Performance Analysis of a Live Sharded System

RnD Report (CS691)

Master of Technology
in
Computer Science Engineering

by

Debdoot Roy Chowdhury
(23M0765)

Under the Supervision of

Prof. Vinay J. Ribeiro

Department of Computer Science Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
Mumbai - 400076, India

May, 2024

Contents

1 Introduction

Blockchain sharding is a groundbreaking innovation in decentralized systems, tackling the scalability limitations that have long hindered blockchain networks. Sharding involves dividing the blockchain into smaller partitions known as shards, allowing for parallel processing of transactions and smart contracts. This report focuses on the implementation of blockchain sharding, particularly within the MultiversX [1] network (formerly Elrond), which utilizes sharding to enhance scalability and efficiency.

Implementing blockchain sharding requires careful coordination of various components. Initially, the network is segmented into shards, each responsible for processing a specific set of transactions. This partitioning enables parallel processing, leading to increased transaction throughput and reduced latency. Within each shard, consensus mechanisms like Proof of Stake (PoS) [3] or Practical Byzantine Fault Tolerance (PBFT) [2] validate transactions to maintain network integrity. Cross-shard communication mechanisms are essential for facilitating interactions between shards and ensuring the coherence of the blockchain.

Despite its potential benefits, blockchain sharding presents significant challenges. Security and decentralization must be preserved across shards, necessitating robust consensus mechanisms. Efficient cross-shard communication without introducing vulnerabilities is another key challenge. Managing shard state and data availability is crucial to maintaining the integrity of the blockchain, requiring solutions like state rent and synchronization protocols.

Analyzing the performance of live sharded systems, such as MultiversX, is vital for evaluating scalability and effectiveness. Performance metrics like transaction throughput and latency offer insights into system efficiency. This analysis guides protocol upgrades and optimizations to address emerging challenges and improve scalability. By iteratively enhancing performance, sharded blockchain networks can remain competitive and relevant in the evolving blockchain landscape.

2 Existing Live Sharded Blockchain System

Several blockchain projects have been actively working on or have implemented sharding to some extent. Here are a few projects that have been actively exploring or implementing blockchain sharding:

- **NEAR Protocol**
  NEAR Protocol [6] is designed for high-throughput decentralized applications (DApps) and aims to provide a scalable and developer-friendly blockchain platform. NEAR Protocol uses sharding to parallelize transaction processing across multiple shards, enabling increased scalability.

- **Zilliqa**
  Zilliqa [5] is a blockchain platform that implemented sharding to address scalability challenges. It uses a technology called ”sharding” to divide the network into smaller groups, or shards, each capable of processing its own transactions in parallel. Zilliqa has focused on delivering a high-throughput blockchain platform for decentralized applications.

- **Polkadot**
  Polkadot [8] is a multi-chain network that enables different blockchains to interoperate. While not a traditional blockchain sharding solution, Polkadot introduces a relay chain and parachains. Parachains can be considered as shards, and they operate in parallel, contributing to the overall scalability of the Polkadot network.

- **MultiversX** (Formerly Elrond)
  MultiVersX [1] is a blockchain platform that employs sharding to achieve high throughput and scalability. MultiVersX's architecture includes multiple shards, each responsible for processing a subset of transactions. This design allows MultiVersX to process thousands of transactions per second.

- **Avalanche**
  Avalanche [7] is a decentralized platform that uses a novel consensus protocol called Avalanche consensus. While not strictly traditional sharding, Avalanche achieves high throughput by allowing multiple subnets to run in parallel. Each subnet can be considered as a separate chain, contributing to the overall scalability of the network.

3 MultiVersX

A revolutionary development in blockchain technology, the MultiversX network offers a comprehensive solution that combines the simultaneous implementation of state, network, and transactions—the three essential components of sharding [4]. Together with its ground-breaking "Adaptive" component, this unique design enables dynamic network configuration, guaranteeing that security and scalability are maintained at ideal levels to accommodate fluctuating demand.

Moreover, MultiversX provides a strong substitute for conventional Proof of Work protocols by addressing the consensus issue through its Secure Proof of Stake mechanism. Through the mitigation of various attack vectors, this method not only improves network security but also allows for exceptional throughput and quick transaction execution, increasing system efficiency overall.

Beyond scalability and security, MultiversX's unmatched accomplishments go beyond a revolution in the blockchain industry by tackling some of the most complex consensus and sharding problems. Using this technological power, MultiversX can reach remarkable performance metrics with relatively low cost per transaction, even on basic hardware setups. Furthermore, its dedication to providing an exceptional developer experience transfers into improved accessibility and usability for end users, establishing a new benchmark for blockchain platforms in terms of both user pleasure and performance.

3.1 Architecture

3.1.1 Rounds and Epochs

The MultiversX network splits time into epochs and rounds, with an epoch consisting of a set number of successive rounds. The network's beginning phase is the first round of its first era, which is referred to as the genesis round. The timeline of round and epoch is show in Fig. 1.

Every round lasts the same amount of time over the whole network—it is now 6 seconds. Because every shard processes transactions simultaneously and synchronously,

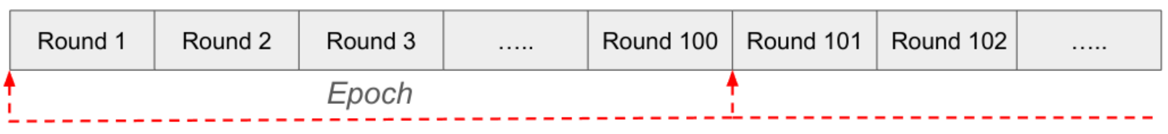
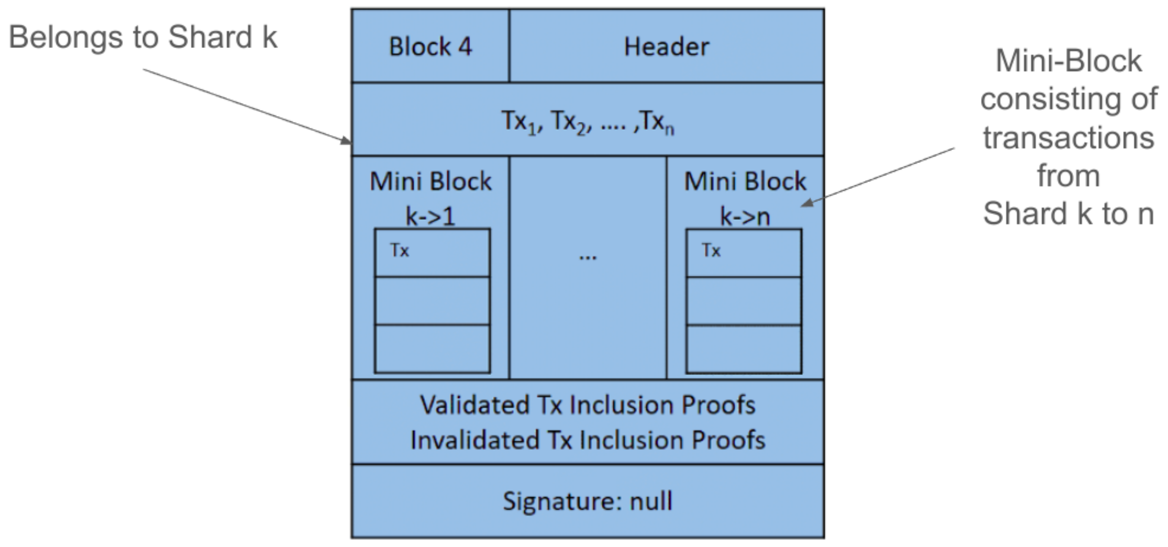


Figure 1: Timeline denoting Rounds and Epochs

every shard's round permits adding a maximum of one block to the shard's blockchain. There are situations in which a round ends without a block being added to the blockchain. These include situations where consensus is not reached or when the group's chosen leader is not present to propose a block.

An epoch is a set of consecutive rounds in which the configuration of the network is unaltered. First, epoch lengths of 24 hours are obtained by computing the number of rounds in an epoch. In the interim between epochs, the network can perform tasks to wrap up the previous epoch and get ready for the next, including calculating incentives for validator nodes and adjusting its architecture based on processing demands and network size. You can get more details about how the network adjusts its topology in Adaptive State Sharding and avoids node collusion.

3.1.2 Blocks and Mini-Blocks



Belongs to Shard k
Block 4
Header
Tx1, Tx2, ... ,Txn
Mini Block k->1
Tx
...
Mini Block k->n
Tx
Mini-Block consisting of transactions from Shard k to n
Validated Tx Inclusion Proofs
Invalidated Tx Inclusion Proofs
Signature: null

Figure 2: Structure of a Block with consisting MiniBlocks

Every Block is made up of several Mini-Blocks (Refer Fig. 2). A maximum of 5000 transactions can be saved in each Mini-Block, which is composed of similar-type Transactions (Txns with the destination address to a certain shard are stored in one Mini-Block). In addition, a block can include a maximum of 20,000 transactions. To expedite retrieving and storing, it is helpful to divide up transactions into distinct mini-blocks.

3.1.3 Meta-Shard

The Meta-Shard will notarize all network and global data activities (node joining, node leaving, computation of eligible validator lists, node assignment to the shard's waiting lists, and consensus agreement on a block in specific shard challenges for invalid blocks).

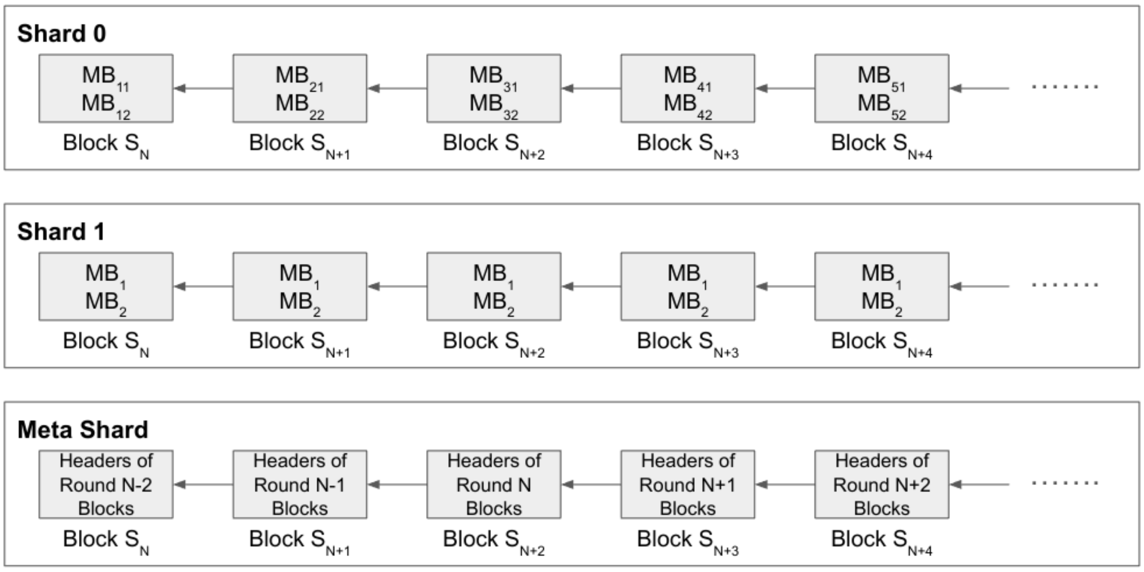


Figure 3: Shards along with Meta-Shard

A distinct shard that supports cross-shard activities and communicates with all other shards is in charge of managing the Meta-Shard consensus. Block headers and, if needed, proofs for the challenges of the invalid blocks are sent to the Meta-Shard by the other shards at the beginning of each epoch's cycle. Blocks containing this data will be assembled on the Meta-Shard, where consensus must be reached. To safely handle cross-shard transactions, shards can ask for details about blocks, miniblocks, eligible validators, nodes on waiting lists, etc. after the blocks have been confirmed by the consensus group.

From Fig. 3 it can be figured out that in every round the Meta-Shard includes block headers of the $N-2$th round where N is the present round.

3.2 Adaptive Based State Sharding

The binary tree structure, which handles state transitions, enhances scalability, and distributes account addresses, is the foundation of the whole sharding technique.

The tree structure a logical depiction of the account address space that is utilized for deterministic mappings, such as sibling calculation, shard allocation, etc. The shards with their ID numbers are represented by the leaves of the binary tree. If there is just one shard/leaf (a), all account addresses are mapped to it, and all transactions will be carried out here, starting with the root (node/shard 0). Moreover, the address space will be divided into equal halves based on the last bits in the address if the calculation for the ideal number of shards requires two shards (b).

In certain cases, if the ideal number of shards is not a power of two, the tree may also get out of balance. Only the leaves on the last level are impacted by this situation. When the quantity of shards reaches a power of 2, the structure will once more achieve

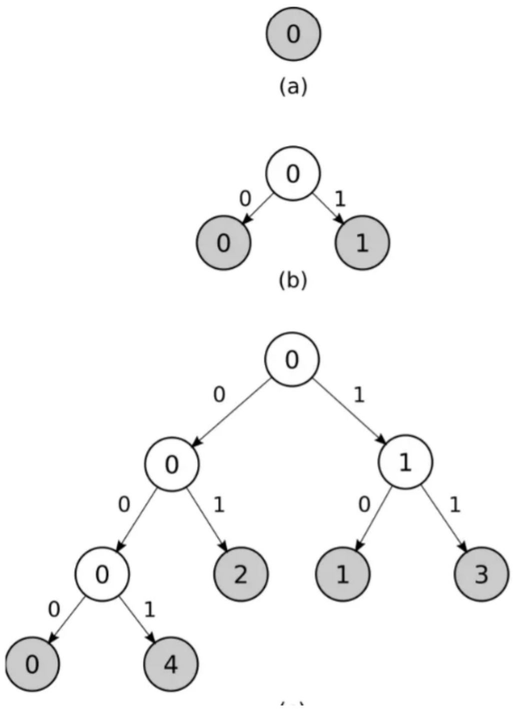


Figure 4: An example of the Sharding Scheme

equilibrium. It may be claimed that active nodes assigned to these shards would earn less in fees since the unbalanced binary tree causes the shards at the lowest level to have half the address space of nodes of a shard situated one level higher; block rewards are unaffected. Nonetheless, this issue is resolved by randomly redistributing one-third of each shard node per epoch (as explained in the Chronology section) and maintaining a balanced node distribution based on the tree level.

Looking at the tree Fig. 4, starting from any leaf and going through branches towards the root, the encoding from branches represents the last n bits of the account addresses that will have their associated originating transactions processed by that leaf/shard. Going the other way around, from root to leaf, the information is related to the evolution of the structure, sibling shards, the parent shard from where they split. Using this hierarchy, the shard splits when the threshold for transactions or nodes gets crossed or the shards. The entire state sharding mechanism benefits from this structure by always keeping the address and the associated state within the same shard.

3.2.1 Shard Redundancy

When there are not enough online nodes in a shard or when the distribution is restricted geographically, state sharding on the blockchain is prone to shard failure. If one shard were to fail—that is, if all nodes were offline or if consensus could not be reached because more than one-third of the nodes are not responding—there is a significant chance that the entire architecture would depend solely on super-full nodes, which download every block from every shard in its entirety and thoroughly verify everything.

MultiVersX's protocol includes a protection mechanism that, as seen in Fig. 5, forces the shards from the last tree level to also keep the state from their siblings, therefore introducing a tradeoff in the state holding structure. When sibling shards merge, this

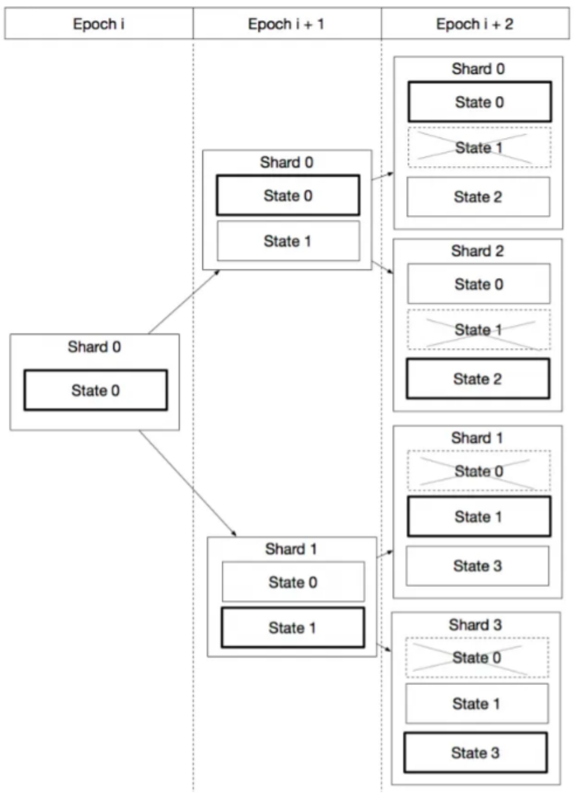


Figure 5: An example of the Shard Redundancy across Epochs

approach minimizes communication and does away with bootstrapping because the shards already contain the data.

3.3 Secure Proof of Stake

The ideal dimension for the consensus group is combined with the selection of random validators, eligibility based on stake and rating, and the formation of SPoS. The following stages outline the algorithm:

1. A tuple consisting of the public key, rating, and locked stake defines each node. It must first register through a smart contract, sending a transaction containing the minimum amount needed to participate in the consensus as well as additional information, if it wants to take part in it.

2. At the conclusion of the current epoch e, the node joins the node pool and awaits the shard assignment. All of the nodes that joined during epoch e and all of the nodes that require rearranging (less than one-third of each shard) are combined into a new set of nodes by the shard assignment process. Every node in this group will be moved to the shard waiting lists.

3. The node will be added to a shard's list of eligible nodes after the joining epoch.

4. Based on a set of variation parameters, the round r, the randomness source added to the previous block, and the eligible list, a deterministic function can choose each node to include in an optimum sized consensus group (with respect to security and

communication). It is impossible to guess the random number until the block is actually signed by the previous consensus group, which is known to all shard nodes through rumors. Due to this feature, it is a good source of randomness and shields against harmful attacks that are very adaptive. The selection function defined by MultiVersX returns the consensus group, or the collection of selected nodes, the first of which is the block proposer.

5. The validators will co-sign the block, which will be constructed by the block proposer using a modified practical Byzantine Fault Tolerance (pBFT).

6. A new consensus group will be chosen using round r and the randomness source from the previous block if the block proposer failed to construct a block during the allotted time period for whatever reason (malicious, offline, etc.).

To reduce the communication overhead that comes with an increased number of shards, a consensus will be run on a composite block. This composite block is formed by:

1. **Ledger Block**: The block containing all cross-shard and intra-shard transactions for which confirmation proof was obtained, to be added to the shard's ledger.

2. **Multiple Mini-Blocks**: The block containing cross-shard transactions for an alternative shard. On the composite block that contains both intra- and cross-shard transactions, the consensus will only be run once. Each shard's block header is forwarded to the metachain for notarization once a consensus has been obtained.

3.4 Block Confirmation

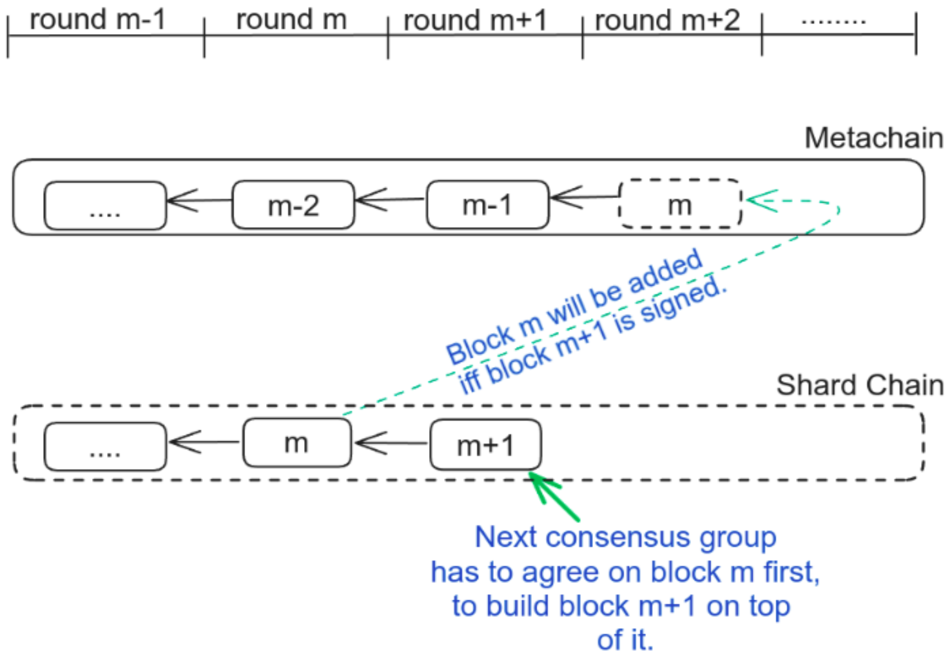

round m-1
round m
round m+1
round m+2
....
Metachain
....
m-2
m-1
m
Block m will be added iff block m+1 is signed.
Shard Chain
....
m
m+1
Next consensus group has to agree on block m first, to build block m+1 on top of it.

Figure 6: Confirmation of a Block

When a block reaches the meta-shard chain, it is considered confirmed. But after it enters its particular shard chain, it needs to wait for one more cycle. Only valid blocks will be stacked one on top of the other by a consensus group. One round's delay ensures that the shard will only be at risk in the extremely unlikely event if the majority of the malicious group of nodes is selected for two rounds in a row. In the event of cross-shard transactions, where the destination shard block must enter the meta chain in order for the block to be confirmed, the block confirmation time is different.

Given Figure 6, Block m in the shard chain, will only get added to the meta chain once another block $m + 1$ is added on it. So, 2 extra rounds are required for the confirmation of an intra-transaction once it's respective shard has processed it.

3.5 Cross-Shard Execution

The block structure is represented by a list of miniblocks for each shard that have the actual transactions inside, and a block header that contains information about the block (block nonce, round, proposer, validators timestamp, etc.). All transactions with the sender in the current shard and the recipient in a different shard, or the sender in a different shard and the destination in the current shard, are contained in each miniblock. The quantity of miniblocks in a block containing the same sender and recipient is unrestricted. This means that a block may contain several miniblocks that have the same sender and recipient.

MultiVersX employs an asynchronous approach for its cross-shard transaction strategy. The sender's shard performs validation and processing first, followed by the recipient's shard. Since the sender may completely validate every transaction started from the account in this shard, namely the current balance, transactions are first sent in this shard. After then, the nodes in the receivers' shard only require the proof of execution provided by the metachain to verify signatures, scan for replay attacks, and ultimately update the receiver's balance with the transaction amount.

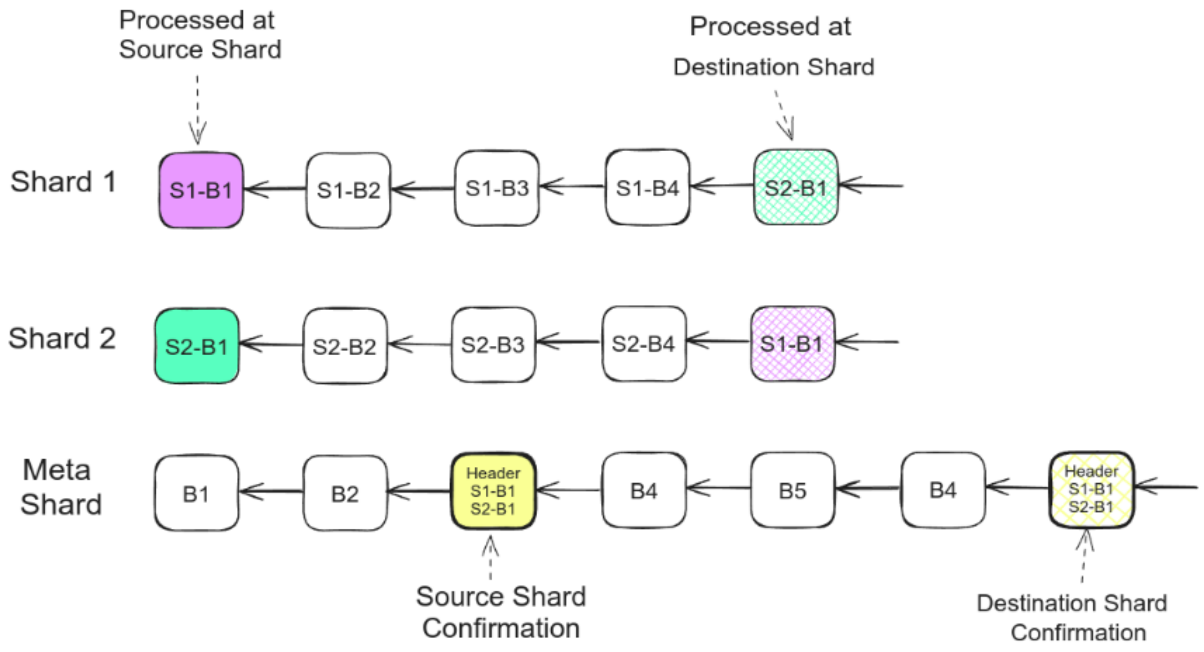


Figure 7: An example of the Cross-Shard Transaction timeline

As seen in Fig. 7, Block S_i-B_j represents the jth block on the shard chain i. A block of cross-shard transactions is created by Shard 1 and 2 with each transaction ending in the opposing shard. In the metashard, the transactions are validated following 2 rounds. After one round of processing in the meta shard, these blocks are processed one more time in the final shard. After the state is applied in the final shard and the blocks are processed one last time in the meta shard, the final confirmation takes place.

3.6 Local Testnet and Proxy Network

The testnet and primary SDK have been constructed using the Go language, employing Docker to create containers, each serving as an autonomous node. Interfacing with the testnet and executing fundamental operations is facilitated through the mxpy SDK, scripted in Python.

The mxpy SDK operates atop the MultiVersX Proxy network, utilizing proxy nodes to engage with the chains. Python-based scripts developed for performance analysis make extensive use of these libraries. This framework underscores a robust infrastructure for testing and evaluating network efficiency, offering insights into the system's functionality and performance metrics within the context of blockchain technology.

4 Development

This section encapsulates all developmental endeavors conducted for the purpose of analysis. Such endeavors encompass additions to code, modifications to existing network infrastructure, fine-tuning of the proxy network, and utilization of API endpoints to retrieve pertinent values. These initiatives were aimed at enhancing the functionality and efficiency of the system, facilitating comprehensive examination and assessment of its performance. Through meticulous coding adjustments, network optimizations, and strategic utilization of API endpoints, a robust foundation was laid for in-depth analysis, enabling the extraction of valuable insights into the intricacies of the system's operation within the blockchain ecosystem.

4.1 Creating Nodes and Accounts

The pre-existing code within the testnet framework is designed to initialize a predefined number of nodes upon network startup. As previously outlined, this initialization process entails the creation of distinct containers to represent each individual node. To extend this functionality to runtime, we adopted the same approach, allowing for the dynamic creation of nodes as required.

Each node within the network is associated with an individual account, from which it receives rewards for validation and staking purposes. However, a challenge arose when creating new accounts at runtime, as there was no established method to initialize them with a default balance. To address this issue, we developed a method whereby newly created accounts are collectively funded by validators from their existing balances. Consequently, validators are initialized with substantial balances to facilitate the seamless operation of the network.

4.2 Loading Transactions to Network

Three distinct methods have been devised to saturate the network with a maximal volume of transactions, each method specializing in a particular transaction type. The first method exclusively injects intra-shard transactions into the network, while the second method focuses solely on cross-shard transactions. Conversely, the third method combines both intra-shard and cross-shard transactions to comprehensively inundate the network.

Each of these methods is configurable to achieve the desired load quantity over a specified duration. Notably, the third method, which encompasses both intra and cross-shard transactions, offers the flexibility to allocate a fraction of the load to each transaction type, allowing for fine-tuning of the transaction mix. Through these meticulously designed methods, the network can be rigorously stress-tested under various transaction scenarios, enabling comprehensive evaluation of its performance and scalability.

4.3 Measuring Throughput

To gauge the network's throughput, a systematic approach has been devised based on a timeline and the number of transactions processed within each block. Over a span of several rounds, the total number of transactions processed in every block serves as a metric for the network's present throughput.

For intra-shard transactions, a transaction is considered processed if it successfully enters a block within its respective shard. Conversely, for cross-shard transactions, processing entails inclusion in a block within the destination shard.

A dedicated script has been meticulously crafted to orchestrate this process. Upon commencement of the experiment, the script logs the round time and proceeds to utilize the Proxy mechanism in each subsequent round to retrieve the blocks appended to the chain of each shard. The cumulative count of transactions within these blocks serves as an indicator of the network's throughput, allowing for a comprehensive analysis of its operational efficiency and transaction processing capabilities.

4.4 Measuring Confirmation Time

Confirmation of transactions occurs upon their inclusion in the meta-shard chain. Intra-shard transactions achieve confirmation within 3 rounds, while cross-shard transactions require 5 rounds. Confirmation throughput is gauged similarly to processing throughput, tracking the inclusion of transactions in each round. A meticulous script monitors the network's activity, noting round times at the experiment's onset. Leveraging the Proxy mechanism, the script traverses each round, tracking transaction inclusions in the meta-shard chain. Aggregating confirmed transactions over the specified timeframe provides insight into the network's confirmation efficiency, critical for evaluating reliability and scalability.

5 Experiments

For experiments, we simulated the local testnet with various configurations of shards and integrated their Proxy Node with both Go and Python SDKs for interaction with the testnet. A Load Testing repository in Python was created to interface with the local testnet, facilitating account creation, transaction loading, and network status analysis. Changes were made in the local testnet code to align with the Load Testing script.

Each experiment was conducted over 32 consecutive rounds with varying loads on a machine with configurations of Intel 5th Gen Processor, 16 GB of physical Memory and a constant Bandwidth of 100 MiB.

5.1 Loading Intra-Shard Transactions

The objective of this experiment is to ascertain the peak throughput of the network concerning solely intra-shard transactions, while also exploring the impact of varying shard numbers and imposed loads on achieving maximum throughput.

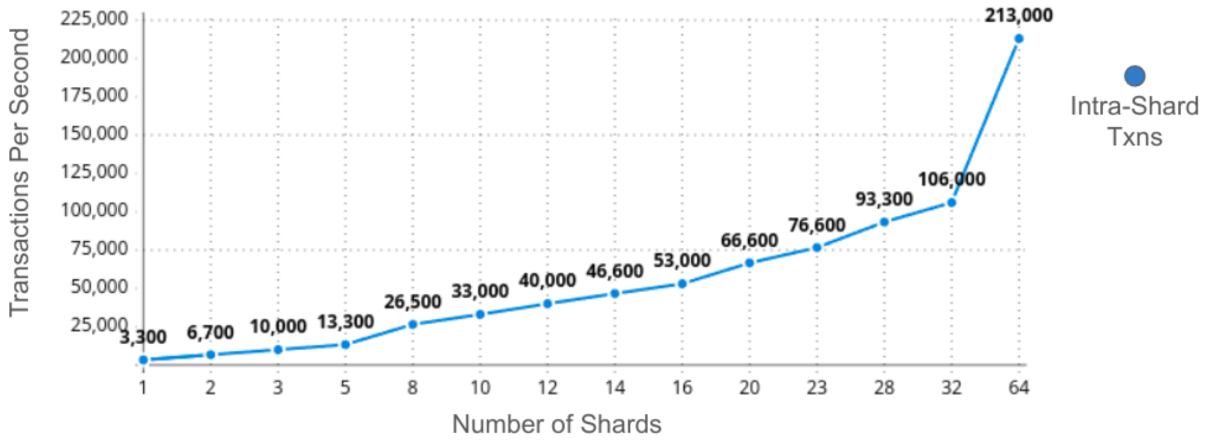


Figure 8: Observation of the Experiment in TPR Vs. Number of Shards

Several notable observations emerged from the experiment (Fig. 8):

- Notably, whenever the network approached maximum throughput levels for consecutive rounds, it responded by subdividing a single shard into two, thereby augmenting the overall throughput capacity.
- A significant bottleneck within the system was identified in the form of the proxy node, primarily due to its limited buffer size, which impeded the network's throughput potential.
- It's imperative to highlight that the testnet's architecture imposes an upper limit of 64 shards, restricting scalability beyond this threshold.
- An additional concern surfaced regarding the potential bottleneck effect within the metachain. This bottleneck may manifest in confirmation latency if the metachain becomes saturated with shard block headers, necessitating more than one meta block to confirm transactions effectively.

These observations underscore critical considerations in optimizing network throughput and scalability, highlighting areas for potential refinement and architectural enhancements to ensure sustained performance under varying load conditions.

5.2 Loading Cross-Shard Transactions

The primary aim of this experiment is to determine the peak throughput of the network exclusively concerning cross-shard transactions. Furthermore, the study seeks to investigate the influence of varying shard numbers and imposed loads on achieving optimal throughput levels.

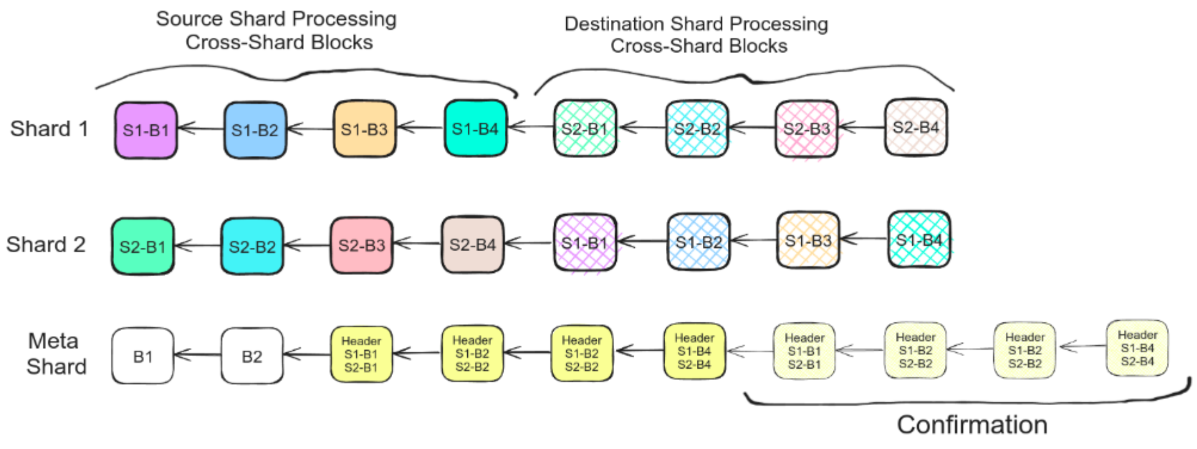


Figure 9: Maximum throughput with only Cross-Shard Transactions

The maximum throughput achieved when the transaction load consists solely of cross-shard transactions is depicted in Fig. 9. It is noteworthy that before the transaction block from the source shard reaches the destination shard, the source shard has the capacity to process an additional three blocks. This observation implies that a shard can process a total of four blocks within an eight-round interval if the blocks exclusively contain cross-shard transactions. Consequently, the throughput in this scenario is expected to be approximately half compared to the case of intra-shard transactions, where eight blocks of transactions are processed within the same eight-round timeframe.

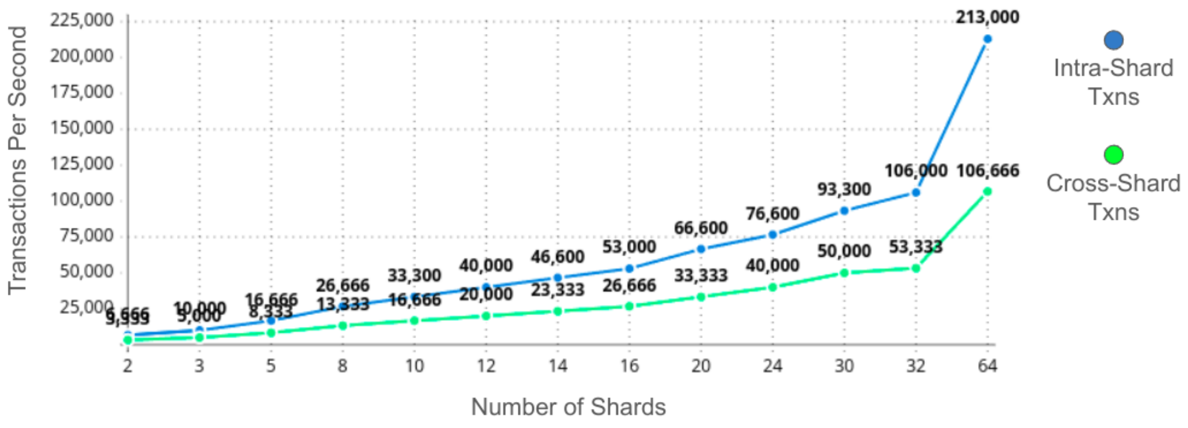


Figure 10: Observation of the Experiment in TPR Vs. Number of Shards

Several notable observations emerged from the experiment (Fig. 10):

- It was observed that the throughput attained through cross-shard transactions is precisely half of that achieved with intra-shard transactions when every block reaches its maximum capacity.
- Uniform Destination Account Selection: Each transaction's destination account was uniformly selected from all accounts within the network. Consequently, each shard, among its total transactions in a block (20k), experiences an equitable distribution of transactions to all other shards. As a result, a shard only requires one block to process all transactions originating from other shards.
- Despite the idealized distribution observed in the experiment, it is crucial to acknowledge potential deviations in real-time scenarios. In practical deployment settings, certain shards may encounter overwhelming loads when finalizing cross-shard transactions. This possibility raises concerns regarding potential bottlenecks and latency issues within the network.

5.3 Loading Mixed Shard Transactions

The primary aim of this experiment is to determine the peak throughput of the network with different percentages of types of transactions. Furthermore, the study seeks to investigate the influence of varying shard numbers and imposed loads on achieving optimal throughput levels.

We varied the percentage of Intra-Shard transaction from 90% to 10% and noticed the how much the throughput is scaled. As with the increase of cross-shard transactions, the throughput should show a decline if we make the number of shards constant. But with the increase in the number of shards, the throughput should generally increase for all the cases.

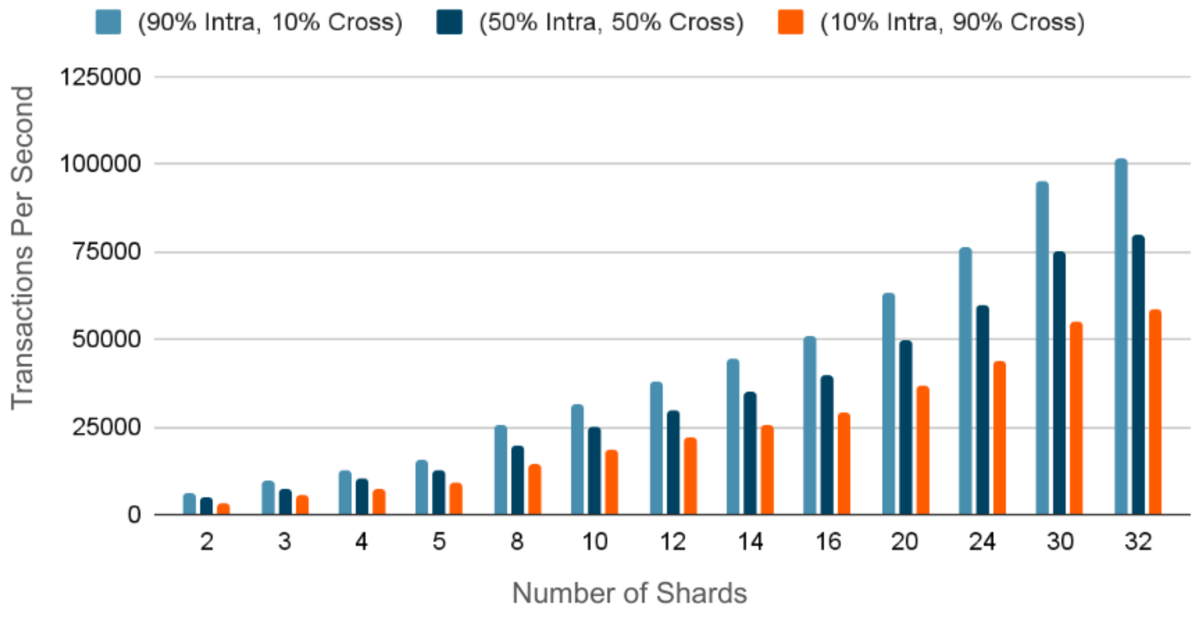


Figure 11: Observation of the Experiment in TPR Vs. Number of Shards

Notable observations (Fig. 11):

- Network with lower number of Shards performs better than network with higher number of Shards when Cross-Shard transaction percentage decreases from 90% to 50%.
- So, dynamic pruning of shard tree is required to increase the throughput.

5.4 One-Vs-All Cross-Shard Transactions

The objective of this experiment is load a shard with transactions from every other shards. It resembles the one vs all scheme where every shard creates blocks full of transactions which end at one particular shard.

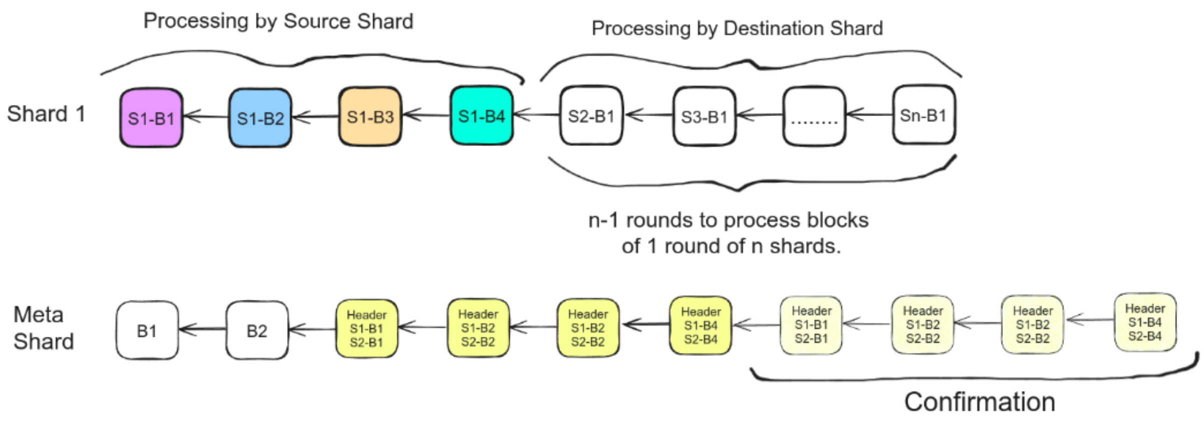


Figure 12: An example of the OvA Cross-Shard Transaction Timeline

The shard which receives these loads of cross-shard transaction will fail to process them in 1 round. It may lead to N rounds if N shards sends blocks full of transactions (Refer Fig. 12).

For instance, if 2 shards send 2 blocks full of transactions to a specific single shard, that shard will need 2 rounds to process those two blocks. Similarly for N-1 shards trying to load transactions to Nth shard, it will take N-1 rounds to process all of the N-1 blocks for the Nth shard.

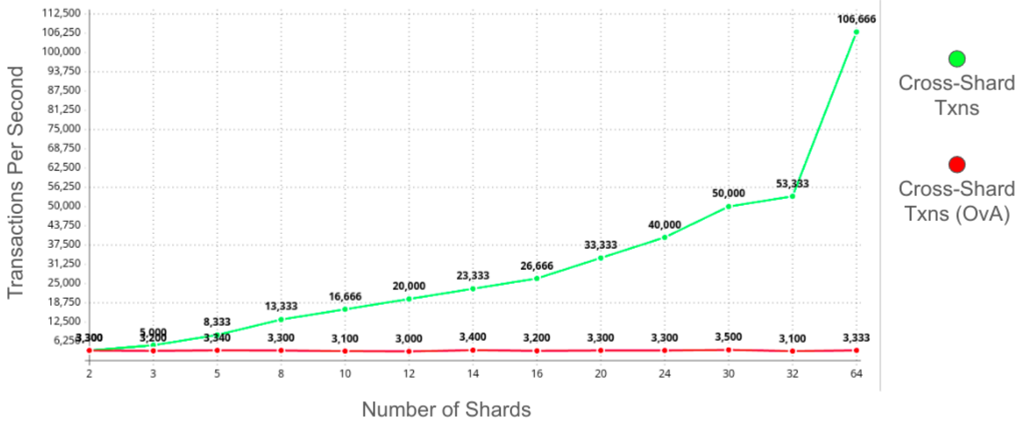


Figure 13: Observation of the Experiment in TPR Vs. Number of Shards

Notable observations emerged from the experiment (Fig. 13):

- With a timeline of 32 rounds where the source shard can create 4 blocks at the first 4 rounds, the rest of the round it spent finalizing the cross-shard blocks from other transactions. Thus, even if we increase the size of the shard, the throughput remains constant to 3500 TPR.
- Shard re-adjustment or any other method must be used to distribute the cross-shard transactions to other shards.

5.5 Increase in Cross-Shard Transaction w.r.t. increase in Shard

This experiment exploits the tree sharding infrastructure of this scheme and analyzes by how much percentage the cross-shard transaction is increased if we increase the number of shards. As the sharding structure of this scheme divides one shard address space into two when sharding occurs, paves the way for an increase in cross-shard transactions when the number of shards grows. If a shard generates transactions uniformly, the percentage

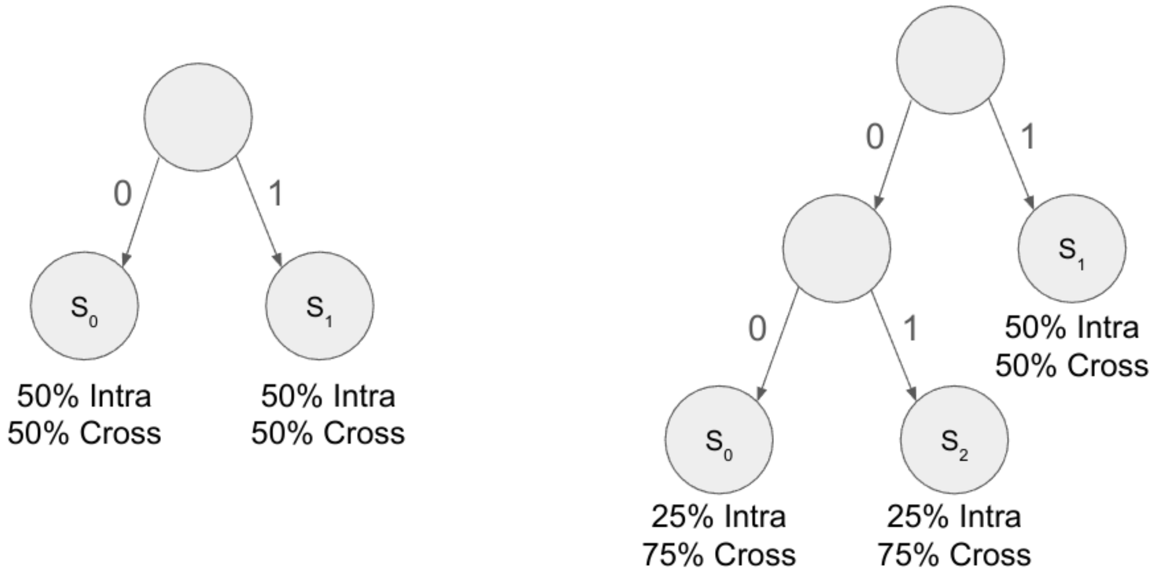


Figure 14: Distribution of generated transaction for 2 and 3 shards

of cross-shard transactions generated should increase with the increase in the number of shards. As seen in Fig. 14 the share of cross-shard transaction jumps from 50% to 75% with the increase in shard from 2 to 3.

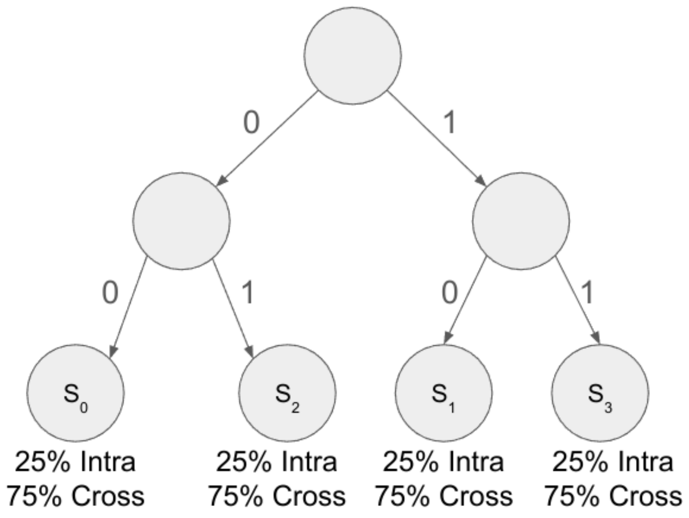


Figure 15: Distribution of generated transaction for 4 Shards

Considering 4 shards (Fig. 15), each shard generating transactions having destination account uniformly from all shards. In each block full of transactions (20000 Txns), each shard will be sending 5000 Intra-Shard Txns and 15000 Cross-Shard Transactions with source shard itself. These full blocks can be sent consecutively for 4 rounds until the shards have to process the blocks coming from other shards. So, after 4 rounds, the next 4 rounds it will process blocks of 5000 Intra-shard Txns and 15000 Txns with source as other shard.

So for 8 rounds and 4 shards, a shard will produce 40000 intra-shard transactions and 60000 cross-shard transactions.

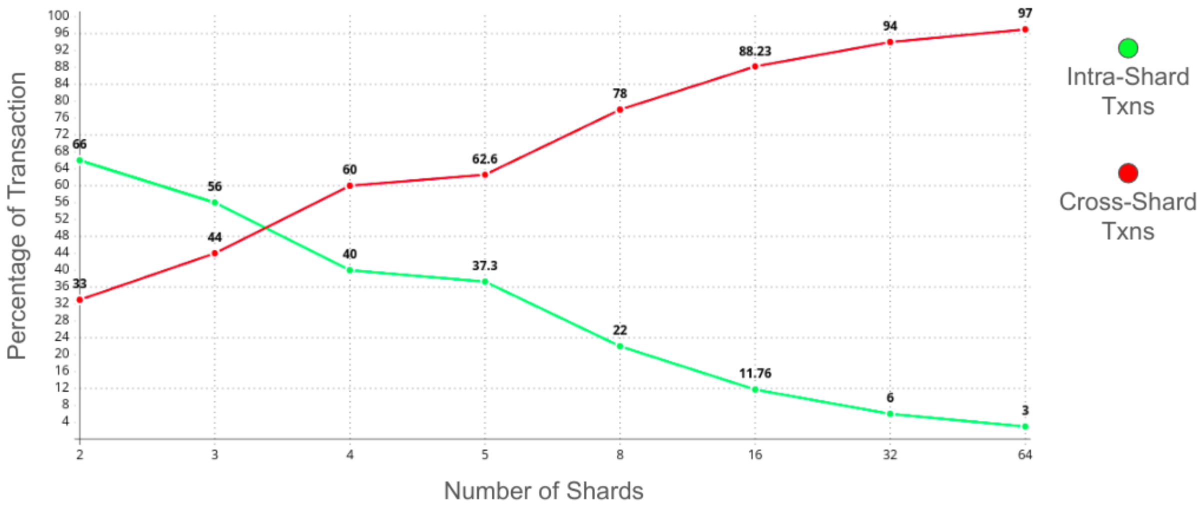


Figure 16: Observation of the Experiment in Type Percentage Vs. Number of Shards

The experiment done has shown the correctness (Fig. 16) of the above hypothesis. With 64 shards, the cross-shard transaction is reaching almost 97% of the total transactions in the network.

5.6 Minimising Cross-Shard Transactions

In a state-sharded blockchain system, two accounts with addresses mapped to different shards may frequently engage in transactions. However, processing transactions between these accounts across shards incurs latency compared to intra-shard transactions. One solution proposed to mitigate this latency is to relocate one of the accounts to the same shard as the other, thereby facilitating faster transaction processing.

However, implementing this solution introduces overheads, including identifying similar transactions between the accounts, creating new wallets for the relocated account, and transferring the entire balance to the primary account. This process necessitates careful consideration of transactional integrity and security. Furthermore, managing the relocation process efficiently requires robust mechanisms for identifying and reconciling transactions across shards.

It was observed that in a scenario with 4 shards and 1000 accounts, with blocks fully occupied by transactions, the proportion of cross-shard transactions could be reduced to approximately 48% from 60%. Furthermore, accounts engaging in more than 3 transactions within a single round benefitted significantly from the proposed optimization technique. This observation underscores the effectiveness of intra-shard account relocation in minimizing cross-shard transaction overheads and improving overall network efficiency. By strategically managing account placement and transaction routing, blockchain systems can optimize throughput and transaction processing speed, enhancing user experience and network scalability.

6 Conclusion

MultiversX stands as a pioneering public blockchain, utilizing the innovative Secure Proof of Stake algorithm within a genuine state-sharded architecture to achieve throughput comparable to VISA and confirmation times in seconds. We analyzed the stated performance of the network to find out the improvement and vulnerabilities of the network in compared to other scalability solutions and pre-existing network like Bitcoin. We also stated some solutions to how these vulnerabilities can be overcome with simple methods along with the overhead they create. Looking ahead, potential avenues for future research offer promising opportunities to push the boundaries of scalability and efficiency in blockchain systems. By fostering collaboration and innovation in these areas, we can propel the evolution of sharding methodologies and pave the way for a new era of decentralized applications and services.

References

[1] Multiversx whitepaper.

[2] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OsDI*, volume 99, 1999.

[3] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper*, 19(1), August 2012.

[4] Paradigm Research. Elrond: High-throughput public blockchain with adaptive state sharding, 2019.

[5] A. Secure. The zilliqa project: A secure, scalable blockchain platform. 2018.

[6] Alex Skidanov and Illia Polosukhin. Nightshade: Near protocol sharding design. 2019.

[7] Dmitry Tanana. Avalanche blockchain protocol for distributed computing security. In *2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE, 2019.

[8] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, 21(2327):4662, 2016.