block, which does not store detailed transaction information but solely captures the account balances after the execution of the current WS Block. When verifying the

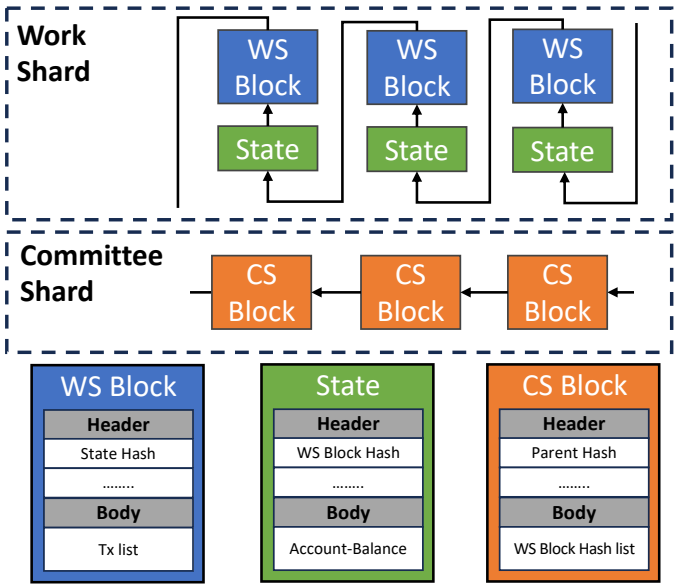

Fig. 3. The structure of blockchain, block, and state in WS and CS. WS shard uses State-Block-State structure, while CS shard adopts block-block structure. WS Block includes a header (last State Hash, etc.) and a body (TX list). The State's header contains the last WS Block Hash and the Account-Balance list. CS Block's header contains the Hash of previous CS Blocks, etc. while its body includes the Hash value of WS Blocks reaching consensus.

correctness of a WS Block, it is sufficient to examine the list of account balances in the corresponding previous State.

Blockchain in Committee Shard: The CS blocks in the blockchain in CS do not record transaction data. Instead, they store the Hashes of the WS Blocks that have achieved consensus among all WSs under their jurisdiction within a specific time period. The primary objective is to ensure that all consensus WS Blocks are known to all nodes within the CS, thereby mitigating the potential for malicious attacks in cross-shard transactions.

Fig. 3 shows the blockchain, block, and state structures in WS and CS.

C. Consensus stage

Definition: Lipper has up to five phases ($phase_i$, $i \in [1,5]$). These five phases are in chronological order. If consensus cannot be reached in $phase_i$, it proceeds to phase $phase_{i+1}$. However, once consensus is achieved in $phase_i$, it does not advance to the next phase. Instead, it initiates a new epoch.

Any consensus should be completed within a time-bound, denoted as T. Specifically, for each phase $phase_i$, we have these following parameters:

- Num_i: the number of nodes that have to vote until $phase_i$ ends;
- T_i: the time-bound that $phase_i$ should be completed;
- S_i: the Safety threshold of $phase_i$, and we can calculate the Liveness threshold L_i by ($L_i < 1 - S_i$).

Workflow: After the random assignment of nodes to the work shard and committee shard through bootstrapping, Lipper proceeds to the consensus stage. During this stage, Lipper operates with concurrent execution of each committee shard. Within each committee shard, the work shards also run concurrently.

1) Leader selection:

a) Randomized Leader Election:: Lipper's protocol dictates that the leading node for both the WS and CS is determined by the index order within the sequence C. The node bearing the lowest index naturally assumes the leadership position. Given that the indices are randomly assigned, the selection of the leader node is equally random, which prevents any deterministic predictability that could be exploited by adversaries.

b) Leader Replacement Strategy:: Should a leader propose a block that fails to achieve consensus, indicating a possible fault or malicious intent, the protocol specifies an automated leader replacement process. Honest nodes, through a predefined agreement, will supersede the compromised leader with the next lowest-indexed node not identified as adversarial. This mechanism ensures a resilient leadership that maintains the network's integrity even in the face of potential security breaches.

2) Consensus in Work Shard: The consensus mechanism in a WS unfolds in several phases, with the initial phase being pivotal to the progression of the shard's state. The leader, determined by the lowest index within the shard, undertakes the task of gathering transactions from the network. Upon successful collection, the leader compiles these into a WS Block and distributes it across the shard's nodes.

The standard protocol for achieving consensus within a WS is a binary, vote-based system that operates synchronously, as detailed in [15]. Within the designated time window T_1, consensus must be achieved. This is quantified by the formula ($S_1 \times Num_1$), where S_1 represents the required percentage of agreement, set at 100%, and Num_1 is the total count of nodes within the WS, denoted as N_w.

The outcomes of this primary phase of consensus are binary:

- **Consensus reached on the WS Block:** Should the WS Block garner universal acceptance, the shard transitions into a new epoch, marking the commencement of a subsequent WS Block's generation.
- **Consensus cannot be reached on the WS Block:** A lack of consensus reached triggers the advancement to subsequent phases, denoted as $Phase_j$, where j ranges from 2 to 5.

In the event that the WS escalates to $Phase_j$, additional nodes termed 'Helpers' are solicited to review the preceding WS Block's State. These helpers, amounting to ($Num_j - Num_{j-1}$) in number, cast their votes, striving to attain consensus with a newly defined Safety threshold, S_j. It is crucial to note that each helper is permitted a singular vote in respective WS shard. Votes from prior phases are not discarded but rather are aggregated with the votes garnered in $Phase_j$, thereby employing a cumulative voting strategy.

Should $Phase_j$ conclude with a consensus reached under defined Safety threshold, S_j, the WS moves forward as per the agreed-upon decision. Contrarily, a consistent inability to

reach consensus necessitates the selection of a new leader, following the predefined protocol for leader succession. This multilayered phase approach ensures the robustness and reliability of the consensus process, adapting dynamically to the evolving state of the shard.

3) Helper: Helpers participating in a particular WS only need to access the State of the previous epoch to vote for the current WS Block. Unlike the nodes within the WS, the helper does not need to store or synchronise all data. In Lipper, whenever a node casts a vote on a WS Block, it is required to send this vote to all other nodes within the same CS. As a result, at any given moment, a node can obtain information about the status of all the WS under the same CS, including whether consensus has been reached and which phase each WS is currently in. By utilizing this information, a node can determine which phase it should participate in and act as a helper for any WS within the CS.

4) Committee shard: In the Lipper, the role of the CS leader centers around aggregating the Hashes of all WS Blocks that have successfully reached consensus within the CS. This process is pivotal for creating the CS Block, an aggregated record encapsulating the collective state of the shard, indicative of the consensus status across the various WSs.

The CS Block is then disseminated to all nodes within the CS employing a gossiping protocol. This protocol is selected for its efficiency in peer-to-peer network communications, ensuring that the CS Block is distributed rapidly and reliably to each node within the shard, thus maintaining synchronized ledger states.

Consensus on the CS Block is achieved using a binary vote-based synchronous protocol within a time bound T. A crucial aspect is the Safety threshold, set at 50%. This threshold ensures that a simple majority is sufficient for decision-making, which is critical for the consistent operation of the CS. By setting the threshold at 50%, Lipper ensures that the CS can always reach a consensus, thereby maintaining the stability and functionality of the overall system. This design choice is essential for preventing deadlock and guaranteeing the smooth running of the Lipper blockchain.

A distinctive feature of Lipper's design is the independent operation of consensus mechanisms in WS and CS shards. This independence allows each shard to function autonomously, ensuring that the operation in one shard does not impede or depend on the progress in the other.

D. Cross-shard transaction

In Lipper, cross-shard transactions facilitate data transfer between distinct work shards. To ensure the authenticity and timeliness of records from other work shards, the acceptance of the corresponding committee block is crucial for the successful finalization and usability of transactions within the work block. Similar to shards in other sharding approaches, the committee shard allows for direct integration of cross-shard transaction protocols commonly used in traditional sharded blockchains. Established methodologies like Omniledger [11], RapidChain [19], and Monoxide [16] provide effective techniques for processing cross-shard transactions. These proven approaches can be seamlessly incorporated into the design of Lipper, ensuring efficiency and compatibility with existing solutions.

In this proposed design, we draw inspiration from RapidChain, recognizing that cross-shard transaction concepts have been explored previously without claiming novelty. Cross-shard transactions involve participant identifiers, such as wallet addresses for sending funds or information, and addresses for receiving funds. Individual transactions, denoted as Tx'_{cross}, occur between sender and receiver addresses in different shards. The initiation of Tx'_{cross} is performed by the sender shard WS'_{sender} and transmitted to the receiver shard WS'_{receiver}.

When processing Tx'_{cross}, the transaction is first acknowledged and recorded by WS'_{sender} in the corresponding WS block, deducting the specified amount from the sender's address. The CS then records the hash of this WS block as proof of deduction. Subsequently, a proof is sent to WS'_{receiver}, which writes it to the corresponding block, confirming the successful receipt of funds or information by the receiver.

E. Determine the parameters

S_i: In the first phase, the objective is to achieve consensus on the work shard with the condition $S_1 < 100\%$. Similarly, in the fifth phase, consensus is sought on the committee shard with the condition $S_5 < 50\%$. The values of S_i for the remaining phases can be computed by:

$$S_i < (100 - (i-1) \times \frac{50}{5-1})\% \tag{1}$$

Num_i: $N_w = Num_1$ and $N_c = Num_5$. The Num_1 can be computed by:

$$Num_1 = \log_v(P_{f1}) \tag{2}$$

where the maximum ratio of adversarial population in the system is $v < 1/3$. P_{fi} represents the probability for a particular work shard to be compromised in $phase_i$. $Num_j, j \in [2, 5]$, can be computed by:

$$P_{fj} = \sum_{i=\lfloor (S_j)(Num_j) \rfloor + 1}^{Num_j} \binom{Num_j}{i} v^i (1-v)^{Num_j - i} \tag{3}$$

Specifically, this equation calculates the probability that a shard, randomly selected from a global population with an adversarial fraction v, comprising Num_j nodes, has an adversarial population exceeding the fraction S_j. It sums the probabilities of all scenarios where the proportion of adversarial nodes within this subset surpasses the critical threshold S_j, taking into account the exact likelihood of each possible combination of adversarial and honest nodes within the shard.
Security threshold: We define that the system is failed if an illegal block is accepted in either the WS or CS of the system. We denote the probability of system failure as Pf. It can be computed as:

$$(P_{f1} \times \lfloor N/Num_1 \rfloor + P_{f5} \times \lfloor N/Num_5 \rfloor) \leq Pf$$
$$\leq \left(\sum_{i=1}^{5} (P_{fi}) \times \lfloor N/Num_1 \rfloor + P_{f5} \times \lfloor N/Num_5 \rfloor \right) \quad (4)$$

This equation estimates the system failure probability, Pf, by considering both the lower and upper bounds. The lower bound is determined by the product of the probability of a single shard being compromised, both in the first phase (P_{f1}) and the fifth phase (P_{f5}), and the number of such shards in the system. This lower bound reflects the best-case scenario where all consensus decisions are successfully made in the first phase, within the WS, without the need for additional helper nodes, and then confirmed in the CS. The upper bound, on the other hand, aggregates these probabilities across all five phases ($\sum_{i=1}^{5}(P_{fi})$) and additional the fifth phase (P_{f5}) for being confirmed in the CS. This represents the worst-case scenario where consensus attempts in every phase are exposed to the risk of being compromised by adversarial nodes. In this scenario, each attempt to reach consensus, from the first phase to the fifth, carries a risk of adversarial correction, cumulatively adding up to the maximum possible system failure probability.

Given the uncertainty of when WS reaches consensus, we can only derive the lower bound and the upper bound for Pf by considering the worst and the best scenario. If we determine N, the value of Num_i could be calculated. We denote λ as the security parameter, where $Pf \leq 2^{-\lambda}$.

T_i: Establishing the time-bound for synchronous protocols poses an ongoing challenge for protocol designers. The selection of an optimal time-bound necessitates striking a balance between redundancy and network efficiency. Longer time-bounds offer redundancy but can impede nodes from staying synchronized with network communication. Prior studies, such as Rapidchain [19], have utilized pre-scheduled consensus among committee members to determine a new time-bound Δ on a weekly basis. Consequently, Lipper could adopt a similar approach to achieve consensus on updated values for T and T_1. If we have determined T_1 and T, $T_j, j \in [2, 5]$ could be computed as:

$$T_j = T_1 + \frac{T - T_1}{Num_5 - Num_1} \times (Num_j - Num_1) \quad (5)$$

V. Evaluation

A. Implementation and Setup

We have developed a prototype of Lipper using Go. To facilitate comparative analysis, we have also implemented a single-layer sharding protocol similar to Rapidchain, as Rapidchain [19] itself is not open source. Moreover, as Gearbox [5] does not provide any implementation and the authors did not present any details on how it can be implemented, we can only simulate Gearbox's functionality in our experiments.

Our experiment utilized a testbed of fifteen servers, each equipped with 32-core AMD EPYC 7R13 processors running at 3.30 GHz and offering 128 vCPUs. The servers were equipped with 256 GB of memory. To mimic typical blockchain testbed setups, we set the bandwidth of all connections between nodes to 20 Mbps, with a latency of 200 ms.

Each Block of Rapidchain, Gearbox and the work shard in Lipper contains up to 4096 transactions, with each transaction being 512 bytes in size. For simplicity, we assume each block size $2Mbytes$. The security parameter $\lambda = 22$, which means Pf was set to be less than 10^{-7}. v represents the maximum ratio of adversarial nodes in the system, where $v < \frac{1}{3}$.

B. Metrics

The primary objective of our evaluation is to assess and compare the performance and efficiency of various protocols in terms of throughput and storage requirements. These metrics are defined and computed as follows:

Throughput: Throughput is defined as the number of transactions processed per second (TPS). Let Nt_T represent the total number of transactions that have been committed within a fixed time frame T. The formula for calculating throughput is given by:

$$Throughput = \frac{Nt_T}{T} \quad (6)$$

Higher values indicate a more efficient protocol in handling transactions.

Storage: torage efficiency, denoted as S_{tx} (Storage per transaction), measures storage required for each transaction. It is calculated by

$$S_{tx} = \sum_{i=1}^{Nb_T} \left(\frac{Blocksize \times N_i}{Nt_T} \right) \quad (7)$$

Nb_T refers to the number of blocks that have received a consensus decision within the time-bound T. N_i indicates the number of nodes required to receive the ith block, which is equivalent to the number of voters for that block. Nt_T stands for the number of transactions committed within the time period T.

C. Comparing to Rapidchain

Rapidchain is a well-known one-layer sharding protocol with both L and S being less than 50%. It is a representative protocol that does not dynamically adjust L and S values in our experiments.

In our experiments, we run Lipper with $N = 5000$ nodes. With this setting of N, $Num_1 = N_w = 19, Num_2 = 28, Num_3 = 50, Num_4 = 94, Num_5 = N_c = 293$. We set $T = 500s$ and $T_1 = 60s$, which are chosen based on the assumed network bandwidth. We compute $T_2 \approx 71.24s, T_3 \approx 92.12s, T_4 \approx 217.37s$ accordingly. To make a fair comparison, we also run the experiments of Rapidchain with the $N = 5000$ nodes and the same network settings. With the same network bandwidth, Rapidchain's $\Delta \approx 33.33s$. We randomly fluctuate the runtime adversarial population (P_a) in different epochs as shown in Figure 4 where the experimental outcomes for twenty epochs are also presented. Specifically, $P_a \leq v < \frac{1}{3}$ within each committee shard is maintained during the epochs.

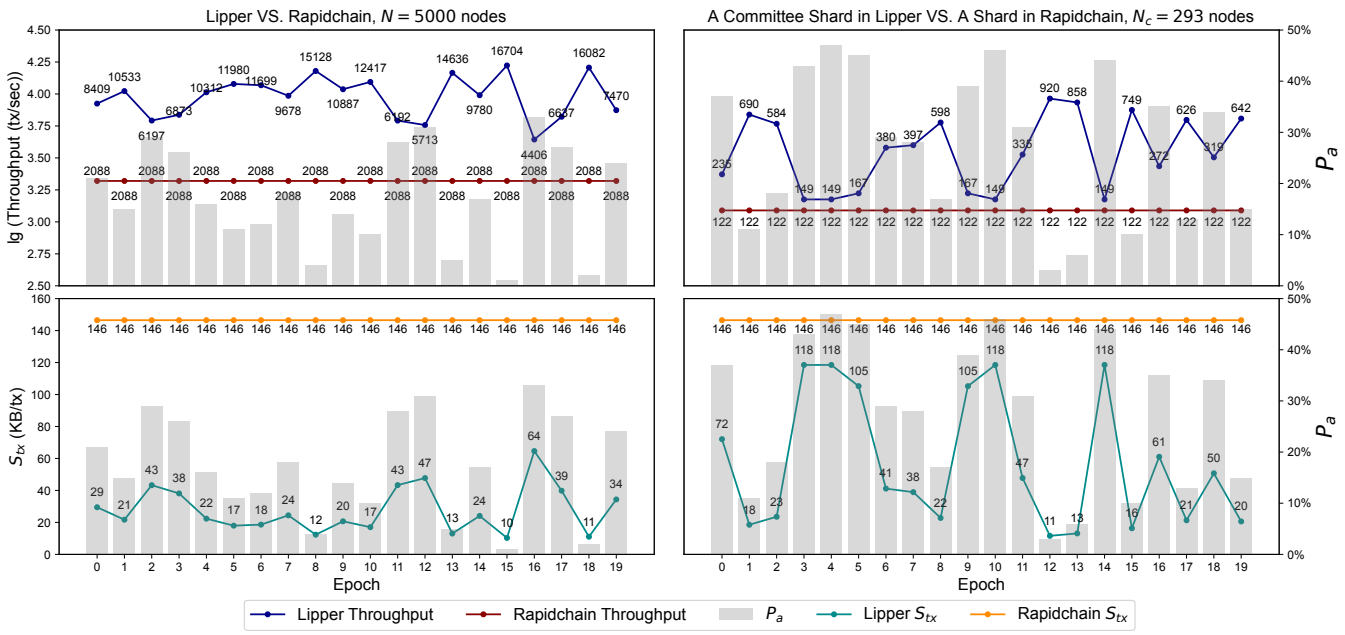

Fig. 4. The experiment result of a system of Lipper vs. Rapidchain. Note that in the whole system, $P_a \leq v < \frac{1}{3}$, and in the CS, the $P_a < \frac{1}{2}$.

Afterward, an additional twenty epochs were executed, where P_a values were generated randomly, satisfying $P_a < \frac{1}{2}$ within each committee shard.

The experimental results indicate that Lipper promptly adjusts its throughput and storage consumption based on the runtime adversarial population. Notably, Lipper demonstrates superior performance compared to Rapidchain even when the runtime adversarial population approaches 50% in a single committee shard. This is attributed to the presence of work shards and helpers that can achieve consensus before $phase_5$ under such conditions. In contrast, Rapidchain always uses shards of the same size as a committee shard in Lipper to reach consensus, which results in low transaction throughput. Furthermore, considering that the overall system maintains $P_a < \frac{1}{3}$, the likelihood of the adversarial population within a single committee shard to approach $\frac{1}{2}$ is low. Consequently, as suggested by Fig. 4, Lipper exhibits significant improvement in transaction throughput and storage consumption compared to Rapidchain.

D. Comparing to Gearbox

As mentioned earlier, Gearbox was evaluated by simulations based on mathematical models. In this section, we compare the simulated performance of Gearbox with the experiment results of Lipper.

Gearbox operates with several liveness gears. It starts a gear with small shards having high S and low L. Whenever a shard loses liveness, it changes to another gear with a larger L. When switching a gear, Gearbox completely dissolves the shard and reassembles a new shard of a larger size from scratch. This process incurs time overhead for synchronization. Originally, Gearbox assumes that the adversarial population is static, but below the upper bound; thus, the gear only adjusts in one way, i.e., from a small to a larger size. Thereby, the attackers would only need to attack once to deteriorate the performance permanently. To address the problem, the authors of Gearbox mentioned in the paper that it can periodically switch back to a smaller gear. Unfortunately, they did not elaborate on this idea. For the simplicity and fairness of the experiment, we assume the shard membership adjustment of Gearbox has no overheads, and Gearbox will attempt to switch back to a smaller gear after each epoch. Such assumptions are significantly in favor of Gearbox and largely overestimate their performance.

Metric Calculations: Considering that Gearbox has overlapping shards, which implies each node may carry workloads of multiple shards simultaneously. Therefore, we have to take this effect into account in the simulation model.

We introduce $Overlaptime$ as the average number of shards that a node is involved in:

$$Overlaptime = \frac{\sum_{i=1}^{Number\ of\ Shards} Shard\ Size_i}{N} \tag{8}$$

This parameter will be used for the calculation of Gearbox's throughput and storage consumption. When calculating the throughput of Gearbox, we have to consider that each shard may have a different size (in different gear). We can calculate the throughput of Gearbox as follows,

$$Thoughput = \frac{\sum_{i=1}^{E_{block}} \frac{Tx\ in\ Block}{Btime_i}}{E_{block} \times Overlaptime} \tag{9}$$

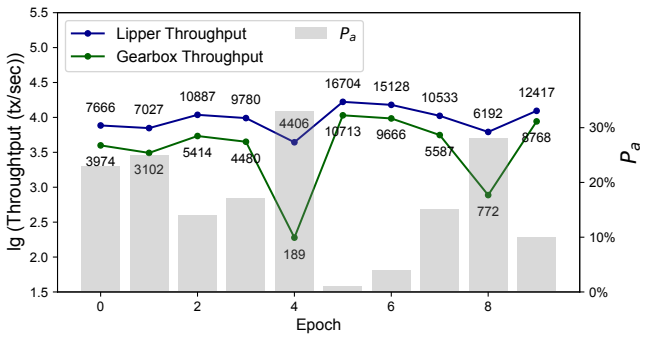

Fig. 5. Comparison of throughput changes for Lipper and Gearbox for dynamic adversarial populations

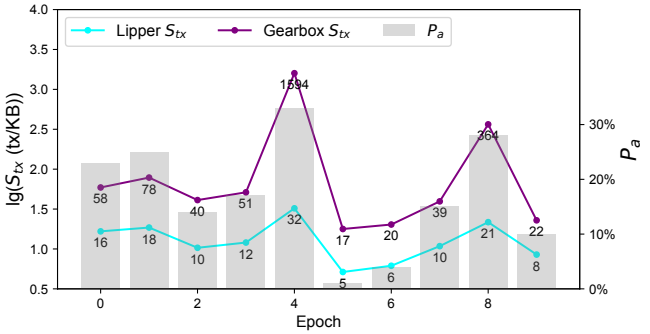

Fig. 6. Comparison of S_{tx} changes for Lipper and Gearbox for dynamic adversarial populations.

where E_{block} represents the block that is generated in an epoch. The time for generating and agreeing on $Block_i$ is $Btime_i$, and $Btime_i$ consists of two parts: the time to switch to a suitable gear and the time to achieve consensus. The first part takes no time if the gear does not need to be changed. This equation is a generalization of the Eqn. 6. It resembles a special case for Gearbox where all shards have the same block generation time and no overlapping.

To calculate the storage consumption of Gearbox, we extend Eqn. 7 by taking into account of shard overlapping:

$$S_{tx} = \sum_{i=1}^{E_{block}} (\frac{Blocksize \times N_i}{E_{tx}}) \times Overlaptime \qquad (10)$$

As one can see, this equation is a generalization of Eqn. 7 for the case with $overlaptime = 1$. We use E_{tx} to denote the number of transactions that reach consensus in an epoch.

Simulation results: Adhering to the recommendations of Gearbox, we employ four gears corresponding to the liveness threshold of 10%, 20%, 25%, and 30%. To align with our work, we add a gear with L as 50%. The specific shard sizes for the five gears are 26, 39, 50, 63, and 293, respectively. Furthermore, the system has $N = 5000$, and $\lambda = 22$. Fig. 5 illustrates the experimental outcomes of throughput of ten epochs, where the P_a values were generated randomly. Specifically, these values adhere to the condition $P_a \leq v < \frac{1}{3}$ across the entirety of the system. Fig. 6 shows the storage consumption in the same ten epochs. The experimental results clearly demonstrate that when P_a is small, Gearbox exhibits performance comparable to Lipper. This is because Gearbox initially creates $\lfloor N/26 \rfloor$ shards with shard size 26 as the lowest gear. This achieves a similar parallelism as Lipper.

However, as P_a approaches $\frac{1}{3}$, a substantial amount of overlap occurs, leading to a substantial decrease in throughput. In the case of $P_a = \frac{1}{3}$, Gearbox requires 1781KB of storage for a single transaction within the entire system, which can be considered highly detrimental. With a high adversarial ratio, it is possible that all $\lfloor N/26 \rfloor$ shards in Gearbox will be adjusted to the highest gear, each shard sized 293. In this scenario, each node, on average, is involved in $(\lfloor N/26 \rfloor \times 293/N)$ shards, or approximately 12 shards. As a result, each node's workload is duplicated approximately 12 times. This overhead may significantly reduce the throughput and incur considerable storage consumption.

Note that the overhead of shard reconstruction in Gearbox is omitted in our experiments. Therefore, in reality, the performance of Gearbox is expected to be lower than the presented results.

VI. Related Work

Several sharding protocols aim to address blockchain scalability. Elastico [13] introduced proof-of-work-based committees per epoch but faces security issues with small committees. OmniLedger [11] improves on Elastico but efficiently tolerates up to $t < N/4$ corruptions. RapidChain [19] tolerates up to $N/3$ corrupt parties with a synchronous committee election. Monoxide [16] proposes a scale-out blockchain with workload division among parallel chains.

Avarikioti et al. [1] presents a framework for sharded ledger protocols, introducing consistency and scalability notions.

Recent efforts [5], [18] address adversarial population adjustment. Xu et al. [18] propose a protocol with two-way shard size adjustments, but it's vulnerable to attacks. Gearbox [5] adjusts shard size based on adversarial percentage, but it can't determine runtime adversary population, leading to potential inefficiencies and overlapping shards, impacting parallelism.

VII. Conclusion

Lipper, a sharding protocol, tackles blockchain scalability challenges uniquely. It dynamically adjusts throughput based on adversary nodes in real-time, ensuring high performance. The two-layer design of committee and work shards enhances transactions per second. By adapting liveness and safety thresholds, Lipper surpasses protocols with fixed assumptions about adversaries. It eliminates time-consuming shard adjustments and node overlap, leading to significant throughput improvements, especially in extreme scenarios. In summary, Lipper is a groundbreaking sharding protocol offering substantial scalability advancements to the blockchain ecosystem.

REFERENCES

[1] G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer, "Divide and scale: Formalization of distributed ledger sharding protocols," *arXiv preprint arXiv:1910.10434*, 2019.

[2] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

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

[4] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2518–2534.

[5] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 683–696.

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

[7] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, "Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2020.

[8] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized rumor spreading," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000, pp. 565–574.

[9] K. Khacef, S. Benbernou, M. Ouziri, and M. Younas, "Trade-off between security and scalability in blockchain design: A dynamic sharding approach," in *The International Conference on Deep Learning, Big Data and Blockchain (Deep-BDB 2021)*. Springer, 2022, pp. 77–90.

[10] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*. Springer, 2017, pp. 357–388.

[11] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[12] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*. Springer, 2015, pp. 528–547.

[13] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 17–30.

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

[15] L. Ren, K. Nayak, I. Abraham, and S. Devadas, "Practical synchronous byzantine consensus," *CoRR*, vol. abs/1704.02397, 2017. [Online]. Available: http://arxiv.org/abs/1704.02397

[16] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, 2019, pp. 95–112.

[17] Y. Xu and Y. Huang, "An n/2 byzantine node tolerate blockchain sharding approach," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 349–352.

[18] Y. Xu, Y. Huang, J. Shao, and G. Theodorakopoulos, "A flexible n/2 adversary node resistant and halting recoverable blockchain sharding protocol," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 19, p. e5773, 2020.

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

[20] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1847–1852.