A Two-Layer Blockchain Sharding Protocol Leveraging Safety and Liveness for Enhanced Performance

Yibin Xu
Department of Computer Science
University of Copenhagen
yx@di.ku.dk

Jingyi Zheng
Department of Computer Science
University of Copenhagen
jrb385@alumni.ku.dk

Boris Düdder
Department of Computer Science
University of Copenhagen
boris.d@di.ku.dk

Tijs Slaats
Department of Computer Science
University of Copenhagen
slaats@di.ku.dk

Yongluan Zhou
Department of Computer Science
University of Copenhagen
zhou@di.ku.dk

***Abstract*—Sharding is a critical technique that enhances the scalability of blockchain technology. However, existing protocols often assume the adversarial nodes in a general term without considering the different types of attacks, which limits transaction throughput in runtime because the attack on liveness could be mitigated. This paper introduces Reticulum, a novel sharding protocol that overcomes this limitation and achieves enhanced scalability in a blockchain network.**

Reticulum employs a two-phase design that dynamically adjusts the transaction throughput based on the runtime adversarial attacking either or both liveness and safety. It consists of "control" and "process" shards in two layers corresponding to the two phases. Process shards are subsets of control shards, with each process shard expected to contain at least one honest node with high confidence. Conversely, control shards are expected to have a majority of honest nodes with high confidence. In the first phase, transactions are written to blocks, which are then voted on by nodes in the corresponding process shards. Blocks that receive unanimous acceptance verdicts are accepted. In the second phase, blocks that do not obtain unanimous verdicts are collected and voted on by the control shards. A block is accepted if the majority of nodes vote to accept it. The opponents and silent voters of the first phase will be eliminated. In summary, Reticulum leverages unanimous voting in the first phase to involve fewer nodes for accepting/rejecting a block, allowing more parallel process shards. The control shard finalizes the decision made in the first phase and serves as a lifeline to resolve disputes when they surface.

Experiments demonstrate that the unique design of Reticulum empowers high transaction throughput and robustness in the face of different types of attacks in the network, making it superior to existing sharding protocols for blockchain networks.

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

I. Introduction

Blockchain sharding [17], [27], [15], [5], [29], [7], [8], [16], [21], [9], [13], [14], [24], [25] is a method that aims to improve the scalability of a vote-based blockchain system by randomly dividing the network into smaller divisions, called shards. The idea is to increase parallelism and reduce the overhead in the consensus process of each shard, thereby increasing efficiency. However, there is a trade-off between parallelism and security in sharding. While making smaller and more shards can increase parallelism, they are more vulnerable to corruption.

The security of a shard has two key properties: liveness and safety. Liveness concerns the shard's capability to ultimately achieve a consensus on the sequence of output messages, whereas safety pertains to the accuracy and exclusivity of that consensus. For example, if we have $M = 10$ nodes in a synchronous communication shard, a decision is made only if at least seven nodes vote in favor. Then, $S = 6/M$ is the maximum ratio of adversarial population that can be tolerated. Otherwise, an adversarial decision may be reached. However, if at least a $1 - S$ ratio of nodes in the shard always disagree or keep silent with any proposals, the shard can never reach any decisions, and the liveness is compromised. The safety threshold (S) and liveness threshold (L) represent the maximum ratio of adversarial participants in a shard under which safety and liveness are guaranteed.

However, existing sharding solutions[27] guarantee security under worst-case adversarial conditions ($L = S < 50\%$) in the synchronous network and assume that the adversary has equal interests in attacking both safety and

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

liveness simultaneously. It limits the number of parallel shards and the performance gains of sharding when the nodes acting adversarial at runtime is lower than the worst-case.

Recent proposals [7], [22], [23] aim to increase the number of parallel shards by increasing S and decreasing L of each shard. This is based on the fact that $S < 1 - L$ (synchronous model) or $S < 1 - 2L$ (partially synchronous model) holds because given L percent of nodes that attack liveness, a block will only be accepted if at least $1 - L$ or $1 - 2L$ percent of nodes respectively have voted in favor of it. Therefore, the system cannot tolerate more than $1 - L$ or $1 - 2L$ percent of nodes, respectively, that attack safety. Based on this theory, one can generate smaller shards with higher S (and lower L) to tolerate the same number of adversaries globally. Note that $S \geq L$ is maintained in all cases.

Fig. 1 illustrates changing L and S by adapting the shard size proposed in [7], [23]. This design allows using small shards when the adversarial population ratio that intends to attack liveness is low, so more shards run in parallel. When this ratio is higher at runtime, the shards can be respawned with a larger shard size, resulting in an increased L, and a decreased S [7], [23]. The existing shard resizing approaches either reform the shards from scratch, changing the shard memberships of all the nodes [23], or perform local shard adjustments, resulting in overlapping shards [7] as shown in Fig. 1.

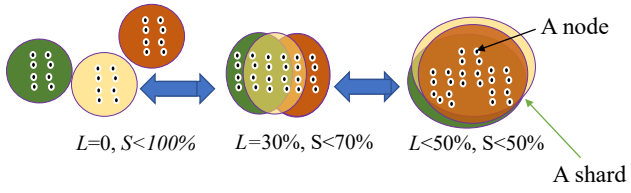

Fig. 1. The typical structure of a sharding approach leveraging liveness and safety thresholds. The size of shards increases with the liveness threshold L increase. There are three shards in the system. The increased size either reduces the number of shards or causes shards to overlap.

Despite the potential performance gains associated with leveraging liveness and safety, several key limitations must be addressed. *First*, the approaches of adapting shard sizes and memberships, as mentioned earlier, can result in frequent and costly shard respawning. Additionally, the presence of overlapping shards when the adversarial population fluctuates leads to additional costs. *Secondly*, our analysis in Sec. V-A uncovers security vulnerabilities present in all existing designs. In particular, synchronous shards with $S \geq \frac{1}{2}$ or partially synchronous shards with $S \geq \frac{1}{3}$ are unable to achieve consensus without the possibility of equivocation. These vulnerabilities undermine the effectiveness of the approaches. *Thirdly*, all approaches so far have no design to mitigate the liveness attacks. Such attacks can be detected and evidenced in runtime, allowing for appropriate punitive actions when liveness is restored while ensuring safety is maintained at all times. For instance, if there are methods to expel a node when this node disagrees with correct proposals or frequently remains silent, such adversarial nodes cannot again attack liveness in the future after being expelled.

A. Contribution

The contributions of this paper are as follows:
(1) We propose Reticulum, the first protocol leveraging liveness and safety but does not suffer from security breaches. Reticulum can inhibit adversarial behaviours. These are achieved without needing to respawn new shards in runtime or to overlap shard memberships, which can bring unnecessary overhead.
(2) We comprehensively analyze Reticulum's performance with a comparison with state-of-the-art approaches. We evaluate Reticulum's performance empirically by both simulation and experiments. We compare Reticulum with two state-of-the-art approaches: Gearbox[7] and Rapidchain[27]. Our analysis shows that Reticulum significantly outperforms both approaches regarding transaction throughput and storage requirements.
(3) We have implemented and open-sourced a prototype of Reticulum. The lack of available implementations of blockchain sharding protocols has created a significant challenge for researchers attempting to compare different protocols through experimentation. Our open-source prototype [11] is an important step toward addressing this issue and will provide researchers with a valuable resource for studying blockchain sharding.

II. Overview

To address the shortcomings of existing approaches, we propose Reticulum, a blockchain sharding approach designed for a synchronous environment. For simplification, Reticulum uses a static sharding scheme, i.e., no new members are added in runtime and we do not respawn any shards. But in reality, the same as Rapidchain[27] and Omniledger[15], it may allow a system reboot with fresh nodes added and some nodes removed every several days using the same design. It may also use the same design as Rapidchain to swap nodes of different shards after several epochs to avoid adversaries (slowly) enlarging the corruption population in a shard (assuming they are capable of doing so). Reticulum adopts two layers of ledgers to provide resilience. Fig. 2 illustrates its structure.

Reticulum employs a two-phase-voting mechanism in which a blockchain epoch is divided into two phases. Each node is assigned to one and only one process shard, which consists of the first layer of ledgers. Each process shard is assigned to and governed by one and only one control shard, which consists of the second layer of ledgers. The shard memberships do not change at runtime and they do not overlap. Moreover, all nodes are connected to a public

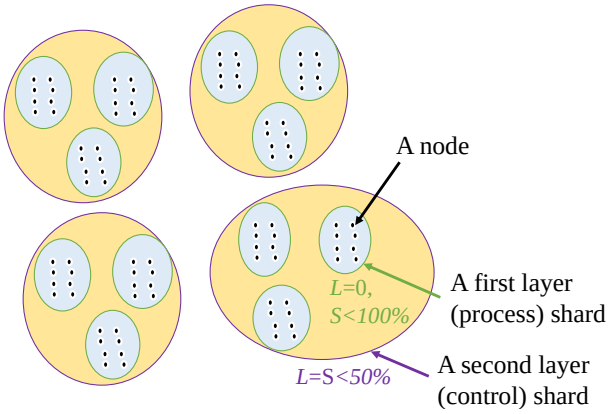

Fig. 2. The structure overview of Reticulum protocol. Every node is, at the same time, in only one process shard ($L = 0, S < 100\%$) and one control shard ($(L = S) < 50\%$) that governs this process shard.

communication chain, which is used for communicating metadata.

In the first phase, the members of a process shard generate and vote for the acceptance of a process block containing transactions associated with the data stored. All process shards have $S < 100\%$, which means that a block will only be accepted with a unanimous vote. On the other hand, the process shards have $L = 0\%$, which means that one adversary would be sufficient to hinder the progress.

In order to prevent progress hindrance caused by an acceptable block failing to receive a unanimous verdict in a process shard, a control shard is used in the second phase to reach a majority verdict on the block that did not pass in the first phase. Each control shard has a threshold of $L, S < 50\%$, which ensures that it provides comparable worst-case security to other state-of-the-art blockchain sharding protocols. The votes in a *process* shard are broadcast to all nodes in the corresponding *control* shard (sized N_c) using a Byzantine broadcast protocol called $(\Delta + \delta)$-BB [2] with $f \leq \lfloor (N_c - 1)/2 \rfloor$, this enables them to align that if a process block needs to be handled in the control shard.

We design that when a process block cannot be accepted in a process shard, but later be accepted by the control shard, all nodes in the process shard which voted for rejection will be expelled from the system. If a node remained silent in voting, it is marked as "violated node" in the current voting. A node can only be marked as "violated node" once in every τ votings for the process blocks, where τ is a pre-defined liveness threshold. If node i is marked twice, it is also expelled from the system.

It is important to note that the cost of the second voting phase will depend on the number of failed process blocks in the first phase. As the adversarial population decreases (or increases), fewer (or more) process blocks will likely fail the first voting phase, which will result in a lower (or higher) cost for the second phase and higher (or lower) transaction throughput. When no adversaries are attacking liveness, Reticulum will achieve its highest transaction throughput because it has no failed process block to vote for in the second phase. In this way, Reticulum is resilient to the dynamic decrease (or increase) of the adversarial population with higher (or lower) transaction throughput.

III. Problem domain

This section defines the network and threat models for our sharding protocol, the overview of Reticulum, and the objectives of Reticulum.

A. System model

Network model: standard synchronous network model. Similar to Rapidchain [27], our network model consists of a peer-to-peer network with N nodes that use a Sybil-resistant identity generation mechanism to establish their identities (i.e., public/private keys). This mechanism requires nodes to attach a Proof-of-Stake (PoS) of a threshold weight that all other honest nodes can verify. We assume that all messages sent in the network are authenticated with the sender's private key. These messages are propagated through a synchronous gossip protocol [12], which guarantees that the message delays are at most a known upper bound Δ. Note that this protocol does not necessarily preserve the order of the messages.

It is worth noting that Gearbox [7] claims to utilize a mixed network model based on L and S, enabling the execution of a partially synchronous model until $L = 25\%$ and $S = 49\%$. However, as will be demonstrated in Sec. V-A, in practice, the decision to accept or reject blocks of a shard in Gearbox can only be safely reached after finalization. Therefore, it is the confirmation of the heartbeat within a block of the "control chain" that signifies the completion of the BFT process of each shard. Since the blocks of the "control chain" also incorporate heartbeats from other shards, some of which may operate with $L > 25\%$ at the time, in many cases, the block is generated only after a known time-bound, which implies a synchronous network model for Gearbox.

Threat model: adaptive but upper-bounded adversaries with τ liveness guarantee. In our threat model, we consider the possibility of nodes disconnecting from the network due to internal failures or network jitter, as well as a probabilistic polynomial-time Byzantine adversary that can corrupt up to $f \leq \lfloor (N - 1)/3 \rfloor$ nodes at any given time. These adversarial nodes may collude with each other and deviate from the protocol in arbitrary ways, such as by sending invalid or inconsistent messages or remaining silent. Note that a non-sharded synchronous communication environment can tolerate $f \leq \lfloor (N-1)/2 \rfloor$. However, for a sharded network with shard size M, it is impossible to withstand $f \leq \lfloor (M - 1)/2 \rfloor$ adversary (the mere majority honesty), while allowing $f \leq \lfloor (N - 1)/2 \rfloor$ globally. This is because the failure probability (more than the threshold adversary nodes being assigned to the shard)

cannot be decreased to a trivial number with a random shard assignment for nodes.

Rapidchain [27] and Gearbox [7] also require that, at any given moment, at least 2/3 of the computational resources belong to uncorrupted online participants. Since the runtime adversarial population is unknown to the uncorrupted participants, achieving *deterministic* consensus requires constant liveness for the same set of $2\lceil\frac{N-1}{3}\rceil+1$ honest nodes, as shown in Theorem 1.

Theorem 1. *For guaranteeing* ***constant*** *liveness and safety in the overall system, a blockchain sharding protocol with a safety threshold S and a liveness threshold L (both less than 50%) necessitates the presence of at least $\left\lceil\frac{2(N-1)}{3}\right\rceil+1$ uncorrupted participants who remain active in the system* ***at all times.***

Proof. Let us assume that the adversary has control over $f=\lfloor\frac{N-1}{3}\rfloor$ nodes across the entire network, and these nodes persistently act in an adversarial manner. Under this scenario, all other nodes must maintain their integrity to satisfy the liveness guarantees and must not leave the system. This is essential to avoid the following situation: Only one uncorrupted node among the $N-\lfloor\frac{N-1}{3}\rfloor$ uncorrupted participants leaves the sharding protocol during a specific epoch, and it happens that this node belongs to a shard containing $f=\lfloor\frac{M-1}{2}\rfloor$ nodes. Then the shard becomes unable to deterministic accept an uncorrupted block for this specific epoch. Consequently, both liveness and safety are lost. The loss of liveness and safety in a single shard compromises the constant liveness and safety guarantee of the overall system. □

We use the τ liveness guarantee for the adversarial nodes, which is defined in Definition 1. If a node violates τ liveness, it will be expelled from the system.

Definition 1. *τ liveness guarantee refers to a blockchain sharding system that mandates that at least $\left\lceil\frac{2(N-1)}{3}\right\rceil+1$ uncorrupted participants constantly act uncorrupted and remain alive, but allowing any other participant to not participate in the system once in every τ epochs.*

B. System overview

Reticulum has a determined number of nodes N. Each node n_i of these N nodes is a member of the public communication chain (PC) and is a member of exactly one process shard and one control shard. We elaborate on our designs below:

Public communication chain. The public communication chain (PC) coordinates the sharding protocol and stores metadata about the shards. It is a totally-ordered broadcast with persistence and a timestamp guarantee, and all nodes in the system execute it. The PC stores only metadata, which means that its size is independent of the contents of the shards.

Shards. Shards are modeled as ledger functionalities that are parameterized by the size of the shard and the type (process/control) of the shard. There are Λ control shards, each containing at least $N_c=\lfloor N/\Lambda\rfloor$ nodes, and β process shards, each containing at least $N_p=\lfloor N/\beta\rfloor$ nodes. Each process shard is governed by only one control shard. Each control shard supervises at least $\lfloor N_c/N_p\rfloor$ process shard. Each process shard governs a non-overlapping set of keys, and the related transactions should be sent to the corresponding process shards. Cross-shard transactions that interact with multiple shards are used if different process shards govern the keys involved in the transactions. Each process or control shard runs an ordinary synchronous BFT protocol, where a leader node is randomly selected to propose a block followed by one round of voting. A process block is accepted in a process shard if all nodes vote to accept it. A control block is accepted in a control shard if the majority of nodes vote to accept it. In normal cases, the control shard does not synchronize the blocks of the process shards. *However, they synchronize the votes for the blocks of each process shard under their governance.* If a block in a process shard does not obtain unanimous votes within a time-bound (T_1), it is synced by all nodes in the corresponding control shard in another time-bound(T_2). The control shard votes on them to finalize the verdicts.

State-block-state structure. The state of the process shard is updated with each process block, and a new state is formed in each epoch. Rather than linking blocks to preceding blocks, they are linked to previous states. If a block does not pass unanimous voting, only the latest state and the rejected block need to be synced by nodes in the control shard, instead of the entire blockchain of the process shard. This greatly reduces the amount of data that needs to be synced, improving the system efficiency.

Bootstrapping. The system is initialized with a predetermined list of N participating nodes, which are ranked using a global random number generated by a distributed random number generator [6]. Nodes are assigned to different shards after bootstrapping according to rank.

Simple Cross-shard communication. Cross-shard communication, which involves the exchange of information and data between different shards in a sharding system, has been extensively researched in the literature [15], [28], [3], [20], [10], [19]. However, implementing a reliable cross-shard communication protocol in a sharding system that potentially have liveness issues be challenging. Gearbox, for instance, customized the existing protocol Atomix [15] to finalize messages on the shards to address this issue, which further complicated the algorithm. In contrast, Reticulum eliminates the need for specialized cross-shard protocols altogether. Since we do not adjust shard membership, signatures contained in cross-shard transactions can be trusted without considering if they are signed by the correct shard members at the time. As a result, we can use ordinary cross-shard protocols for cross-control-shard transactions without worrying about the complications associated with deadlocked shards.

C. Objectives

Our protocol involves processing a set of transactions submitted to our protocol by external users. Similar to those in other blockchain systems, these transactions consist of inputs and outputs referencing other transactions and are accompanied by a signature for validity. The set of transactions is divided into $\beta = \lfloor N/N_p \rfloor$ blocks proposed by disjoint process shards each sized N_p, with $y_{\{i,j\}}$ representing the j-th transaction in the block of i-th process shard. There are $\Lambda = \lfloor N/N_c \rfloor$ control shards, each sized N_c. The i-th process block is governed by $\lfloor i/N_c \rfloor$-th control shard. All nodes within the same process shard have access to an external function $g1$ that determines the validity of a transaction, outputting 0 or 1 accordingly. All nodes in the same control shard have access to an external function $g2$ that determines the validity of a process block that did not pass the unanimous voting in the first phase, outputting 0 or 1 accordingly. $Block(i)$ is the block of process shard i of the current epoch. If $Block(i)$ fails to obtain unanimous acceptance in the first phase and $g2(Block(i))) = 1$, the leader node of $\lfloor i/N_c \rfloor$-th control shard in the current epoch should embed $Block(i)$ to the control block of current epoch and mark it as "accepted". $z_{\{}p, q\}$ represents the q-th process block embedded in the p-th control block that is marked as "accepted". The protocol Π aims to output a set (Y, Z) containing β disjoint process blocks $Y_i = \{y_{\{i,j\}}\}$ where $j \in \{1 \dots |Y_i|\}$ and Λ control blocks $Z_p = \{z_{\{p,q\}}\}$ where $q \in \{1 \dots |Z_p|\}$, satisfying the following conditions:

- Agreement: For any $i \in \{1 \dots \beta\}$, at least one honest node agree on Y_i with a probability of greater than $1 - 2^{-\sigma}$, where σ is the security parameter. In the meantime, $\Omega(\log N)$ honest nodes agree on Z_p with a probability of greater than $1 - 2^{-\sigma}$.
- Validity: For each $i \in \{1 \dots \beta\}$ and $j \in \{1 \dots |Y_i|\}$, $g1(y_{\{i,j\}}) = 1$. For each $p \in \{1 \dots \Lambda\}$ and $q \in \{1 \dots |Z_p|\}$, $g2(y_{\{p,q\}}) = 1$.
- Termination: Each shard $i \in \{1 \dots \beta\}$ decide (approved unanimously or not) on Y_i within T_1. Each shard in $p \in \{1 \dots \Lambda\}$ decide on Z_p within T_2.

The design objectives of Reticulum are to achieve the following while satisfying the above conditions.

- Scalability: The number of control shards Λ and process shards β in Reticulum can grow with the increase of the total number of nodes in the network, denoted by N, without affecting the aforementioned conditions. This design ensures that Reticulum can handle a growing network and maintain high throughput. Furthermore, as the number of honest nodes in the network increases, Reticulum's throughput also increases.
- Efficiency: Reticulum is designed to minimize per-node communication, computation, and storage complexity. The per-node communication and computation complexity is $O(N)$, where N is the total number of nodes in the network. The per-node storage complexity is $O(OT/k)$, where OT is the total number of transactions, and k is the average number of process shards

In summary, our objective is to design a consensus protocol that achieves agreement among a sufficient number of honest nodes, ensuring the validity of the agreed-upon transactions, scales with the size of the network and the runtime adversary population, and has efficient communication, computation, and storage cost.

IV. Reticulum protocol

This section will provide a detailed description of the Reticulum protocol. Reticulum comprises three main components: bootstrapping, two-layer-shard consensus, and cross-shard transactions.

In Reticulum, each node participates in the public communication chain (PC) and exactly one process shard, along with its corresponding control shard. This means that each node maintains two additional chains besides PC: the *process-shard-chain* specific to the process shard and the *control-shard-chain* specific to the control shard. A block proposed for the process-shard-chain is called a *process block*, while that for the control-shard-chain is called a *control block*. A process block includes a collection of transactions, and the control block includes the hashes of process blocks and their vote signatures.

A. Bootstrapping

The Bootstrapping phase is used to assign nodes to shards randomly and unbiasedly. We first share a predetermined list of N nodes with all participants. Then, a random sequence C, where $|C| = N$, is generated collectively utilizing a random beacon [6]. The random beacon must be accessible to all participants, verifiable by all parties, and should not be controlled by any entity.

We use a $getShardIndex$ function to determine a node's shard membership. This function takes a node $Node_j$ as input, retrieves the index C_{index} of $Node_j$ in the sequence C, and then calculates the indices $index_1$ and $index_2$ for the node's process and control shards, respectively. The output is a tuple $(index_1, index_2)$. Note that bootstrapping only runs at the beginning of the protocol or when the protocol re-starts with new node membership. The pseudocode for $getShardIndex$ is given in Algorithm 1.

Lemma 1. *The bootstrapping phase guarantees the safety property of shard membership assignment, ensuring that no node is assigned to multiple shards.*

Proof. Please see the full proof in appendix B1. □

B. The first phase

After nodes are randomly assigned to the process shard and the control shard during bootstrapping, Reticulum enters the consensus stage. An epoch for the consensus stage consists of two phases, with the first phase to be completed within a time-bound denoted as T_1, and the second phase to be completed within a time-bound denoted as T_2. While

Algorithm 1 getShardIndex

N_c and N_p are the numbers of nodes inside a control and process shard, respectively, $N_c | N_p$. A sequence C is generated using a distributed random beacon from $\{Node_1, Node_2, \ldots, Node_N\}$. $|C| = N$. When $N \nmid N_c$, the number of nodes inside the last control shard and the last process shard may exceed N_c and N_p.
function GETSHARDINDEX($Node_j$)
    $C_{index} \leftarrow$ The index of $Node_j$ in the sequence C starting from 0
    $index_1, index_2 \leftarrow \lfloor C_{index}/N_p \rfloor, \lfloor index_1/(N_c/N_p) \rfloor$
    if $C_{index} \geq \lfloor N/N_c \rfloor \times N_c$ **then**
        $index_2 \leftarrow index_2 - 1$ ▷ Add nodes to the last control shard but not a new control shard
        if $N \nmid N_p$ **then**
            $index_1 \leftarrow \min(\lfloor (C_{index}/N_p) \rfloor, \lfloor (N/N_p) \rfloor - 1)$ ▷ Add nodes to the last process shard but not a new process shard
    return $(index_1, index_2)$ ▷ $Node_j$ belongs to ps_{index_1} and cs_{index_2}

Algorithm 2 First phase of Two-layer consensus

procedure FIRSTPHASE($node$) ▷ $node$ run this procedure
    $votes \leftarrow$ empty set of votes
    if this node is the leader of its process shard **then**
        $block \leftarrow$ GENERATEBLOCK($node$)
        BROADCASTBLOCK($block, node.ps$) ▷ broadcast the block to all nodes in the same process shard
    if this node is not the leader of its process shard **then**
        $block \leftarrow$ RECEIVEBLOCK()
    $vote \leftarrow$ GENERATEVOTE($node, block$) ▷ verify and vote for the block
    BROADCASTVOTE($vote, node.cs$) ▷ broadcast the vote to all nodes in the same control shard using a BB protocol
    while in T_1 and CONSENSUS(votes, node.ps)=FAILED **do** ▷ wait for votes
        $votes \leftarrow votes \cup$ RECEIVEVOTE() ▷ add the vote to the set of votes
    $result \leftarrow$ CONSENSUS($votes, node.ps$)
    return $result$
function BROADCASTBLOCK($block, ps$)
    for $n \in ps$ **do**
        send $block$ to node n
function BROADCASTVOTE($vote, cs$)
    for $n \in cs$ **do**
        send $vote$ to node n
function CONSENSUS($votes, ps$)
    $count_1 \leftarrow 0$
    for $v \in votes$ **do**
        if $v.value = approve$ and $v.ps = ps$ **then**
            $count_1 \leftarrow count_1 + 1$
    if Unanimously approved **then**
        return $ACCEPTED$
    else
        return $FAILED$

T_1 is set as a constant number, T_2 depends on the number of process shards and their verdicts in the first phase. We will discuss this in detail in Sec. IV-C.

The first phase is used for the process shards to reach a consensus on the process blocks. After bootstrapping, there are at least N_p nodes in any process shard ps_i, $i \in [0, \beta]$. At the beginning of the first phase, each ps_i tries to generate and reach consensus on a block using a standard binary vote-based synchronous consensus protocol [18] within the time-bound T_1. Each process shard requires a unanimous vote to decide on a block. In other words, as long as there is one adversarial node in ps_i, it cannot decide on a block and cannot evolve into a new state. Therefore the process blocks have $L = 0$. Apparently, the consensus protocol in the first phase only yields two possible outcomes:

- **A block is unanimously accepted:** This implies that there is a probability greater than $1 - 2^{-\sigma}$ that no nodes act adversarial in this process shard.
- **A block has not received unanimous voting:** Because of the synchronous communication assumption, all honest nodes vote and receive each other's vote within T_1. Because we are using a binary vote-based consensus protocol, the honest nodes will decide on the same verdict (either accept or reject) of a block. When a block has not received unanimous voting, it implies that there are adversarial nodes in the shard.

Note that the votes should be broadcasted to all nodes within the same control shard governing this process shard using the Byzantine broadcast protocol $(\Delta + \delta)$-BB [2] with $f \leq \lfloor (N_c - 1)/2 \rfloor$ to allow nodes in the other process shards to know the voting output and the deterministic set of votes. The pseudo-code for the first phase of the two-layer consensus is given in Alg. 2.

Theorem 2. *In Reticulum, nodes can reach a consensus for whether a node is identified as the adversarial node when the votes of the first phase are broadcasted using Byzantine broadcast protocol.*

Proof. Blockchain consensus ensures the validation of the consensus for the sequence of transactions, rather than the correctness of individual transactions. The correctness of transactions can be verified by referring to information in previous blocks of the blockchain. Therefore, a node will only vote to reject blocks if they contain violations such as double-spending or over-spending transactions that contradict previous verdicts. The control blocks also include decisions for expelling nodes that did not participate in the voting for the process shards. To determine this in consensus, it is necessary for nodes to reach a consensus on whether a node voted for the process block within the time-bound. To support this design, when nodes vote in the first phase, the votes are broadcast to all nodes in the control shard (sized N_c) using a Byzantine broadcast protocol called $(\Delta + \delta)$-BB [2] with $f \leq \lfloor (N_c - 1)/2 \rfloor$. It guarantees that, if a vote V is broadcast to all nodes in a control shard of size N_c, all honest nodes in this control shard will terminate the broadcast protocol with the same value V. Hence, they can reach a consensus on if a node within the process shard voted or not. Consequently, when an adversarial node proposes/votes for a process block containing incorrect messages or its liveness violates τ liveness guarantees, it can be identified as an adversarial node in consensus. □

As shown in Theorem 2, Reticulum can safely identify nodes as adversarial in consensus. Reticulum will

confiscate the Proof-of-Stake (PoS) of adversarial nodes and expel them from the system. We assume that most adversaries act rationally and do not wish to be marked as adversarial. However, we demonstrate in Sec. IV-C that Reticulum can also tolerate attacks when the adversary is willing to risk confiscation of their PoS.

C. The second phase

The second phase serves as a "safety net" assisting a final verdict for the blocks not passing the unanimous voting. The nodes in the process shards that fail to output unanimously approved blocks send the last state of the process shard and the failed block to all the other nodes in the same control shard.

Similar to other vote-based consensus protocols, a leader node is selected to propose the control block in the second phase. The control block contains information about the process blocks of each process shard under its control. This information includes: (1) a boolean parameter indicating whether the process block passed unanimous voting within T_1, (2) if the block passed unanimous voting, the signatures from the voters are attached as proof, and (3) if the block did not pass unanimous voting, the hash of the block and a decision to accept or reject the block are attached. This decision is made by the creator of the control block, i.e., the current leader. Nodes can request the full content of the process blocks using the hash. Nodes in the control shard will verify the blocks received from the failed process shards and vote for the control block if they agree with the decision made by the current leader.

The completion of the second phase must be accomplished within T_2. Unlike T_1, the value of T_2 for a control shard is dynamically adjusted based on the count of succeeded process shards within that control shard, denoted as N_s. The calculation for T_2 can be expressed using the following equation:

$$T_2 = \lambda \cdot (\lfloor N_c/N_p \rfloor - N_s + 1) \tag{1}$$

where λ represents a pre-defined constant value and N_p is the size of process shards. $T2$ is set to ensure that the network bandwidth requirement for nodes in the second phase is similar to that in the first phase.

After the aforementioned procedures, check if any node i is considered the adversarial node. The PoS of the node i is confiscated and the node i is expelled from the system. Corollary 1 shows that it is safe to operate with the remaining nodes.

Note that if the majority of nodes in the control shard reject the control block, a new proposed block by a new leader will be voted within the same time-bound T_2. A new epoch is initiated only when a control block is accepted.

When a process block of epoch $X+1$ is rejected in an accepted control block, the corresponding process shard will not undergo any new state evolution in epoch $X+1$. Therefore, the state of that process shard in epoch $X+1$ will be the same as in epoch X. Fig. 3 provides a visual illustration of the process and control shards in Reticulum. The pseudo-code for the second phase is given in Alg. 3.

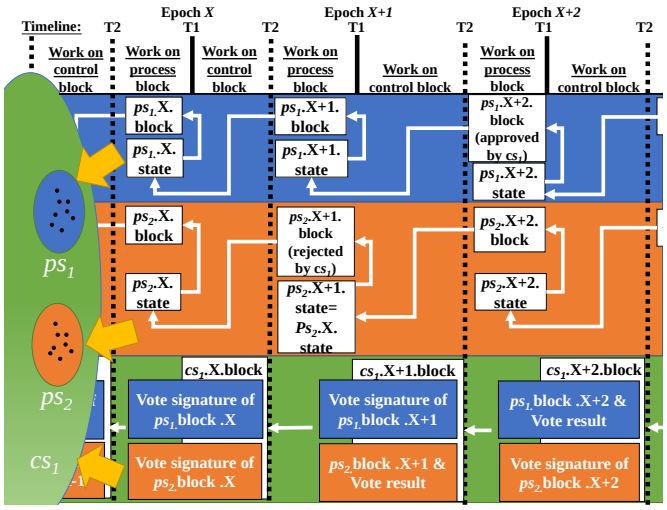

Fig. 3. An example of the two-layer consensus. In this example, the control shard cs_1 contains two process shards, ps_1 and ps_2. We denote the state and block of ps_i for Epoch X by "ps_i.X.state" and "ps_i.X.block" respectively. A new state of ps_i is derived by applying the transactions in ps_i.X.block over ps_i.X.state. In the depicted scenario, ps_1 and ps_2 generated unanimously approved blocks in Epoch X, but ps_2 failed to agree on the process block of Epoch $X+1$ unanimously. Therefore, ps_2.(X+1).block is attached to cs_1.(X+1).block, which denotes the block of cs_1 for epoch $X+1$. Nodes in cs_1 vote on $ps_2.X+1.block$. In this case, cs_1 rejects this block, so $ps_2.X+1.state$ remains the same as $ps_2.X.state$. Note that $T2$ in Epochs $X+1$ and $X+2$ are longer than that in Epoch X because some process blocks did not pass unanimous voting. $T2$ in Epoch $X+1$ and $X+2$ are longer than that in Epoch X because there are process blocks not passing the unanimous voting.

Corollary 1. *Even if the adversaries have a node in each process shard, the system can resist attacks on liveness while maintaining safety.*

Proof. A node is expelled from the system and its PoS be confiscated if it proposed a process block containing faulty information (when it is a leader node), or it voted for a process block containing faulty information or it did not vote at least $\tau - 1$ times in every τ rounds of voting. Consider two types of adversarial behaviors:

1) **Adversaries aiming to retain PoS:** To avoid losing their PoS, adversaries can only remain silent once in every τ round of first-phase voting. By setting τ appropriately, the number of process blocks progressing to the second phase can be restricted. This restriction inhibits the ability of adversaries to influence the consensus outcome and preserves the integrity of the system.

2)**Adversaries wishing to halt system progress:** Adversaries may attempt to halt the system progress by leaving the system permanently, even if their PoS is confiscated by the system. The loss of a relatively small β number of PoS (one adversarial node per process shard), is outweighed by the potential gain of bringing the sys-

Algorithm 3 Second phase of Two-layer consensus

procedure SECONDPHASE(*node*) ▷ *node* run this procedure
  if *node* ∈ $failedprocessshard$ **then**
    send last state and the process block to nodes in the same control shard
  if *node* is the leader node in control shard **then**
    for $B \in$ process shards **do**
      VerifyProcessBlock(B)
    create and broadcast the control block accordingly
  vote for the control block
  if control block is accepted by majority within T_2 **then**
    for node $i \in$ control shard **do**
      confiscated node i's PoS and expel node i if node i is considered the adversarial node.
    initiate new epoch
  else
    repeat second phase with same T_2
  if process block of epoch $X+1$ is rejected in control block **then**
    state for process shard in epoch $X+1$ is set to be identical to state in epoch X
function VERIFYPROCESSBLOCK(B)
  if B passed unanimous voting within T_1 **then**
    attach signatures of voters to B to the control block
    attach boolean parameter indicating B passed unanimous voting to the control block
  else
    attach boolean parameter indicating B failed unanimous voting to the control block
    attach decision to accept or reject B to the control block
Calculate T_2:
λ - constant value
N_s is the count of succeeded process shards in control shard
$T_2 \leftarrow \lambda \cdot (\lfloor N_c/N_p \rfloor - N_s + 1)$

tem to a complete stop. Although such attacks could be detrimental, the system is designed to expel these nodes at the second epoch since the attack and recover liveness by seeking unanimous consensus among the remaining nodes in the shard.

It can be inferred that the expelled nodes, which fail to participate in the system at every τ, are not classified as one of the $\left\lceil \frac{2(N-1)}{3} \right\rceil + 1$ constant uncorrupted participants nodes. By referencing *Theorem 1* (the liveness guarantee theorem), given that the system maintains τ liveness guarantee for the honest nodes, there are sufficient honest nodes to ensure the constant safety and liveness guarantee of the control shards after deleting the expelled nodes. Consequently, the process shards can safely seek unanimous consensus among the remaining nodes.

Hence, the two-layer consensus protocol resists attacks on liveness and will always recover from the attacks while upholding safety. □

Theorem 3. *The two-layer consensus protocol guarantees safety, ensuring that only blocks approved by a unanimous vote in the first phase or by a majority vote in the second phase are accepted, and rejected blocks do not affect the state evolution in subsequent epochs.*

Proof. Please see the full proof in appendix B2. □

D. Cross-shard transaction

Cross-shard transactions involve the exchange of records between different process shards. Because nodes only know transactions within their own shards, the cross-shard transaction requires verifying the authenticity and timeliness of records from other process shards.

A shard in Gearbox may experience deadlock, so Gearbox requires liveness detection when doing cross-shard transactions. Reticulum does not have deadlock shards. The transactions in the process block is finalized and become usable when the corresponding control block is accepted. If consider the control shard as a shard in other sharding approaches, it allows direct plug-in for various cross-shard transaction protocols used in the classical sharded blockchains. Several approaches exist for processing cross-shard transactions. Omniledger [15] utilizes a client-driven method that collects and transmits availability certificates. RapidChain [27] divides cross-shard transactions into single-input single-output transactions and commits them in a specific order. Additionally, Monoxide [21] employs relay transactions for cross-shard processing. These methods can be seamlessly applied in Reticulum.

We show one possible design for implementing cross-shard transactions, which closely replicates the design implemented by Rapidchain. In particular, each cross-shard transaction includes unique participant identifiers, such as one or more wallet addresses for sending money (or any recorded message) and one or more addresses for receiving the funds. Each cross-shard transaction can be decomposed into a set of transactions involving a single sender address and a single recipient address in different shards. We refer to these individual transactions as Tx_{cross}. The Tx_{cross} is emitted by ps_{send} and send to $ps_{receive}$ which representing the shard containing the sender address and the shard containing the recipient address respectively. Alg. C in the appendix provides the pseudocode for this process. Note that we do not claim novelty for the cross-shard transaction designs.

Lemma 2. *The cross-shard transaction mechanism in Reticulum ensures the security and integrity of transactions between different process shards, providing transaction validity, consensus participation, proof of transaction, and a fixed shard membership approach.*

Proof. Please see the full proof in appendix B3. □

E. Shard size and time-bound

This section calculates the process shard size N_p, the control shard size N_c and the settings for time-bound.

Set the value of N_p: Let N be the number of nodes and N_p be the number of nodes in a process shard. The system has $\lfloor N/N_p \rfloor$ process shards and each sized at least N_p. Because nodes are randomly assigned to the shards, to fulfill a given P_{fp}, where P_{fp} represents the probability

for a *particular* process shard to be compromised, it yields the equation:

$$N_p = \log_{P_a}(P_{fp}) \tag{2}$$

where the maximum ratio of adversarial population in the system is P_a.

Set the value of N_c. Let N_c denote the number of nodes in a control shard. The control shard's safety threshold is $S = L < 50\%$. A control shard can output illegal blocks if adversarial nodes control more than $\lfloor \frac{1}{2} N_c \rfloor$ nodes. The failure rate of a control shard is:

$$P_{fc} = \sum_{i=\lfloor \frac{1}{2} N_c \rfloor + 1}^{N_c} \binom{N_c}{i} (P_a)^i (1 - P_a)^{N_c - i} \tag{3}$$

With a given P_{fc} and P_a, N_c can be determined accordingly. Tbl. I exhibits the size of the process and control shards with different P_a, P_{fp} and P_{fc}.

TABLE I
PROCESS SHARD SIZE AND CONTROL SHARD SIZE

$N_p\&N_c$ \ P_a / P_{fp}, P_{fc}	15%	20%	25%	30%	33%
10^{-5}	7&27	8&41	9&63	10&105	11&149
10^{-6}	8&35	9&51	10&79	12&131	13&185
10^{-7}	9&41	11&61	12&95	14&155	15&221

Security threshold: The failure rate represents the probability of any shard (regardless if it is a control or a process shard) outputting illegal blocks and is denoted as P_f.

$$P_f < \left((P_f)_{threshold} = 2^{-\sigma}\right) \tag{4}$$

with predefined security parameter σ.

Determining P_{fp} and P_{fc}: Given $(P_f)_{threshold}$ and N, we calculate the approximate $P_{f_{approx}}$ bounded by:

$$P_f < \left(P_{f_{approx}} = P_{fp} \times \beta + P_{fc} \times \Lambda\right) < (P_f)_{threshold} \tag{5}$$

where $\beta = \lfloor N/N_p \rfloor$ and $\Lambda = \lfloor N/N_c \rfloor$. We want to minimize N_p and N_c to allow as many shards as possible. Tbl. II shows several N_p and N_c with the given $P_{f_{approx}}$.

TABLE II
PROCESS SHARD SIZE AND CONTROL SHARD SIZE, $P_a = 33\%$, CALCULATED USING ALG. 4

$N_p\&N_c$ \ N / $P_{f_{approx}}$	500	1000	5000	10000	20000
10^{-5}	15&221	15&221	17&257	17&257	19&293
10^{-6}	17&257	17&257	19&293	19&293	21&329
10^{-7}	19&293	19&293	21&329	21&329	23&367

Parameters for time-bounds Setting up the time-bound for synchronous protocols is an open question for protocol designers. A large time-bound delivers redundancy, while a short time-bound may cause some nodes to be unable to catch up on the communication of the network. Previous work, e.g., Rapidchain runs a pre-scheduled consensus among committee members about every week to agree on a new time-bound Δ. Thus, in theory, Reticulum can use the same design to agree on new $T1$ and λ.

Algorithm 4 Determining P_{fp} and P_{fc} with given P_a and $P_{f_{approx}}$

1: **function** GET$N_p(P_{fp})$
2: **return** $\lceil \log_{P_a}(P_{fp}) \rceil$
3: **function** GET$N_c(P_{fc})$
4: **return** the smallest N_c that failure rate less than P_{fc}
5: **function** GETN_c AND $N_p(P_{f_{approx}})$
6: $\arg\max_i f(i) = \{i \times \lfloor N/GETN_p(i) \rfloor + i \times \lfloor N/GETN_c(i) \rfloor | f(i) \leq P_{f_{approx}}\}$
7: **return** $GETN_p(i)$ and $GETN_c(i)$

V. Analysis

This section shows an in-depth analysis for Reticulum.

A. Security Breaches

This section analyzes a possible equivocation of consensus associated with the design that leverages the liveness (L) and safety (S) of the shards. Indeed, The adversary cannot force the acceptance of a block containing incorrect information if the system only accepts a block when at least $(S \times M) + 1$ nodes in a shard sized M voted to accept it. This is because the system only allows at most S ratio of adversarial population in a shard (the probability for a shard to have more than S ratio of adversarial population is negligible.)

However, we prove in Lemma 3 that synchronous shards with $S \geq \frac{1}{2}$ or partially synchronous shards with $S \geq \frac{1}{3}$ are unable to achieve consensus without the possibility of equivocation, which can lead to security breaches. This issue also affects Reticulum because the process shards of Reticulum maintain $L = 0\%$ and $S < 100\%$.

Lemma 3. *Synchronous shards with $S \geq \frac{1}{2}$ or partially synchronous shards with $S \geq \frac{1}{3}$ are unable to achieve consensus without the possibility of equivocation.*

Proof. We know that the maximum number of adversarial nodes allowed in a shard is $f = S \cdot M$.

When a block containing correct information is being voted upon, the security assumption only guarantees that honest nodes receive the same set of $M - f$ honest votes from each other. Since $M - f < (S \times M) + 1$ in the synchronous model, or $M - f < (2S \times M) + 1$ in the partially synchronous model, it would require the adversary nodes to vote to accept the block. It is possible that the adversary nodes send inconsistent votes to different nodes. This implies that not all honest nodes will consider the block as having received more than $(S \times M) + 1$ votes for approval.

Consider a synchronous example where the adversary convinces an honest node, Alice, that a block A has

received $(S \times M) + 1$ votes in favor of acceptance, so Alice thinks the block is accepted. But all other honest nodes only received $S \times M$ votes in favor of acceptance, so they think the block is rejected. After the time-bound expired, the next leader node will propose a new block B to replace A. Since Alice believes that A has been accepted, she will vote to reject B. However, in some cases, it is still possible for B to obtain $(S \times M) + 1$ votes in favor of acceptance.

Ideally, Alice could attach the vote results she previously received when voting to reject B since A arrived first. However, if Alice is an adversary, she can post the vote results of A after block B has already received $(S \times M) + 1$ votes for acceptance, or even after new blocks have built on top of B.

This demonstrates that although the adversarial nodes cannot enforce the acceptance of adversarial blocks, they can still cause equivocation. Because the nodes outside the shard did not listen to the voting inside the shard, they could not tell either A or B received enough votes first, the system could not reach a unequivocal consensus. □

For these shards to function safely without external censorship, each honest node would need to initiate an instance of Byzantine Broadcast (BB) to share the votes it has received. However, the partially synchronous BB only functions with $f \leq \lfloor (M-1)/3 \rfloor$ [2], which contradicts the setting of f in the shard. The synchronous BB allows for an adversarial majority, but it requires $(\frac{M}{M-f} - 1)\Delta$ per vote, where Δ is the known communication time-bound. Therefore, only synchronous shards with $S \geq \frac{1}{2}$ may function independently, but they take a long time to finalize the voting process.

Gearbox uses a design in which each shard regularly posts a heartbeat (a collection of vote signatures) to a "control chain" that includes all nodes in the system. This design serves solely the purpose of detecting liveness. However, a new epoch of a shard does not wait for such a heartbeat to appear in the block of the "control chain." This oversight disregards the possibility of equivocation views among honest nodes.

Even if a new epoch starts when the heartbeat is confirmed, there are still problems. The leader node of the "control chain" must wait until a specified time-bound to generate the block and announce any dead shards, unless it has received valid heartbeats from all shards. However, since there is no consensus on the set of vote signatures (heartbeats), the leader node may mistakenly consider a shard as dead while other honest nodes believe otherwise. These circumstances may lead to multiple iterations of the consensus-reaching process in the "control chain" until the honest nodes are aligned.

Theorem 4. *Reticulum is secure when operating with $L = 0\%$ and $S < 100\%$ process shards and $L, S < 50\%$ control shards.*

Proof. Reticulum does not suffer from the security breaches indicated in Lemma 3 because the process shards do not function independently. The blocks of the process shards in Reticulum are confirmed in the corresponding control shard before the next epoch starts. The votes of the process blocks are broadcasted using synchronous BB with $f \leq \lfloor (N_c - 1)/2 \rfloor$ to all nodes in the corresponding control shard. Therefore, the honest nodes in the control shard are natively aligned and it is not possible to have consensus equivocation. We show in Sec. V-B that such a protocol only requires 4Δ in overall and at the communication level of $O({N_c}^2)$. □

Reticulum maintains the control shards instead of a "control chain" of Gearbox. This is to ensure that the vote broadcasting only involves N_c nodes, instead of N nodes.

B. Analysis of Overhead in Byzantine Broadcast Protocol Utilization

In existing sharding protocols, they adopt BFT protocols like HotStuff[1] to build replicae, which incurs $O(M^2)$ communication complexity, as it requires nodes to forward the proposal (block) to avoid equivocation. There is no need to forward votes and is safe to make a decision when a block receives $f + 1$ identical votes (assume synchronous communication) as it is impossible for a different verdict to receive more than f votes. When a consensus decision is made for a proposal, it only guarantees that the votes exceeded f, but not a consensus for the exact set of votes.

The first phase of Reticulum requires a determined set of votes to determine adversarial nodes, therefore, each node i utilizes a Byzantine broadcast protocol called $(\Delta+\delta)$-BB [2] to broadcast its vote V_i, which requires four steps:

1) Step 1: The broadcaster (node i) sends the proposal (V_i) with $O(N_c)$ complexity.
2) Step 2: Everyone votes for and re-transmit V_i with $O({N_c}^2)$ complexity.
3) Step 3: Everyone commits and locks with $O({N_c}^2)$ complexity.
4) Step 4: A Byzantine agreement is conducted, which incurs an overhead of $O(N_c)$.

Therefore, $(\Delta + \delta)$-BB demands 4Δ to complete and the communication complexity for determining a V_i in consensus is at the level of $O({N_c}^2)$. Thereby, the overall complexity for the BFT process in Reticulum reaches $O({N_c}^3)$, which is a big overhead.

To optimize this, we design that step 2 for all instances of $(\Delta+\delta)$-BB are combined together. It is safe to do so as step 1 must end at a fixed time-bound Δ. In this combined design, a vote in step 2 is a collection of votes for the proposals in step 1. When step 2 are combined, steps 3 and 4 are automatically combined.

Then Raticulum's two-layer consensus incurs the same complexity level: $O({N_c}^2)$ for send and forwards the proposal and $O({N_c}^2)$ for send and forward all the votes for the proposal. Therefore, Raticulum's communication overhead

is still at the level of $O({N_c}^2)$, just with a bigger constant than HotStuff or other BFT protocols.

C. Attacks on liveness with different τ

This section analyzes the trade-off between the requirement for the adversarial nodes and the system performance. When the adversary has more than τ nodes in a process shard, the adversary can permanently stop the process shard without being expelled from the system. This is achieved by each adversarial node taking terms to not participate in voting in the first phase of the two-layer consensus or the adversarial leaders propose incorrect blocks. Therefore, $\tau \geq N_p$ should be set to guarantee the eventual liveness of the process shard. The rate for successfully accepting a process block within the process shard is

$$R_p = \frac{\tau - a}{\tau} \tag{6}$$

where $\tau \geq N_p$ and a is the adversary nodes in the process shard. We allow $N_p - 1$ adversarial nodes in this shard in the worst case. Thus, considering the worst case, $R_p = \frac{\tau - N_p + 1}{\tau}$.

Fig.. 4 shows different τ corresponding to different N_p and R_p. We show some numerical results: considering the worst case R_p, to maintain a success rate $R_p = 40\%$, $R_p = 70$ or $R_p = 90\%$, an adversarial node in a process shard sized $N_p = 15$ may go offline for one epoch in each 24, 47 or 140 epochs respectively. This is a reasonable relaxation compared to the constant liveness requirement.

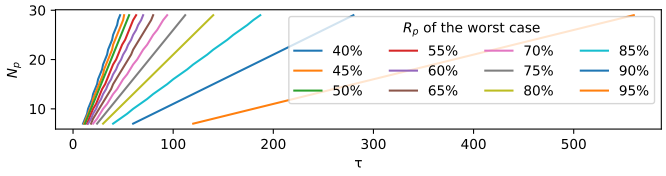

Fig. 4. τ corresponding to Rp of the worst case

If the adversary wishes to attack liveness and be expelled, it can stop every process shard and the first phase of the overall system for around $2a$ epochs. Afterward, each process shard maintains $R_p = 1$.

Discussion over τ liveness: The general security assumption for Rapidchain assumes that at any given moment, at least 2/3 of the computational resources belong to uncorrupted participants. It infers the constant liveness for the same set of honest participants as shown in Theorem 1. Without the constant liveness guarantee, the verdict reached is not deterministic, for the similar reason shown in the proof of Lemma. 3.

If we wish to apply τ liveness also to the honest nodes, the verdict reached must go through one round of byzantine broadcast tolerating a majority adversarial population before the next round of voting may start. which incurs significant costs. Therefore, the same as all the existing approaches, we just assume the constant liveness for the honest nodes.

Admittedly that allowing a larger τ (inferring a stronger R_p) would bring the stricter liveness requirement for the adversarial (unstable) nodes. One practical relaxation, in reality, is to design that the nodes are only being given a long suspension when identified as adversarial nodes. They may participate in voting after the prison term.

TABLE III
SOME NOTATION FOR THE CALCULATION

Notation	Description
E_{shard}	$E_{shard} = \lfloor N_c/N_p \rfloor$ denotes the number of process shards within a control shard.
E_{time}	denotes the length for one blockchain epoch. $E_{time} = T_1 + T_2$ for Reticulum.
B_s	$B_s = 2MB$ denotes the block size of the process block. It contains 4096 transactions.
E_{tx}	$E_{tx} = E_{shard} \times 4096$ denotes the number of overall transactions per epoch logged to the blockchain within a control shard in Reticulum or a shard in Rapidchain.
N_i	N_i indicates the number of nodes that store the i-th process block in the current epoch. N_i is determined by whether the process shard i reached a consensus in the first phase. If the process shard unanimously agreed on its process block, $N_i = N_p$; otherwise, $N_i = N_c$.

D. Analytical performance comparison with Rapidchain

This section provides the performance comparison between Reticulum and Rapidchain. Because there is no source code for Rapidchain available for experiments, this section provides an analytical comparison.

The two-layer consensus of Reticulum will first reject the process block which contains wrong information in the process shard and then again in the control shard. The nodes which proposed these blocks will be marked as adversarial nodes and be expelled. Therefore, in general terms, we would expect that the process blocks are correct, and it is only a matter of being accepted at the process shard or at the control shard. Therefore we may approximate the throughput (the number of transactions processed per second) and the storage requirement (the storage per transaction) as follows, please find some notations for this section in Tbl.. III:

Throughput: We calculate

$$Throughput = \frac{E_{tx}}{E_{time}} \tag{7}$$

R_p affects the length of T_2, so it affects the throughput.

Storage: S_{tx} measures the overall storage incurred in the nodes when the blockchain within a control shard logged one transaction. We calculate

$$S_{tx} = \sum_{i=1}^{E_{shard}} \frac{B_s \times N_i + (N_i - N_p) \times State_i}{E_{tx}} \tag{8}$$

where $B_s \times N_i + (N_i - N_p) \times State_i$ calculates the combination of the storage incurred for all the nodes that stored

process block i. $State_i$ is the size of the current state of process shard i. R_p determines N_i, so it also affects S_{tx}.

To illustrate the relationship between R_p and the performance and compare it with Rapidchain, we perform one mathematical simulation. In this simulation, we set $(P_f)_{threshold} = 10^{-7}$, $\lambda = 50$, $P_a = 33\%$, $N = 5000$, $N_c = 329$ and $N_p = 21$. Therefore, Reticulum has 15 control shards and each control shard has $E_{shard} = 15$ process shards inside. Each shard in Rapidchain maintains $L, S < 50\%$ so it sized N_c nodes, the same as the control shard of Reticulum. Therefore, Rapidchain has 15 shards. Because Rapidchain only uses one layer consensus, its performance is dependent on the size of its blocks and the number of shards. The two protocols have different settings. In order to relate the two works, we set the same upload bandwidth for broadcasting blocks. We also set the same E_{tx} for both approaches, so a block of a shard in Rapidchain sized $B_s \times N_p$, containing E_{tx} transactions. We make E_{time} different for the two approaches.

In Reticulum, the upload bandwidth requirement for the nodes sending the process block and the latest states to all nodes of the same control shard is denoted as UB. This may happen in the second phase of the two-layer consensus. 1) *In the worst scenario* when all process shards within the control shard fail to obtain the unanimous approval within T_1, then $T_2 = (E_{shard}+1) \times \lambda = 800s$ and $UB = \frac{B_s \times N_c + (N_c - N_p) \times state}{T_2 - \Delta}$. We assume that each *state* size $256KB$. It is intended not to define the structure of the state in this paper, as that is application-oriented. Here we assume, each state contains 10922 wallet addresses (160 bits each) and their balances within the process shard (32 bit each). 2) *In the best case scenario*, all process blocks obtained the unanimous approval in the first phase, then $T_2 = 50s$ and there is no need to post any process block to the control shard. Assuming $\Delta = 10s$, the upload bandwidth requirement to send the process block to the control shard in the worst case is, therefore, $UB \approx 952.708KB/s$. With this upload bandwidth, it is reasonable to set $T1 = \frac{B_s \times N_p}{UB} + 4\Delta \approx 86s$. Then in the worst case, one epoch for Reticulum lasts $E_{time} = T_1 + T_2 = 886s$. In the best case, it lasts $136s$.

To use the same upload bandwidth requirement for the leader node of Rapidchain, $UB = \frac{B_s \times N_c}{E_{time} + \Delta}$. $E_{time} = \frac{B_s \times N_c}{UB} - \Delta \approx 698s$, which is similar to T_2 of the worst case. In this setting, Rapidchain has a constant storage per transaction $S_{tx} = \frac{B_s \times N_c \times N_c}{E_{tx}} \approx 3608KB$, it does not need to sync the states and has a constant transaction per second of $\frac{E_{tx}}{E_{time}} \approx 88.022tx/s$ for a shard and $1320.330tx/s$ in the overall system (15 shards). Note that S_{tx} refers to the overall storage incurred in the network instead of the storage incurred for an individual node.

Fig. 5 shows the simulation for the theoretical performance of Reticulum with different R_p. E_{time} is calculated according to the given UB. It significantly outperformed Radpichain as, in theory, Rapidchain can be seen as the

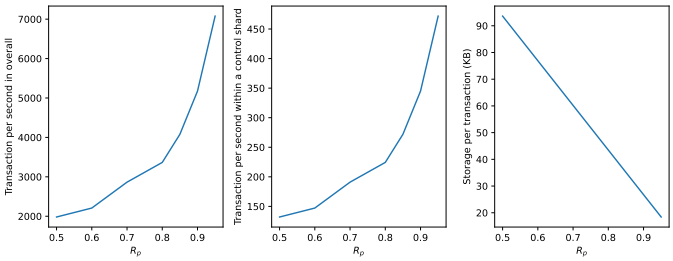

Fig. 5. The theoretical throughput and storage considering different R_p, with $UB = 952.708KB/s$

design when Reticulum deteriorates to the one-layer design that all process shards constantly failed. With $P_a = 33\%$, $N_p = 21$ and if set $\tau = 40$, we get $R_p = 50\%$ (assuming the worst case). In this case, the transaction per second for the overall system is $1980.5tx/s$, approaching the double performance of Rapidchain and only incurs $93.62KB$ storage per transaction overall in the network. When setting $\tau = 411$, we get $R_p = 95\%$ (assuming the worst case). In this case, the transaction per second for the overall system is $7077.645tx/s$ and only incurs $18.4099KB$ storage per transaction (as more transactions are only stored in the process shard) in overall in the network. Fig. 5 also confirms the general assumption of the larger R_p brings the stronger performance.

Note that we do not consider cross-shard transactions. As mentioned in Sec. IV-D, we may plug in different cross-shard transaction protocols and may use exactly the same design as Rapidchain. There are no different properties in this field between the two protocols.

VI. Experiment

This section provides a combination of experiments and mathematical calculations to evaluate the key characteristics of Reticulum and compare it with a Rapidchain-like protocol and also with Gearbox. The Rapidchain-like protocol (baseline) refers to an ordinary one-layer synchronous sharding protocol. We implement both baseline and Reticulum in Golang. Gearbox was evaluated purely via simulations on mathematical models since the authors provided no source code, and some detail designs not related to sharding are not presented, rendering experiments infeasible.

Experiment setup. We experiment Reticulum and baseline but simulate Gearbox. It is fair (maybe a little unfair for Reticulum) to make a comparison in this way, as we set the communication delays used in the mathematical simulation the same as we observed in the experiment environment for Reticulum. Our experiment was conducted on fifteen servers each equipped with 32-core AMD EPYC 7R13 processors, providing 128 vCPUs running at 3.30 GHz, 256 GB of memory, and a network bandwidth of 10 Gbps. We added a 200ms delay to each message to simulate the geographic distribution of nodes.

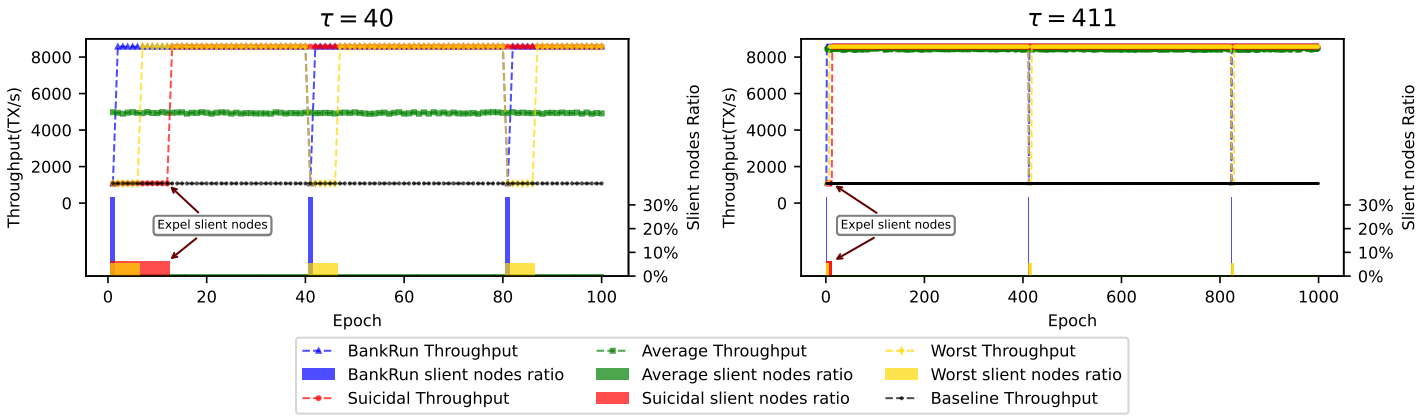

Fig. 6. The experiment result of a system of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $P_a = 33\%$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and different τ and the result of Baseline with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $E_{time} = 698s$ and each shard sized 329. Silent nodes stand for the nodes (globally not just within a shard) that did not vote for the process shard at the given epoch. The experiment setup matches the simulation setup we did in Sec. V-D. As can be seen from the picture, Reticulum outperforms Baseline in all cases. When τ is larger, the attack *worst* occurs less frequently. We need at least τ+1 epochs to illustrate BankRun attacks. Therefore, the experiment last 100 epochs for $\tau = 40$ and 1000 epochs for $\tau = 411$.

Attack strategies. We consider four kinds of attacks from the adversary. *BankRun*, where all adversarial nodes do not vote for the process blocks at a single epoch. *BankRun* can only occur once in every τ epochs. *Average*, where each adversarial node does not vote once in a random epoch in every τ epochs. *Worst*, where only one adversarial node refuses to vote at each process shard in every epoch. The adversary can stop a process shard for $i < \tau$ epochs in every τ epochs where i is the number of adversarial nodes inside this shard. *Suicidal*, is based on the *worst* strategy but all adversarial nodes vote at most $\tau - 2$ epochs in every τ epoch, and be expelled at the second time when they remain silent in voting.

Comparison between Reticulum and Baseline. Fig.6 suggests that Reticulum has superior performance.

Comparison between Gearbox and Reticulum. Gearbox assumes that the adversarial population in the system (P_{a-run}) is static, unknown but below P_a where $P_{a-run} \leq P_a \leq 33\%$. It continually refines shard sizes and liveness thresholds until an optimal arrangement is attained, where the adversarial population is conquered. However, Gearbox only discussed the mechanisms for when and how to rebuild a shard and to enlarge its size, which is very high level and lacks detailed designs. Therefore, we may not simulate the performance when rebuilding shards. But we do know how many overlapping shards a node is in after shards switch gears. Gearbox recommend the usage of four gears corresponding to liveness values of 10%, 20%, 25% and 30% within a shard. To align with our work, we additionally use a gear of 49%. With a given P_{a-run} ratio of adversarial nodes globally, we simulate the following process: At the beginning, every shard is built at the same size using $L = 10\%$. We check if a shard contains more than L of adversarial nodes, if so we rebuild it with a larger gear. This process is repeated

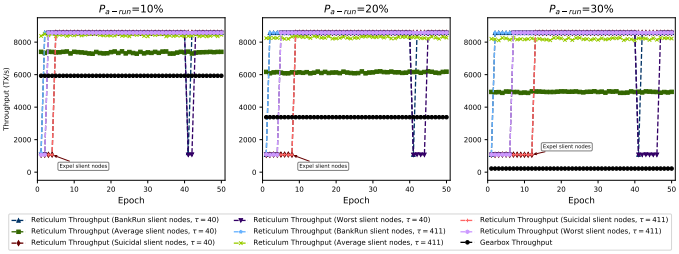

Fig. 7. The experiment result of a system of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and $\tau = 40$. Because Reticulum has methods to mitigate attacks on liveness and Gearbox does not have such features, all P_{a-run} nodes will actively attack in Gearbox, but will attack according to the attack strategies in Reticulum as we outlined previously. The figure also shows the result of Gearbox with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $B_s = 2MB$ (a block has 4096 transactions) after each shard has found a stable gear. In the biggest gear $L = 49\%$, a shard contains 329 nodes, which reaches the same size of a control shard of Reticulum.

until there is no shard that has an adversarial population of more than its gear. We then know the overlapping situation. We calculate the performance accordingly. Fig. 7 shows, with different P_{a-run}, the comparison between Reticulum and Gearbox after every shard of Gearbox has found a stable gear.

Readers are advised to see comparisons for storage and bandwidth usages in Appendix D.

VII. Conclusion

In conclusion, the Reticulum sharding protocol presents a promising solution for enhancing the scalability of blockchain technology. By separately considering attacks on liveness and safety and providing methods for liveness attack inhibition, Reticulum provided superior performance. The two-phase design of Reticulum, consisting

of control and process shards, enables nodes to detect adversarial behaviours and further split the workload and storage into tiny groups (process shards). The protocol leverages unanimous voting in the first phase to involve fewer nodes for accepting/rejecting a block, allowing more parallel process shards, while the control shard comes into play for consensus finalization and as a liveness rescue when disputes arise. Overall, the analysis and experimental results demonstrate that Reticulum is a superior sharding protocol for blockchain networks.

References

[1] Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Yin, M. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 106–118.

[2] Abraham, I., Nayak, K., Ren, L., and Xiang, Z. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (2021), pp. 331–341.

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

[4] Bagaria, V., Kannan, S., Tse, D., Fanti, G., and Viswanath, P. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 585–602.

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

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

[7] David, B., Magri, B., Matt, C., Nielsen, J. B., and Tschudi, D. 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.

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

[9] Huang, C., Wang, Z., Chen, H., Hu, Q., Zhang, Q., Wang, W., and Guan, X. Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding. *IEEE Internet of Things Journal 8*, 6 (2020), 4291–4304.

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

[11] Jingyi Zheng, Y. X. Reticulum source code. https://github.com/Jingyi-Zheng/Reticulum/, 2023.

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

[13] Khacef, K., Benbernou, S., Ouziri, M., and Younas, M. 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)* (2022), Springer, pp. 77–90.

[14] Kiayias, A., Russell, A., David, B., and Oliynykov, R. 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* (2017), Springer, pp. 357–388.

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

[16] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. 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* (2015), Springer, pp. 528–547.

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

[18] Ren, L., Nayak, K., Abraham, I., and Devadas, S. Practical synchronous byzantine consensus. *CoRR abs/1704.02397* (2017).

[19] Wang, C., and Raviv, N. Low latency cross-shard transactions in coded blockchain. In *2021 IEEE International Symposium on Information Theory (ISIT)* (2021), IEEE, pp. 2678–2683.

[20] Wang, G., Shi, Z. J., Nixon, M., and Han, S. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (2019), pp. 41–61.

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

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

[23] Xu, Y., Huang, Y., Shao, J., and Theodorakopoulos, G. A flexible n/2 adversary node resistant and halting recoverable blockchain sharding protocol. *Concurrency and Computation: Practice and Experience 32*, 19 (2020), e5773.

[24] Xu, Y., Slaats, T., and Düdder, B. Poster: Unanimous-majority - pushing blockchain sharding throughput to its limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2022), CCS '22, Association for Computing Machinery, p. 3495–3497.

[25] Xu, Y., Slaats, T., and Düdder, B. A two-dimensional sharding model for access control and data privilege management of blockchain. *Simulation Modelling Practice and Theory 122* (2023), 102678.

[26] Yu, H., Nikolić, I., Hou, R., and Saxena, P. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 90–105.

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

[28] Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., and Knottenbelt, W. J. Sok: Communication across distributed ledgers. In *International Conference on Financial Cryptography and Data Security* (2021), Springer, pp. 3–36.

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

Appendix

A. Related Work

Several sharding protocols have been proposed in recent years to address scalability issues in blockchain systems. This section reviews some of the most representative ones. It should be noted that there are alternative approaches, such as Prism [4] and OHIE [26], which are limited to the Proof-of-Work (PoW) setting, selecting blocks via

resource competition rather than a leader-and-vote-based consensus, and will not be discussed here.

Typically, each shard in a blockchain sharding protocol has a deterministic node membership per blockchain epoch. Each shard runs a BFT protocol to reach a consensus. Elastico [17] is probably the first sharding protocol proposed for public blockchains. During each blockchain epoch, each participant solves a proof-of-work puzzle based on randomness from the previous epoch (e.g., the hash of blocks, signatures, and transactions). The least significant bits of the PoW are used to form committees that run each shard and process transactions. However, the protocol becomes insecure when used with small committees; a separate study showed that the probability of a shard being unsafe becomes close to 97% after only six epochs [15]. OmniLedger [15] improves over Elastico in various aspects. It uses a synchronous PoW to generate identities and assign participants to shard committees on an independent identity blockchain. However, like Elastico, OmniLedger can only efficiently tolerate up to $t < N/4$ corruptions among the total number of parties in the system. RapidChain [27] is a synchronous sharding protocol that can tolerate up to $N/3$ corrupt parties among the total number of participants. It is initialized by a committee election protocol that selects a reference committee of size $M = O(\log N)$. At the end of each epoch, the reference committee is responsible for generating fresh randomness used to select committees for all the shards at the end of the first epoch and to reconfigure the committees of existing shards in subsequent epochs. Avarikioti et al. [3] presents a framework with security properties tailored for sharded ledger protocols, building upon the Bitcoin backbone model. The framework introduces the notions of consistency and scalability for sharded ledgers and analyzes various existing sharded ledger protocols in this framework.

A common issue of the works mentioned above is that they only consider tolerating an upper bound of the adversarial population, which may not perform well when the adversarial population is lower than the upper bound. Recent efforts [23], [7] attempt to adjust the shards according to the adversarial population. The first work for sharding that uses $S > 50\%$ to the knowledge of the authors, is proposed in 2020 [22], a proposal which seeks to achieve $f \leq (M-1)/2$ in a shard with $f \leq (N-1)/2$ in overall via node classification. However, the method's effectiveness depends on the adversary's strategy, which is inaccurately represented in the paper. Consequently, the protocol's reliability in real-world scenarios is uncertain. A protocol based on [22] enabling two-way adjustments to shard size based on the actual percentage of the adversarial population and the runtime workload, without overlapping shard membership, is proposed in [23]. However, the protocol is vulnerable to attacks by adversaries who frequently trigger adjustments in both directions, causing significant overhead for adjusting shards and data synchronization. Additionally, the paper inherits the error of [22], which makes it unreliable. Gearbox [7] is a protocol that enables one-directional adjustment of shard size based on the percentage of the adversarial population in the network. It operates under the assumption that the actual adversarial population in a shard is unknown but fixed and below the worst-case ($(L = S) < 50\%$). Gearbox cannot determine the runtime adversary population; it can only approximate it by gradually "switching gears." The system starts with a small shard size and hence a small L value. If a shard loses liveness, Gearbox continuously increases the shard size and its L value until it surpasses the actual adversarial population in the shards. At this point, the shard is considered alive and can output blocks. This approach, however, does not account for runtime fluctuations in the population of adversarial nodes or the presence of different adversaries over time. Maintaining a shard size that accommodates an all-time high adversarial population is inefficient when the percentage may vary at runtime. Simply making shard size adjustments in both directions to enable runtime increase and decrease of L will not work in practice. Because the attacker can easily trigger a loop in "switching gears", resulting in frequent and costly shard adjustments. Additionally, overlapping shards can occur in Gearbox, because it only resizes shards without adjusting the number of shards. The overlapping shards can suffer from duplicated workloads, and more transactions need to involve more than one shard, resulting in more cross-shard transactions and less parallelism. To maintain non-overlapping shards, resizing them to form new ones when a single shard loses liveness is necessary. However, this needs to divide or merge transactions governed by the original shards into new ones, which is complicated and costly. Thus, despite the limitation, Gearbox opts for not adjusting the number of shards to avoid the aforementioned issues. Lastly, the previous two works [22], [23] and Gearbox are insecure, as shown in Lemma 3.

B. Security analysis for the design of Reticulum protocol

1) Bootstrapping: .

Lemma. *The bootstrapping phase guarantees the safety property of shard membership assignment, ensuring that no node is assigned to multiple shards.*

Proof. To prove the safety property of the bootstrapping phase, we need to show that no node is assigned to multiple shards, i.e., there are no conflicts in shard membership assignments.

Suppose there exists a node $Node_j$ that is assigned to both ps_{index_1} and cs_{index_2}, where ps_{index_1} represents the process shard with index $index_1$, and cs_{index_2} represents the control shard with index $index_2$. This would imply a conflict in shard membership assignment.

Let's consider the getShardIndex function and its steps:

1) C_{index} is the index of $Node_j$ in the sequence C. Since C is generated using a random beacon and each node

is assigned a unique index, there are no duplicate indices.
2) $index_1$ is calculated as $\lfloor C_{index}/N_p \rfloor$. This calculation ensures that $Node_j$ is assigned to a specific process shard based on its index.
3) $index_2$ is calculated as $\lfloor index_1/(N_c/N_p) \rfloor$. This calculation ensures that $Node_j$ is assigned to a specific control shard based on its process shard index.
4) If C_{index} exceeds or is equal to $\lfloor N/N_c \rfloor \times N_c$, $index_2$ is decremented by 1 to avoid the creation of a new control shard. This adjustment ensures that nodes are not duplicated across multiple shards.
5) If $N \nmid N_p$, additional nodes are added to the last process shard but not a new process shard. The calculation of $index_1$ limits the assignment of nodes to the last process shard, avoiding duplication.

Based on the steps of the getShardIndex algorithm, we can conclude that the assignment of a node to a process shard and a control shard is unique and does not conflict with any other shard assignments. This guarantees the safety property of the bootstrapping phase, ensuring that no node is assigned to multiple shards.

Therefore, the bootstrapping phase design provides a safe and conflict-free mechanism for assigning shard memberships to nodes.

□

2) Two-layer Consensus Safety: .

Theorem. *The two-layer consensus protocol guarantees safety, ensuring that only blocks approved by a unanimous vote in the first phase or by a majority vote in the second phase are accepted, and rejected blocks do not affect the state evolution in subsequent epochs.*

Proof. To prove the safety of the two-layer consensus protocol, we will demonstrate that:

1) If a block is unanimously accepted in the first phase, it is considered safe and will not proceed to the second phase for further processing. Its vote signatures are sent to the control shard.
2) If a block does not receive unanimous voting in the first phase, it will proceed to the second phase for further processing.
3) Only the process blocks that are embedded in the control block and achieve majority approval in the second phase are accepted, ensuring safety.
4) Rejected blocks in the second phase do not impact the state evolution in subsequent epochs, maintaining safety.

Proof of Claim 1: Suppose a block B is unanimously accepted in the first phase. This implies that all nodes in the process shard agree on B. Since the consensus protocol used in the first phase is a binary vote-based and synchronous consensus protocol, all honest nodes will reach the same verdict for B within the time-bound T_1. As long as there is no adversarial behavior in the process shard, unanimous approval is guaranteed. Therefore, block B is considered safe and will not proceed to the second phase for further processing. The vote signatures are sent to the control shard to finalize the verdicts.

Proof of Claim 2: Now, consider a block B' that does not receive unanimous voting in the first phase. This implies that there are adversarial nodes in the process shard. Either the block B' is a block containing wrong information, or there exist adversarial nodes, not voting to accept B'. Since the consensus protocol requires unanimous approval for a block, B' cannot be considered safe without further verification. Hence, B' will proceed to the second phase for further processing.

Proof of Claim 3: In the second phase, the nodes in the control shard verify the process blocks embedded in the control block proposed by the leader node. If the control block achieves majority approval within the time-bound T_2, it means that the majority of nodes in the control shard agree with the decision made by the leader node. This majority agreement ensures safety by accepting only the process blocks that are marked as accepted by the leader node. The accepted blocks are considered safe and reliable for the subsequent state evolution.

Proof of Claim 4: If a process block of epoch $X+1$ is rejected in an accepted control block, the corresponding process shard will not undergo any new state evolution in epoch $X+1$. Instead, the state of the process shard in epoch $X+1$ remains the same as in epoch X. This ensures safety by maintaining the consistency and integrity of the state across epochs. Rejected blocks do not affect the state evolution in subsequent epochs, preventing any potentially malicious or incorrect blocks from disrupting the system's operation.

Based on the proofs of the above claims, we can conclude that the two-layer consensus protocol guarantees safety. Only blocks that achieve unanimous approval in the first phase or are embedded in the control block and achieve majority approval in the second phase are accepted, while rejected blocks do not impact the state evolution in subsequent epochs.

Therefore, the two-layer consensus protocol provides a reliable and secure mechanism for achieving consensus in a distributed system.

□

3) Cross-shard transactions: .

Lemma. *The cross-shard transaction mechanism in Reticulum ensures the security and integrity of transactions between different process shards, providing transaction validity, consensus participation, proof of transaction, and a fixed shard membership approach.*

Proof. To prove the security of the cross-shard transaction mechanism in Reticulum, we examine each of the key aspects identified in the security analysis.

Transaction Validity: The validity of each cross-shard transaction is ensured through signature verification

and balance sufficiency checks. In Algorithm C, both the sender shard (ps_{send}) and recipient shard ($ps_{receive}$) validate the signature of the transaction and verify that the sender's balance is sufficient. Only transactions with valid signatures and sufficient balances are processed and included in the respective shards' blocks.

Consensus Participation: Reticulum leverages a two-layer consensus process for including cross-shard transactions in the blocks of their respective shards. This ensures that the transactions are agreed upon by the participating nodes. The consensus process guarantees that the majority of nodes reach a consensus on the validity and order of transactions, preventing malicious actors from manipulating the transaction history.

Proof of Transaction: The cross-shard transaction mechanism in Reticulum includes the Merkle proof of the block containing Tx_{cross1} as proof to the recipient shard, $ps_{receive}$. This proof allows $ps_{receive}$ to verify the authenticity and integrity of the transaction. By validating the Merkle proof, $ps_{receive}$ can confirm that the sender has sufficient funds and that the funds have been deducted. This proof mechanism ensures the security and correctness of cross-shard transactions.

Fixed Shard Membership: Reticulum's fixed shard membership approach simplifies the handling of cross-shard transactions for approaches that leverage liveness and security by avoiding the complexities of dead shards and shard membership changes. With a fixed set of nodes participating in the consensus process for an epoch, the same as the classical protocols like Rapidchain or Ominiledger, the system achieves stability and security. Reticulum's deadlock-free design much simplified the process and the time to finalize the cross-shard transactions.

Based on the above analysis, the cross-shard transaction mechanism in Reticulum ensures the security and integrity of transactions between different process shards. By enforcing transaction validity, leveraging consensus participation, providing proof of transaction, and employing a fixed node membership approach, Reticulum establishes a secure foundation for cross-shard transactions.

Therefore, we can conclude that the lemma holds, and the security of cross-shard transactions in Reticulum is assured. □

C. Some pseudo-code

Algorithm 5 Handling of Cross-Shard Transactions

$ps_{receive}$

procedure HANDLETRANSACTION(Tx_{cross})
  if signature is valid **then**
    Generate two transaction: Tx_{cross1} ← deduct from sender address; Tx_{cross2} ← deposit to recipient address;
  else
    Discard this transaction
    return
  Send Tx_{cross1} to ps_{send}
  while Get the Merkle proof of the process block that contains Tx_{cross1} and the process block has been accepted **do**
    The current leader node adds Tx_{cross2} to the process block of $PS_{receive}$ as a normal transaction.

ps_{send}

procedure RECEIVE(Tx_{cross1})
  if signature is valid & balance of sender is sufficient **then**
    The current leader node adds Tx_{cross1} to the process block as a normal transaction
  else
    Discard this transaction
    return
  while The block that contains Tx_{cross1} has been accepted into the blockchain of ps_{send} **do**
    Send the Merkle proof of this process block to $ps_{receive}$

D. More experiment results

To emulate the outcomes of the Gearbox system, we instantiate a network comprising 5000 nodes, with a portion represented by P_{a-run} being designated as adversarial. These adversarial nodes are distributed randomly within the network. Nodes are subsequently assigned to shards through a random allocation process. We then assess each shard to determine whether the percentage of adversarial nodes exceeds the shard's predefined liveness threshold. In cases where this threshold is surpassed, we dissolve the existing shard and construct a new one with a higher gear setting. This process is iteratively executed until all shards maintain a percentage of adversarial nodes that falls below their respective liveness thresholds. There are five gears corresponding to the liveness threshold of 10%, 20%, 25%, 30% and 49%. The shard size corresponding to these gears are 26, 39, 50, 63 and 293 respectively, this is calculated considering $P_a = 33\%$. We repeat this entire procedure 1000 times, observing the distribution of shards across different gear configurations, as illustrated in Fig. 8. Fig 8 also shows the number of shards in Gearbox that a node is simultaneously located in (overlap times) with the given adversarial ratio in runtime. When a node is in multiple shards, it has multiple workload.

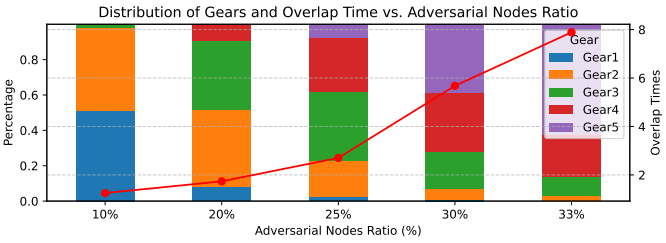

Fig. 8. The number of overlapping shards and the distribution of gears with different adversary ratio globally

As demonstrated in Section V-A, the shards within the Gearbox ecosystem face a critical challenge in achieving independent functionality for the majority of the time. Consequently, these shards must patiently await the finalization of their respective blocks within the control chain before commencing the next epoch. Failing to do so would expose them to the risk of consensus equivocation.

One potential remedy involves configuring the block interval of the control chain to be shorter than that of

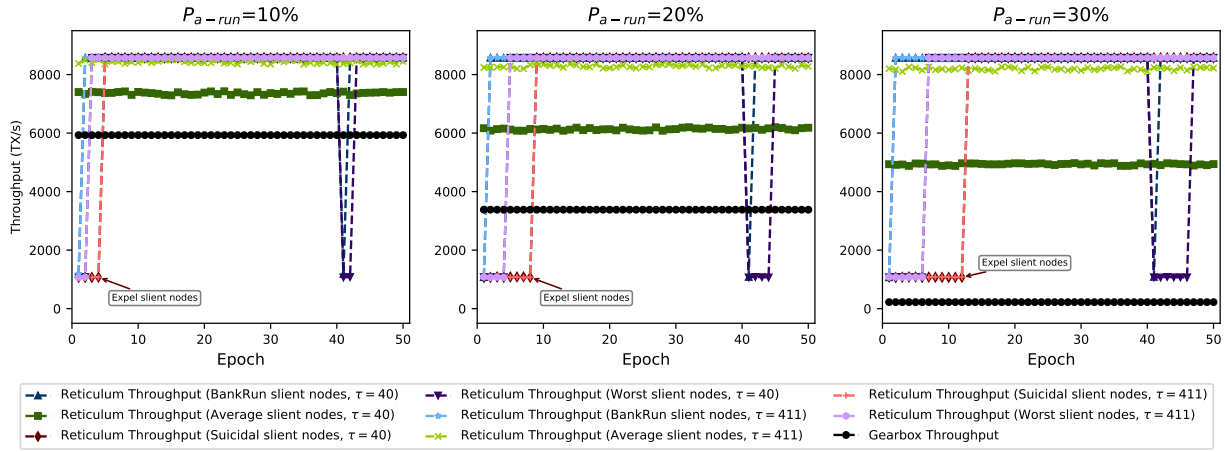

Fig. 9. The experiment result of a system of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and $\tau = 40$. Because Reticulum has methods to mitigate attacks on liveness and Gearbox does not have such features, all P_{a-run} nodes will actively attack in Gearbox, but will attack according to the attack strategies in Reticulum as we outlined previously. The figure also shows the result of Gearbox with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $B_s = 2MB$ (a block has 4096 transactions) after each shard has found a stable gear. In the biggest gear $L = 49\%$, a shard contains 329 nodes, which reaches the same size of a control shard of Reticulum.

each individual shard. This approach ensures that once a shard has accepted a block, and this decision has reached at least one honest node within the shard, the current leader node of the control chain can be promptly informed of this decision. This, in turn, allows for the incorporation of this decision into the next control block, substantially mitigating the risk of equivocation.

However, reducing the block interval of the control chain comes with a significant drawback: it imposes a substantial communication burden on network nodes. This is due to the fact that the acceptance of a control block necessitates the involvement of the entire system's nodes, which can be a considerable number. To address this issue, we have devised a design in which the entire system outputs only one control block for each block height. This ensures that the length of the blockchain within the shards remains identical to that of the control chain. Consequently, each shard must await the acceptance of the next control block before initiating a new blockchain epoch.

It is worth noting that in this simulation, we make the simplifying assumption of zero communication cost, implying that once the last shard reaches consensus, the control block is instantaneously generated and accepted by all nodes—a scenario that may be unrealistic in practice. Also, we should acknowledge that we do not account for the possibility of adversarial behavior by the leader node of the control chain, such as falsely claiming that certain shards are dead when they are still operational. Additionally, we do not address the potential scenario in which the control block is not accepted, which could introduce further complexities.

In terms of setting the time-bound for block generation and a round of voting within the shard, our approach closely follows the simulation design employed in the Gearbox paper. Within the Gearbox paper, it is mentioned that, in the context of the control chain, the latency in milliseconds for messages involving committees (comprising nodes) of size s can be reasonably approximated using the linear functions $l = 0.37s + 6$. Similarly, for the shard, the approximation is $l = 0.67s + 20$. Unfortunately, the paper does not delve into the specifics of how these numerical values were derived.

In our paper, we also take into account the latency incurred when disseminating data, and we model this latency as linearly proportional to the number of nodes involved. To ensure the time-bound for shard-related activities is greater than the latency incurred during message transmission, we've established a relationship that satisfies $l = 2.12x$ where x represents the number of nodes within the shard. Note that this estimation does not consider the potential slowdown when a node is in multiple shards and have multiple workload in parallel. 2.12 is chosen base on the fact that Rapidchain would be able to generate a block and conduct one round of voting that involves 329 nodes in a shard within $698s$, which is measured in runtime in the same experiment environment for Reticulum and Rapidchain. Consequently, the time-bound for block generation and the subsequent voting rounds are set at $55.12s$, 82.68, $106s$, $133.56s$, and $621.16s$, corresponding to the various shard sizes in different gears. This approach ensures that the time intervals are both consistent with the Gearbox paper's principles and suitably adjusted to account for the latency incurred in our specific context. Fig. 9 shows the performance of the comparison between Geaxbox and Reticulum in terms of throughput and Fig.10 shows that for the storage. The storage and download bandwidth comparison between Reticulum and Baseline are given in Fig.11 and Fig.12 respectively.

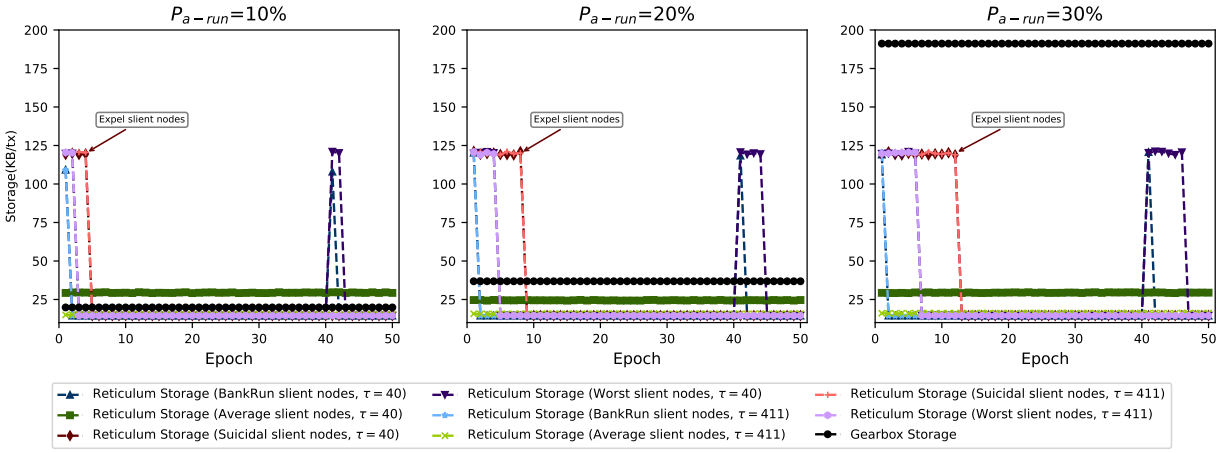

Fig. 10. The experimental results in terms of storage occurred over the entire network of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and $\tau = 40$. The figure also shows the result of Gearbox with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $B_s = 2MB$ (a block has 4096 transactions) after each shard has found a stable gear. As can be seen from the picture, Reticulum uncontestedly outperforms Gearbox in all cases. Note that the storage does not refer to the storage for a single node, but the storage occurred over the entire network.

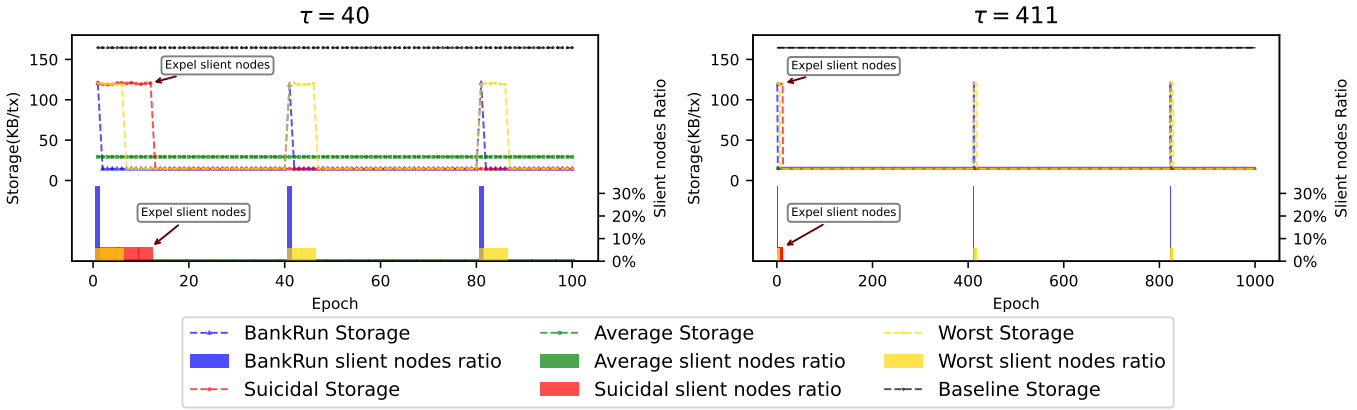

Fig. 11. The experiment result (storage) of a system of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $P_a = 33\%$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and different τ and the result of Baseline with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $E_{time} = 698s$ and each shard sized 329. $B_s = 2MB$ (a block has 4096 transactions). Silent nodes (globally not just within a shard) stand for the nodes that did not vote for the process shard at the given epoch. The experiment setup matches the simulation setup we did in Sec. V-D. As can be seen from the picture, Reticulum outperforms Baseline in all cases. When τ is larger, the attack *worst* occurs less frequently. We need at least τ+1 epochs to illustrate BankRun attacks. Therefore, the experiment last 100 epochs for $\tau = 40$ and 1000 epochs for $\tau = 411$. Because transactions are handled in the process shard level, which consists of fewer nodes than a shard in Baseline, nodes keep fewer transactions compared to Baseline. Note that the storage does not refer to the storage for a single node, but the storage occurred over the entire network.

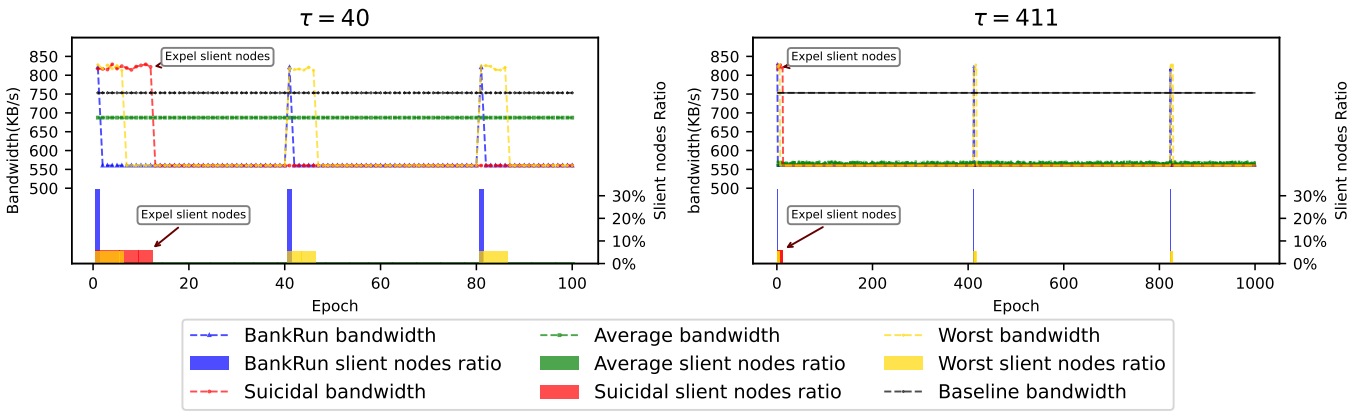

Fig. 12. The experiment result (download bandwidth) of a system of Reticulum with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$, $N_c = 329$, $N_p = 21$, $P_a = 33\%$, $T_1 = 86s$, $\lambda = 800$, $B_s = 2MB$ (a block has 4096 transactions) and different τ and the result of Baseline with $(P_f)_{threshold} = 10^{-7}$, $N = 5000$ and $E_{time} = 698s$ and each shard sized 329. $B_s = 2MB$ (a block has 4096 transactions). Silent nodes (globally not just within a shard) stand for the nodes that did not vote for the process shard at the given epoch.