3

Scalable Blockchain

Bitcoin is the first decentralized system that provides users with virtual currency to buy and sell services and goods. Since then, Bitcoin has autonomously evolved because of its properties of resiliency, integrity, and transparency. However, Bitcoin suffers from scalability issues, which restrict its wider adoption. Scalability in blockchain refers to the capability of a blockchain network to handle a growing amount of transactions or workloads efficiently. It's an essential aspect that determines how well a blockchain can grow and adapt to increased demands without compromising performance and security [135]. Scalability is a key area of research and development within the blockchain community, as improving it is crucial for the widespread adoption and practical use of blockchain technology in various applications beyond cryptocurrencies [23]. Over the past years, the Bitcoin community has witnessed major divisions (forks) over how Bitcoin should scale [136]. Consequently, it is a simple tradeoff between scalability and centralization: as the size of the blockchain grows, few devices will have the capacity to store and process the complete blockchain, which leads the network to become more centralized [137]. Scalability has emerged as a significant hurdle as the technology matures and its adoption grows. To address these challenges, sharding, a concept traditionally used in databases, has been adapted to blockchain technology as a promising solution. Sharding partitions the blockchain network into smaller parts called shards, where each shard can process its own transactions. It increases the network's throughput and scalability by allowing multiple transactions to be processed simultaneously. It also enhances decentralization by distributing the computational load across different nodes. Despite the advancements in sharding made by state-of-the-art schemes like ELASTICO [138], OmniLedger [74], RapidChain [78], and Chainspace [80], they encounter limitations such as high cross-shard transaction latency, low parallelism, centralized reconfiguration, and vulnerability to 1% attack, double spending attack, and majority attack. To address these challenges, we proposed two solutions, namely, *BlockTree* and UnoShard. In this chapter, we first explore the challenges in achieving scalability, sharding in blockchain, and the proposed *BlockTree* and *UnoShard* solution for scalable blockchain.

3.1 CHALLENGES IN ACHIEVING SCALABILITY

There are three major challenges in achieving scalability in blockchain: satisfying blockchain trilemma, synchronization and high latency, and performance concerns. Below, we have defined these challenges in detail.

- **Blockchain Trilemma:** Scalability, security, and decentralization are known as the blockchain trilemma. The significance of this trilemma is that enhancing one aspect leads to compromises in other aspects. For instance, enhancing the scalability by adding more transactions per block can compromise security and decentralization. It is a tradeoff between scalability and centralization: as the size of the blockchain grows, few devices will have the capacity to process the complete blockchain, which leads the network to become more centralized [74].
- **Synchronization and High Latency:** Synchronization involves ensuring that all nodes in the blockchain network have the same, most current version of the ledger, whereas latency refers to the time it takes for a transaction to be confirmed and reflected across all nodes, in the network. Ensuring the ledger synchronization across all the nodes in the network with high latency or consensus delay is a big challenge in achieving scalability. Lowering the consensus delay depends on the effective design of the consensus algorithm. Therefore, designing an effective consensus algorithm to ensure synchronization with low latency is highly important [139].
- **Performance Concerns:** Ability to process a high volume of transactions within a short period is one of the biggest challenges in blockchain scalability. Conventional blockchains like Bitcoin and Ethereum show very limited performance, such as 4-7 transactions per second and 30-40 transactions per second, respectively, which is significantly lower compared with centralized payment systems like Visa and Mastercard. Thus, scaling the blockchain network to meet global transaction demands without compromising security and decentralization is a big challenge. Apart from transaction processing capacity, the high computational and communication overhead also creates a hurdle in scaling the blockchain performance [21].

3.2 BLOCKCHAIN SHARDING

Sharding is a widely adopted method to achieve high scalability in blockchain. In sharding, the blockchain network is split into smaller networks known as shards, where each shard processes its transactions independently. This means transactions can be processed in parallel, significantly increasing throughput [26]. As

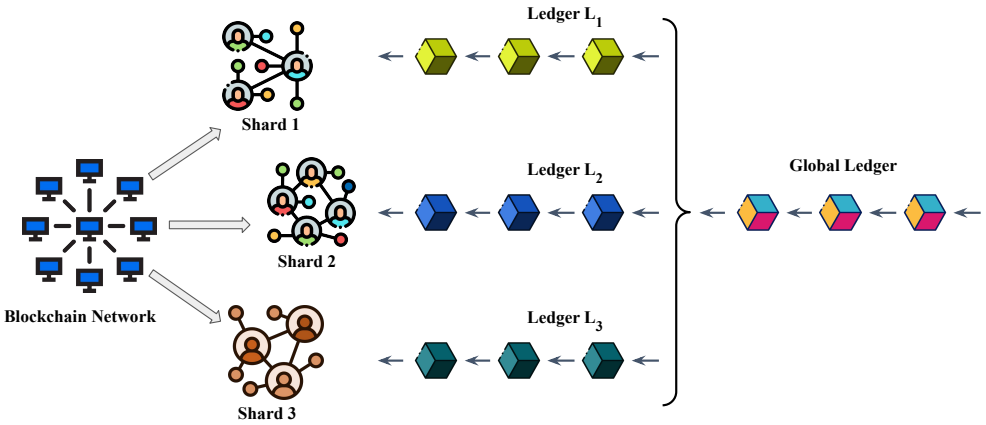


Figure 3.1: An example of a sharded blockchain with three shards. Each shard has its own local ledger L_1, L_2, L_3, and one global ledger.

shown in Figure 3.1, in a sharded blockchain, nodes are allocated into different shards. Each shard maintains its own copy of the ledger (local ledger), i.e., L_1, L_2, L_3 that contains transactions. Apart from these local ledgers, one global ledger is shared with all the shards, which contains all the transactions shared amongst the shards. A transaction may involve a single shard or multiple shards. If a transaction involves a single shard, it is called intra-shard communication, and if it involves multiple shards, it is called a cross-shard transaction. To execute and confirm the cross-shard transaction, nodes in multiple shards communicate with each other.

3.3 CONTRIBUTIONS

While blockchain sharding is an intuitive principle, designing a blockchain sharding protocol is challenging. Sharding blockchain introduces consistency issues across shards. In addition to this, cross-shard communication introduces additional delay, which further reduces the system throughput. In the first chapter, we have listed the key issues of blockchain sharding that need to be solved to enhance the system's performance. To solve these issues, BlockTree opted for two approaches: first, minimizing the constraint that each transaction is broadcasted and processed by the entire network, and second, restructuring the relationship between blocks from a chain to a graph. On the other hand, the UnoShard works in three different phases, i.e., fair shard allocation, intra-shard consensus, and cross-shard consensus. The shard allocation phase leverages the Delegated Proof of Stake (DPoS) consensus scheme, where the number of shards is equivalent to the number of delegates that have been selected. The intra-shard consensus phase used the concept of a trust-based Practical Byzantine Fault Tolerance (PBFT) scheme, where the trust is calculated based on the direct interaction between the nodes and the feedback from the nodes. The block generated by each shard is named Local Block (*LB*). The last phase is the most critical in every sharding scheme: cross-shard consensus. The cross-shard consensus gathers all the local blocks and generates a Global Block (*GB*) for the global ledger, which is shared amongst all the peers in the network. Following are the major contributions of BlockTree and Unoshard.

- **Restructured Blockchain:** Proposed scheme, BlockTree, a restructured blockchain design from linear to nonlinear structure that can process multiple transactions in parallel and serves the purpose of sharding transactions.
- **Consensus Algorithm:** Apart from restructuring blockchain, we proposed a novel leader selection and consensus algorithm. To mark the new leader selection in the shard, we have created two different types of blocks, i.e., data block and leader block. The leader block is created when a new leader is selected, and that leader creates the data blocks, which contain the new set of transactions.
- **Cross-Shard consensus:** In UnoShard, we have proposed a novel cross-shard consensus mechanism, which creates the global ledger in cooperation with all the shards to ensure zero possibility of double-spending and fault tolerance. To establish cooperation among all the shards, cross-shard consensus used a novel concept called vote chain, which ensured the confirmation of all the transactions processed by each shard and bunched in the form of a local block.
- **Trusted Intra-Shard Consensus:** The proposed UnoShard leverages the trust-based PBFT consensus mechanism in each shard to confirm the transactions and generation of local blocks. The trust value of each node is computed based on the comprehensive evaluation of the node's historical per-

formance and behavior within the network. This includes factors like the node's accuracy in transaction processing, consistency in participating in consensus activities, uptime, and overall cooperative behavior. The node which has the highest trust value is called the leader node.

- **Shard Allocation:** In addition to cross-shard and intra-shard consensus methods, we designed the shard allocation method based on the delegated proof of stake (DPoS) method. The number of shards is equal to the number of delegated in the selected in the first phase of the UnoShard scheme.
- **Performance and Security Analysis:** We have conducted a comparative comprehensive performance analysis of BlockTree based on the parameters, throughput, consensus delay, transaction frequency, fairness, and leader selection time. We have also performed a security analysis of the BlockTree and UnoShard based on fork prevention, scalability, double-spending, 1% attack or shard takeover attack, and fault tolerance prevention. Both analysis shows the superiority of BlockTree and UnoShard over the state-of-the-art schemes.

3.4 BLOCKTREE: A NONLINEAR STRUCTURED, SCALABLE DISTRIBUTED LEDGER

The BlockTree is a blockchain protocol that offers better and improved consensus delay and throughput than protocols such as Bitcoin-NG and Algorand without losing the security. This protocol improved in two ways: the first structure of the blockchain and the second consensus algorithm. The structure of a BlockTree is no longer a chain structure but follows a tree structure (m-way tree structure). The protocol splits the network based on the number of miners in the network. Here, "m" defines the upper limit of the number of miners in the network. If, at a particular time, the number of miners increases more than m (i.e., $number_of_node > m$), it splits the network from that instant, i.e., a new shard is created. This process is repeated in every group, and it grows like an m-way tree. As soon as the network is split and a new shard is created, the current blockchain also gets forked, and the newly created shard starts mining and adding the block into this new branch of the chain. This way, we can employ the parallel execution of transactions, increasing the throughput and decreasing the consensus delay. Figure 3.2 is the structure of the BlockTree. Every miner node keeps track of a number of miners in the group with parameters $number_of_node$, and *Algorithm* 1 keeps working continuously and checks $number_of_node > m$ for creating a shard. This parameter is updated every time a new miner is added to the network. Furthermore, in each newly created shard, a group leader is selected periodically, i.e., there is a fixed time interval between selecting any two group leaders. *Algorithm* 2 described the process of leader selection. The leader is responsible for the serialization of transactions generated in that interval. The protocol generates two types of blocks: leader block and data block. The leader block is added when a new leader is selected, and after this leader block, the data block is added based on the consensus algorithm. The leader block contains no transactions, while the data block contains the transactions. The leader block acts as a mark of a leader, i.e., it defines which miners add the data blocks. If it is found that some transactions are being forged or modified, then the leader node is marked as a bad miner so that it cannot take part in the next group leader selection. Thus, the probability of the current leader getting selected is minimal so that all the miners of the group can get an equal opportunity to become a leader.

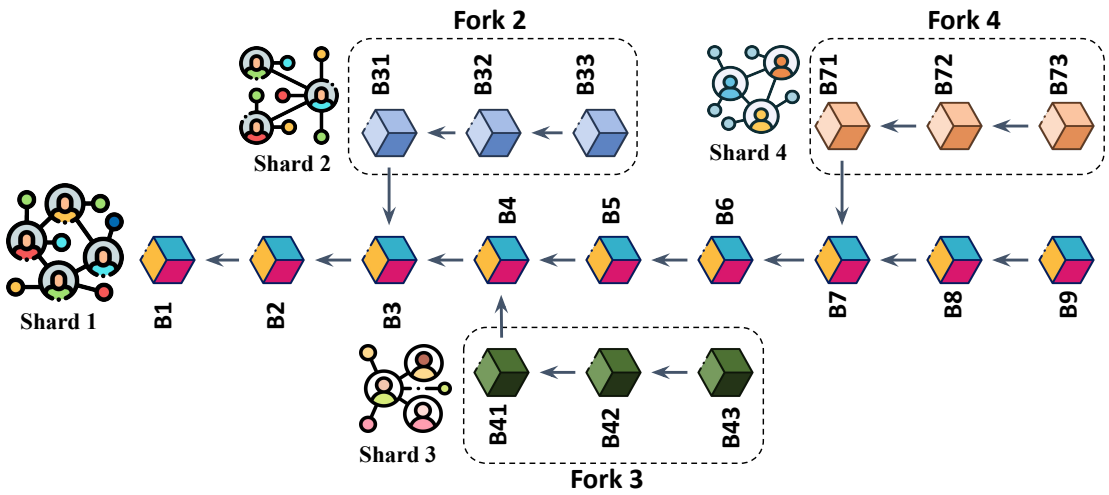


Figure 3.2: Struture of BlockTree

3.4.1 Sharding in BlockTree

As we have mentioned in the previous section, when the condition $number_of_node > m$ (i.e., the count of miners is greater than the set limit m) is true, a shard is created in the blockchain. This newly created shard included the recently joined miners. Therefore, any miner who joins the network must keep the time of joining the network. Every leader block also includes the time of joining the network for miners. In each shard, for generating and verifying the data blocks, BlockTree uses the *Algorithm* 3. After verification, the leader adds the data blocks to the chain. Since we have employed the consensus algorithm for leader selection and the data block verification, the BlockTree does not suffer from illegitimate fork generation. Due to the inclusion of two consensus algorithms, BlockTree also does not suffer from the double-spending problem. *Algorithm* 1 describes how a shard is created in the BlockTree. This algorithm continuously works at all the shards of the blockchain network and checks how many miners have been added. *Algorithm* 1 works as follow:

1. This is a continuous process until the system functions properly. Since it is a public blockchain, any node can join and leave the network. Any node who wants to be a miner is added into the current group of miners as current_miner_group.add (new_node) after checking if $number_of_node < m$ is true or not. If it is true, then *new_node* is added, and all the nodes follow the *Algorithm* 2 for the leader selection and *Algorithm* 3 for the data block creation and verification.

2. But if the condition $number_of_node < m$ is false, the current shard has more miners than the limit m so that the shard is split with minimum size (size varies from system to system). This newly created shard contains the recently joined nodes.

3.4.2 Transaction Sharding

The BlockTree is a sharding-based distributed ledger protocol. The structure of the scheme is modified so that, instead of all the miners mining one block, we have divided the miners into shards so that many groups of miners (shards) can mine the blocks parallelly. This change in structure and sharding of transactions

Algorithm 1: Algorithm for Shard Creation

Input: New_Node (Node's Public Key and Joining Time)
Output: Shard Creation
Function `createShards(`*nodes*`):`

```
while true do
    if number_of_node < m then
        current_miner_group.add(new_node);
        // Follow the Algorithm 2 to select the leader and Algorithm 3 to
           create and verify the data blocks
    else
        create_shard(minimum_Group_Size);
        current_miner_group.shard(all_recent_new_node)
```

improved the throughput and the consensus delay. One more parameter that is enhanced due to this sharding is the scaling of the blockchain network, i.e., BlockTree is the highly scalable protocol compared to the state-of-the-art schemes. According to the scheme, anyone can join the network, and this does not cause any overhead on the network because as soon as the number of miners increases from the defined limit, the network is split, and a new shard is created that mines the block on the forked chain. Hence, due to the structural change, the performance is improved in terms of throughput and consensus delay, and the scalability issue is also resolved.

3.4.3 Leader Selection, Leader Block, and Data Block

The leader blocks are used to mark the leader selection. The leader block is similar to the Bitcoin block, except that it does not contain any transaction value. Every leader block must contain miner ID, block number, timestamp, coinbase transaction, and nonce value. A leader block is valid if its hash is smaller than the threshold value. It also contains the leader's public key because this public key is used to verify all the subsequent data blocks until the new leader block is added. The time is equally divided among all the leader blocks to maintain a uniform rate of data block generation. Leader selection in BlockTree is different from Bitcoin-NG in that the probability of selecting the current node as the leader node is minimal, i.e., unlike Bitcoin-NG, BlockTree is fair, where all nodes have an equal chance to become a leader. The two parameters of the algorithm, status and age, are essential for leader selection to be fairer. The status tells about the behavior of the node because if any node is found malicious, its status is set to bad, and the age parameter tells whether it is a current leader or not. *Algorithm* 2 for leader selection works as follows:

1. This process is followed in every group of the BlockTree blockchain network. All the nodes participate in the leader selection process. For this, every node should have the timestamp T, previous hash block (PBH), which is calculated by the function $getHash()$. Though every node has a copy of the current blockchain, using the $getHash()$ function gets the last block's hash. Along with the timestamp and hash of the previous block, it set a *nonce* value to 0.

2. A threshold hash value $thresholdValue$ is set, and every node starts calculating the target hash value $targetHash$, which should be less than the $thresholdValue$. Node uses the function $calculateHash()$ to calculate the hash value. Calculating the $targetHash$ value also checks the status and age of the

node for fair selection and eliminates malicious nodes. The node with the least age has the highest preference for leader selection.

3. After the leader node selection, a leader block (*leaderBlock*) is created using block creation function *createBlock*() with parameters, T, *minerID*, *PBH*, *status*, *age*, *block_number*, *targetHash* and appended to the existing ledger.

Algorithm 2: Leader Selection Algorithm

Input: Node[i], i=1,2,3,.....
Output: Leader Node
Function `leaderSelection(`*nodes*`):`

```
while LeaderIsSelected == 0 do
    for each node[i] in group do
        T = generate_unique_timestamp()
        nonce = 0
        targetHash = calculateHash(T, nonce, previousBlockHash, node[i].ID)
        while targetHash >= thresholdValue do
            nonce+ = 1
            targetHash = calculateHash(T, nonce, previousBlockHash, node[i].ID)
        if status! = bad&age(node[i]) < age of all the nodes then
            node[i] = leader
            LeaderIsSelected = 1
            broadcastLeaderSelection(node[i].ID)
if node[i] == leader then
    leaderBlock = createBlock(T, PBH, minerID, status, ageblock_number, targetHash);
    blockchain.append(leaderBlock);
```

There are two types of nodes in the blockchain network: one that generates the transactions, usually called ordinary nodes, and the second type of node, called miner node, which does the mining work. One of the miner nodes is elected as the leader node following the leader selection algorithm, and the leader block is generated to mark the entry into the blockchain ledger. A leader is allowed to generate data blocks until the next leader is selected. The time between the two leader blocks is the interval for data block generation. The size of the data block is predefined. The data block content is ledger entries, Mrt, T, and PBH. The leader's private key signs the data blocks, which are verified by the other miners (peer nodes). Due to timestamp and status parameters, the malicious data block is prevented from being added to the chain. Figure 3.3 shows the arrangement of a data block and a leader block. The leader node collects all the recently added transactions from the transaction pool. Compute the Mrt and then compute the data block, including T, PBH, and *nonce* value. After creating the data block, the leader signs it and sends it to other nodes in the shard along with the *nonce* to verify it. Each non-leader node independently verifies the block and then broadcasts a "valid block" message to all nodes, including both the leader and the other non-leader nodes. To verify the data block, each non-leader node calculates the Merkle root and checks whether the received block and calculated block are the same or not. If both are the same, then set the block status as valid; otherwise, set it as invalid and set the status of the leader node as bad so that it will not be selected again as a leader node. By broadcasting the "valid block" messages to all nodes, the protocol guarantees that the consensus process is transparent and that all nodes can participate in verifying and accepting the

leader block. This approach prevents any single point of failure and ensures that the blockchain remains secure and decentralized. If the received valid block status is more than $2K+1$, the block is added to the chain; otherwise, it is discarded (where K is the number of faulty nodes). *Algorithm* 3 shows the generation and verification of data blocks. A detailed description of the *Algorithm* 3 is as follows:

1. All the ordinary nodes connected to the miner nodes generate the transactions and put them in the transaction pool (TP). All the miner nodes share the transaction pool. The leader node collects all the transactions from the TP and then, after calling the function $createHashTree$, calculates the Mrt and passes these transactions.

2. After calculating the Mrt, it also calculates PBH using the function $getHash$ and recod the current timestamp T.

3. After calculating Mrt, PBH and getting timestamp T and $nonce$, it creates a data block somewhat different from the leader block. The data block is created using the function $createBlock()$ with the parameters described above. After creating the block, the leader node signs it using its private key and broadcasts it into the network for verification.

4. The non-leader nodes receive the data block and verify using the public key of the leader. These nodes also calculate the Mrt from the TP, create the data block, and check whether the received block matches with the regenerated block. If both match, then set the $block_status$ as $valid$; otherwise, set it to $invalid$ and set the node status $node.status$ as bad so that in the next leader selection, it will not be selected and considered as the malicious node.

5. If the leader node receives at least $2K+1$ $valid$ response, it adds a data block into the blockchain ledger.

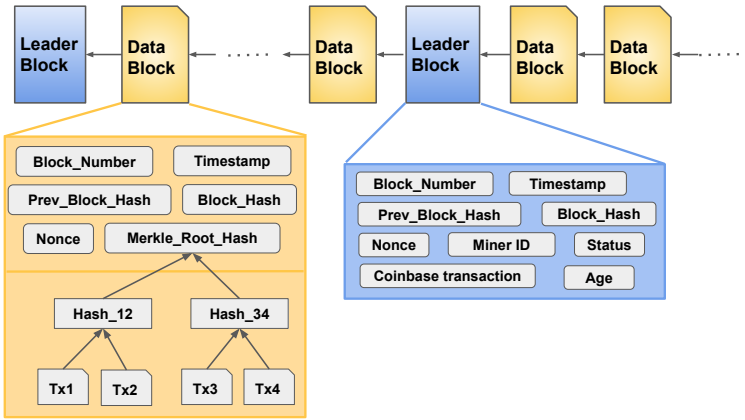


Figure 3.3: Types of blocks and their arrangement

Total Ordering of Transactions

In BlockTree, the total ordering of transactions is achieved through a combination of linear ordering within each shard and timestamp-based ordering across the entire network. Below is a detailed description of how the ordering of transactions is achieved in BlockTree.

Algorithm 3: Data Block Verification Algorithm

Input: Transactions: $Tr_1, Tr_2, ..., Tr_n$
Output: Data_Block
Function `dataBlockVerification(`*nodes*`):`

```
TP.append = new_transaction;
if node == leader then
    Mrt = createHashTree(TP.getAllTransactions);
    T = Timestamp;
    nonce = random_number;
    PBH = getHash(previous_block);
    BH = getHash(Mrt, T, PBH);
    block = createBlock(Mrt, T, PBH, BH, nonce, glocalTimestamp, node[i].ID);
    sign_block(block);
if node != leader then
    receive_block(block, nonce);
    Mrt = createHashTree(TP.getAllTransactions);
    block* = createBlock(Mrt, T, PBH, BH, nonce, glocalTimestamp, node[i].ID);
    if block == block* then
        block_status(valid);
    else
        block_status(not_valid);
        node.status = bad;
if (count(block_status == valid) ≥ 2K + 1) then
    blockchain.append(block);
    synchronize_timestamps(block);
```

1. **Linear Ordering within Shards:** Each shard in the BlockTree structure functions like a traditional blockchain, maintaining a linear sequence of blocks. Within a shard, the leader node generates a leader block to mark the beginning of its term, followed by a sequence of data blocks that contain transactions. These blocks are linked sequentially, referencing the previous block's hash, thereby creating a linear chain of blocks. This method ensures that transactions within a shard are processed in a clear, sequential order, similar to the traditional blockchain approach used in Bitcoin. Transactions are included in the blocks in the order they are received and verified by the leader. This ordering ensures that all transactions within a shard are added to the ledger in a consistent and linear manner. The process of adding transactions to blocks, verifying them, and appending the blocks to the chain maintains this linear order. As each shard operates independently, the integrity and order of transactions within a shard are preserved, providing a clear and unambiguous sequence of events for all transactions processed within that shard.

2. **Timestamp-Based Ordering Across Shards:** To maintain synchronization and achieve a total order of transactions across different shards, BlockTree employs a timestamp-based ordering mechanism. Each transaction is assigned a global timestamp when it is created, reflecting the time at which the transaction was first generated and broadcast to the network. These timestamps are crucial for ordering transactions across shards, ensuring that even though transactions are processed in parallel

within different shards, they can be consistently ordered across the entire network. Periodic synchronization points are established across shards, during which leaders communicate and reconcile the order of transactions based on their timestamps. When transactions are verified and added to the blocks, their global timestamps are used to order them relative to transactions in other shards. This timestamp-based approach ensures that the entire network maintains a coherent and synchronized order of transactions, despite the parallel processing capabilities provided by the m-way tree structure. By leveraging timestamps, BlockTree ensures that all transactions are integrated into a single, unified ledger, providing a comprehensive view of the transaction history across all shards.

3.4.4 Remuneration Model

In the case of Bitcoin, a miner is compensated for the efforts to mine the block in the form of some bitcoins. Similarly, BlockTree also has remuneration for mining the block to motivate the mining. A leader is compensated for adding the data blocks into the chain. The remuneration is compensated based on the data blocks added by leaders, i.e., the addition of each data block carries a fee, which would be called a transaction fees. BlockTree has a transaction fee as a remuneration. If the number of data blocks added by the leader node is 10, each data block's transaction fee depends on how many transactions it contains. For example, if every data block contains 200 transactions on average, ten blocks would contain 2000 transactions. Hence, the general equation for transaction fees equals to

$$Total_fee = TF \sum_{i=1}^{n} DB_i \tag{3.1}$$

where DB_i is the number of transactions in i^{th} data block and TF is a transaction fee of each transactions.

3.5 UNOSHARD: HARMONIZING SCALABILITY AND SECURITY

BlockTree introduced a novel approach to blockchain scalability by restructuring the blockchain into an m-way tree, enhancing parallel transaction processing. However, despite its innovations, BlockTree faces challenges related to performance, double-spending, and certain security vulnerabilities. To address these issues, we developed UnoShard, a more advanced sharding technique. Unlike BlockTree, UnoShard employs a trust-based PBFT consensus mechanism within each shard and introduces a novel cross-shard consensus mechanism called VoteChain. These enhancements significantly improve performance, prevent double-spending, and strengthen security against various attacks. Therefore, UnoShard offers a more robust solution to the challenges of BlockTree and other state-of-the-art schemes for blockchain scalability and security.

UnoShard is a novel approach that harmonizes scalability and security using the sharding technique. The UnoShard works in three different phases, i.e., fair shard allocation, intra-shard consensus, and cross-shard consensus. The shard allocation phase leverages the DPoS consensus scheme, where the number of shards is equivalent to the number of delegates that have been selected. The intra-shard consensus phase used a trust-based PBFT scheme, where the trust is calculated based on the direct interaction between the nodes and the feedback from the nodes. The block generated by each shard is named Local Block (*LB*). The last phase is the most critical in every sharding scheme: cross-shard consensus. The cross-shard consensus gathers all the local blocks and generates a Global Block (*GB*) for the global ledger, which is shared amongst all the

peers in the network. To combine all local blocks and verify them, UnoShard uses the concept of Vote Chain (*VC*) and uses the Directed Acyclic Graph (DAG) to construct the root hash of all the transactions instead of a merkle root tree for *GB*. To accomplish these tasks, all the shards select a Shard Coordinator (*SC*), usually called an inter-shard coordinator. The *SC* is selected based on the major three critical elements, i.e., accumulative reputation score (ARS), decaying reputation score (DRS), and performance metrics-based reputation score (PMRS). These comprehensive approaches ensure that UnoShard effectively addresses the complexities of sharding in blockchain networks. Further, it is important to note here that UnoShard leverages the UTXO (Unspent Transaction Output) model as the foundation for its transaction processing. The UTXO model, widely used in blockchain networks like Bitcoin, maintains the state of the blockchain through unspent transaction outputs. This model enables efficient transaction validation and supports the scalability and security enhancements that UnoShard aims to achieve.

UnoShard addresses the issues of 1% attack, double spending attack, and majority attack, thus enhancing the overall security and robustness of the blockchain network. By overcoming the limitations of existing sharding schemes, UnoShard presents itself as a robust, scalable, and secure framework, paving the way for the next generation of blockchain applications. Details of these phases are given in subsequent sections.

3.5.1 Shard Allocation

The *Algorithm* 4 outlines the process of distributing nodes between shards in a blockchain network. It begins by initializing arrays for vote weights and delegate votes based on the total number of stakeholders and delegates. The functions of the *algorithm* include *calculatingVoteWeights*, *electingDelegates*, and *nodeAllocation*. There are five key steps based on which the shard allocation is performed in UnoShard. Each of these key steps plays a critical and significant role in ensuring fairness, efficiency, and decentralization in the network. The key steps are as follows.

Computing Weighted Vote

The vote of each stakeholder is weighted based on the stakes they put in to be a part of the active working node in the network. The following equation (4) is followed to compute the weight of every stakeholder's vote.

$$W_i = \frac{s_i}{\sum_{k=1}^{M} s_k} \tag{3.2}$$

where W_i is the weight of the stakeholder's i vote, S_i is the stake of the corresponding stakeholder, and M is the total stakeholders. It is important to note that the stakeholder is nothing more than the consensus nodes in the blockchain network. It is obvious that the stakeholders with the highest stakes in the network have a higher proportional impact on the delegate selection process. The function *calculatingVoteWeights* in *Algorithm* 4 performs this computation.

Delegate Selection and Shard Formation

The second step in the allocation of the shards is the selection of the delegates after computing the weight of the votes of the stakeholders. Let us say V_d is the sum of the total weighted votes for each delegate. Then, $D \in D_{Max}$ is the number of top nodes with the highest V_d selected as delegates. This process ensures an unbiased and fair selection of delegates, which is very important for any shard-based blockchain system.

The third step is the formation of shards. In this step, each delegate $D \in D_{Max}$ is assigned to a shard. This assigned delegate node is also selected as the block proposer or shard leader for the first round of consensus in the intra-shard consensus process. This second and third step is illustrated in *Algorithm* 4 in function *electingDelegates*.

Nodes Allocation to Shards

After assigning one delegate to each shard, the fourth step is to assign a number of nodes to each shard. We have used the following equation (3.3) to allocate the number of nodes to each shard evenly.

$$NNS = \frac{N}{D} \tag{3.3}$$

NNS denotes the number of nodes in each shard, N is the total number of nodes in the network, and D is the total number of delegates selected in the first step. Thus, each shard contains *NNS* nodes to maintain the true balance and efficiency of the network operation. The function *nodeAllocation* is dedicated to this fourth step of shard allocation.

Periodic Delegate Re-election and Re-assignment

New delegates are re-elected and re-assigned after every 100^{th} *GB* in the *GL*. It is an important step to prevent the system from becoming biased and unfair in transaction processing. It also plays an important role in preventing the shard takeover attack. Thus, this step is crucial for the efficient and fair functioning of the blockchain system. To do this, all the functions of *Algorithm* 4 are called again in the correct sequence, i.e., first, *calculatingVoteWeights*, second *electingDelegates*, and third *nodeAllocation*.

3.5.2 Intra-Shard Consensus

This is the second phase of UnoShard. After the successful completion of the shard allocation phase, the intra-shard consensus takes place. The intra-shard consensus is essentially a trust-based PBFT consensus *algorithm*, where the block proposer is selected based on their trust value. As a novel contribution, we have derived an efficient trust computation scheme for the nodes in the shard. This trust computation is important for attack-free transaction processing and verification. Trust assessment among the nodes in the shard is crucial for maintaining a consistent and stable view of the shard. The intra-shard consensus process is again divided into five steps, namely, initial trust assignment, trust score computation, trust score normalization, and PBFT consensus process.

As stated in the Shard Allocation section, during the 'delegate selection and shard formation' step, the top nodes with the highest weight are selected as delegates and are assigned to each shard. This assigned delegate node is also selected as the shard leader for the first round of the consensus. Therefore, in this step, the initial trust score $N(T)_i$ of all the nodes is set to *zero* except the assigned delegate node. That is, let's say N_k is the delegate node, then the trust score $N(T)_k$ of this node is set to 1, and the trust score of other nodes is set to 0. This can be represented as follows:

$$N(T)_i = \begin{cases} 1 & \textit{if } i = k \textit{ is the delegate node} \\ 0 & \textit{otherwise} \end{cases} \tag{3.4}$$

Algorithm 4: Algorithm for Shard Allocation

Input: N (Total number of nodes), D (Total number of delegates), s (Array of stakes, size M), M (Total number of stakeholders)
Output: shardAssignments (Array of nodes assigned to each shard)
Initialize array W of size M for vote weights;
Initialize array V of size D for delegate votes;
Function `calculatingVoteWeights()`:
    $totalStake \leftarrow \text{sum}(s[i] \text{ for } i = 1 \text{ to } M)$;
    for $i = 1$ *to* M **do**
        $W_i = \frac{s_i}{\sum_{k=1}^{M} s_k}$;
Function `electingDelegates()`:
    for *each delegate d* **do**
        $V[d] \leftarrow \text{sum}(W[i] \times$ *votes for d from stakeholder i*$)$;
    selectedDelegates $\leftarrow$ *top D indices in V with highest values*;
Function `nodeAllocation()`:
    nodesPerShard $\leftarrow N/D$;
    currentNode $\leftarrow 1$;
    for *each shard in selectedDelegates* **do**
        shardAssignments$[$*shard*$]$ $\leftarrow$ *list of nodes from currentNode to currentNode + nodesPerShard* - 1;
        currentNode $\leftarrow$ *currentNode* + *nodesPerShard*;
Call `calculatingVoteWeights()`;
Call `electingDelegates()`;
Call `nodeAllocation()`;

After the first round of the consensus process, the trust score of all the nodes is updated. The trust score update depends on two important parameters: 1) the node's historical performance and 2) behavior within the network. These parameters further include factors like accuracy in transaction processing from node and network inclusion time, i.e., up-time in the network. The trust score update process ensures that nodes within the shard maintain a consistent and reliable view of each other's trustworthiness. While it is not required for all honest nodes to have identical trust metrics, the continuous feedback and interaction mechanism lead to a convergence of trust scores over time. This dynamic and decentralized approach ensures that the trust metrics used in the consensus process accurately reflect the overall behavior and reliability of the nodes, thus maintaining the security and efficiency of the blockchain network. Mathematically, this is represented as:

$$N(T)_i^{(new)} = \sum_j N(BV)_{ji} * N(T)_j^{(old)} \tag{3.5}$$

Here, $N(BV)_{ji}$ denotes the behavior value given by node N_j to node N_i, reflecting node N_j's assessment of node N_i's reliability. The $N(T)_j^{(old)}$ is the previous trust score of node N_j. This calculation aggregates trust feedback from multiple nodes to revise $N(T)_i^{(new)}$. The current shard leader prepares the 2X2 reputation

matrix for the C_{ji} as follows:

$$\begin{bmatrix} N(BV)_{1,j}^{(k)} & , & N(BV)_{1}^{(k-1)} \\ N(BV)_{2,j}^{(k)} & , & N(BV)_{2}^{(k-1)} \\ & \vdots & \\ N(BV)_{i,j}^{(k)} & , & N(BV)_{i}^{(k-1)} \end{bmatrix}$$

Here, $N(BV)_{1,j}^{(k)}$ depicts the behavior value of N_j on N_1 in the k^{th} round of consensus, and $N(BV)_{i}^{(k-1)}$ is the behavior value of N_i in the $(k-1)^{th}$ consensus round. To achieve effectiveness and accuracy in trust score computation, the trust score values computed above in equation (3.5) are normalized. Normalization involves adjusting the computed trust score on the 0-1 scale, where 0 means the lowest trust score and 1 means the highest trust score. This normalization step is essential for removing any possibility of biases from historical performance and behavior analysis and preserving the scoring consistency.

The node with the highest trust score is selected as the shard leader for the next consensus round. In the last step of the intra-shard consensus, the shard leader collects transactions from the global transaction pool, with each local block capable of including a maximum of 500-800 transactions, and starts the process of generating *LB*. This constraint ensures that each shard's transaction processing load remains manageable and efficient. To generate the *LB*, the intra-shard follows the typical PBFT process after selecting the shard leader. The process of consensus based on PBFT is shown in Figure 3.4. The role of the trust-based shard leader as the block proposer is pivotal in ensuring the security and integrity of the *LB* generation process. Leveraging the high trust for shard leader selection maintains a reliable and efficient consensus among the nodes in the shard. This ensures that the transactions in the shard are processed by the most trusted nodes, which further enhances the robustness of the shard network. The output of this phase is represented as LB_{S_i}, where S_i denotes the i^{th} shard number.

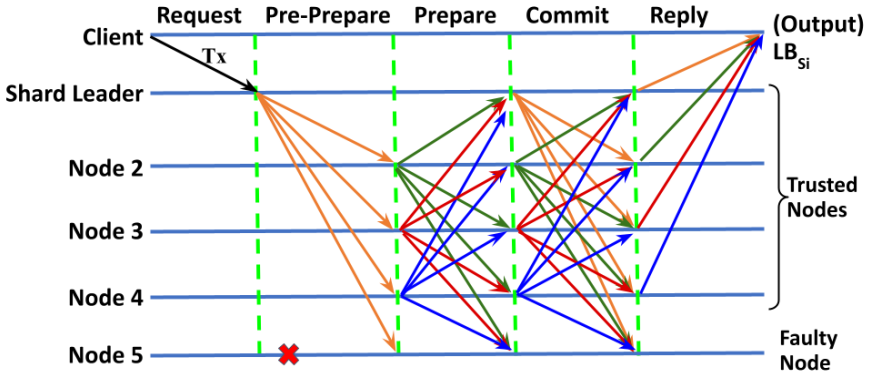


Figure 3.4: Intra-shard consensus process of UnoShard based on PBFT

3.5.3 Cross-Shard Consensus

Cross-shard consensus is the most challenging thing in any sharding scheme of the blockchain network. Therefore, it is also the last and most crucial phase of the UnoShard. This step basically, collects all the local blocks, say $LB_{S_i}, i = 1, 2, ..., k$, where k is the number of shards in the blockchain network and generates

one global block (*GB*) that contains all the transactions from all the local blocks. It is important to note that during the *GB* generation, instead of following the Merkle root tree to get the root hash of the transactions, it uses the DAG structure to be included in the block header. The important purpose of following the DAG over the Merkle root tree is described in the subsequent section. Another key aspect of the Cross-shard consensus is that to establish the cooperation between all the shards for the confirmation of the LB_{S_i}, we have used the concept of VoteChain (VC). That is, VC enables us to synchronize and verify the transactions that are included in the *LB*s from each shard. Moreover, before proceeding with VC and *GB*, this phase selects the inter-shard coordinator from each shard. Thus, the process of this phase is divided into three steps, namely 1) Inter-Shard Coordinator Selection, 2) Creating VoteChain (VC) to confirm all the LB_{S_i}, and 3) creating the *GB* and appending it to the *GL*.

Inter-Shard Coordinator Selection

It is the first step of cross-shard consensus. This step is crucial for the effective and attack-free working of UnoShard's whole sharding process. The inter-shard coordinator selection process is executed based on three important parameters, which are the Accumulative Reputation Score (ARS), Decaying Reputation Score (DRS), and Performance Metrics-Based Reputation Score (PMRS). The whole process of inter-shard coordinator selection is described in the *Algorithm* 5. Because this *algorithm* uses multiple parameters in the computation, we called this a *Multi-Facet Reputation-based Algorithm*. This *algorithm* computes the reputation of each node in the network through a series of reputation factors. The definition of four important parameters is as follows.

A. Accumulative Reputation Score (ARS): This reputation score is calculated based on the node's participation and reliability. This score increases and decreases with the node's positive and negative contribution, respectively. The positive contribution includes successful validation and efficient transaction processing. The value of *ARS* can be 0 or 1, that is, $ARS \in [0,1]$ where 0 means negative participation and 1 means positive participation.

B. Decaying Reputation Score (DRS): This score value represents the participation of the node as active or inactive. The active and inactive participation of the node can be measured based on the response time, which is included in the communication parameter as the *Timestamp*. Again, this can be either 0 or 1, that is, $DRS \in [0,1]$, where 0 means inactive participation and 1 means active participation.

C. Performance Metrics-Based Reputation Score (PMRS): As the name itself described, this score is derived from the addition of quantitative values such as the computation power score (CPS) and the block success rate (BSR). Both the score can be computed as follows:

$$CPS_{N_i} = \frac{Hashing\ Power\ of\ N_i}{Total\ hashing\ power\ in\ the\ network} \tag{3.6}$$

The PMRS incorporates the node's hashing power and block success rate to evaluate its computational efficiency. It is important to note that mentioning "hashing power" in this context does not imply the use of PoW within the UnoShard protocol. Instead, it serves as a performance metric to ensure that nodes involved

in the consensus process have adequate computational resources.

$$BSR_{N_i} = \frac{Number\ of\ Valid\ Blocks\ from\ N_i}{Total\ number\ of\ blocks\ in\ the\ chain} \tag{3.7}$$

After this, the Weighted Sum Calculation (WSC) is carried out as,

$$WSC_{N_i} = CPS_{N_i} + BSR_{N_i} \tag{3.8}$$

Calculate the reputation score ($PMRS_i$) of node N_i using a weighted geometric mean approach (it balances the influence of both parameters) for each node N_i using the finalized reputation calculation method:

$$PMRS_{N_i} = \sqrt{WSC_{N_i}} \tag{3.9}$$

Normalize this value in the range of 0 and 1. That is, this can also be either 0 or 1, i.e., $PMRS \in [0,1]$, where 0 means low performance and 1 means optimum performance.

VoteChain: Confirming All the Local Blocks

VoteChain ensures trust on all the local blocks (LB_{S_i},$i = 1,2,...,k$), where k is the number of shards in the blockchain network. It is an important step towards the finality of the global ledger. A VoteChain comprises linked VoteBlocks, which are generated for all the corresponding *LB*s. Each shard coordinator votes on the validity of each local block generated by each shard. For instance, if there are 20 shards, which generate 20 local blocks, then VoteChain should contain 19 votes on each local block. In this way, we can ensure that every transaction included in *GB* is verified and assigned.

A VoteBlock contains at least $k-1$ voting transactions in each LB_i. Each vote belongs to a unique shard among all k shards. This can be understood as follows. Each VoteBlock corresponds to each local block, i.e., Let us say there are 20 local blocks, then there will be 20 VoteBlock. This illustration is depicted in Fig. 3.5. Further, the VoteBlock constructs a Merkle tree whose leaf nodes are the hash value of each vote transaction generated by the shard coordinators with respect to the corresponding local block. The root of the Merkle tree is included in the header of the VoteBlock. Finally, each VoteBlock is broadcasted to the whole network to help each shard acknowledge the consensus results. Since each VoteBlock is very small in size, it barely affects the storage overhead of the system. Usually, each shard maintains the VoteChain to verify the latest VoteBlocks and make historical consensus results auditable.

All shard coordinators verify the proof of all local blocks generated from all other shards and generate vote transactions as valid local blocks or invalid local blocks. After the intra-shard communication phase, as described above, i.e., transaction verification and recording as a local block, all the corresponding shard coordinators broadcast that local block (LB_i) to other peer shard coordinators for verification and generation of vote transaction to be included in the VoteChain. A local block is to be considered valid if all the shard coordinators confirm the local block as a valid block. When all the local blocks are confirmed, the leader node from the shard coordinator starts the processing of the next phase, i.e., global block generation and final consensus for the global ledger.

It is worth noting that the concept of using vote-based consensus has also been explored in the PRISM protocol [87], which decouples the consensus process into three distinct steps involving proposer blocks, transaction blocks, and vote blocks. The PRISM protocol achieves high throughput and scalability by

Algorithm 5: Multi-Facet Reputation-based Algorithm

Input: AT_i, DT_i, PMT_i
Output: Shard Coordinator Selection based on Reputation Score
Initialization:
for *each node i* **do**
    AT[i] ← 0; DT[i] ← 0; PMT[i] ← 0;
Function InitializeTrustScore(node i);
begin
    AT_i ← InitialATValue, DT_i ← InitialDTValue, PMT_i ← InitialPMTValue;
Function *computeTrustScore*(node *i*, α, β, γ, δ);
begin
    $totalScore \leftarrow \alpha * AT_i + \beta * DT_i + \delta * PMT_i$;
Procedure *SelectLeader*;
begin
    for *each node i* **do**
        score ← *computeTrustScore*(*i*, α, β, γ, δ);
        if *score* > *maxScore* **then**
            maxScore ← score;
            leader ← i;
    return leader;
Procedure *updateTrustScore*(node *i*, AT_Update, DT_Update, PMT_Update);
begin
    $AT_i \leftarrow AT_i + AT_Update$;
    $DT_i \leftarrow \max(0, DT_i - DT_Update)$;// `Decay factor`
    $PMT_i \leftarrow PMT_i + PMT_Update$;
begin
    for *each node i* **do**
        updateTrustScore(*i*, AT_Update_i, DT_Update_i, PMT_Update_i);

employing this multi-block structure. In contrast, UnoShard's approach focuses on sharding the network to process transactions in parallel across multiple shards. While both protocols utilize voting for consensus, UnoShard applies this within the context of sharded local blocks, followed by the aggregation of these blocks into a global block. This distinction shows UnoShard's unique method of leveraging sharding and VoteChain to enhance blockchain scalability and security.

Final Consensus: Global Block Creation and Ledger Updation

In a sharded blockchain network, each shard processes transactions independently, creating local blocks. A leader node then aggregates these local blocks to form a global block. However, this structure introduces the possibility that a single transaction is recorded in multiple local blocks. The traditional solution, which compares each transaction across all shards, results in significant computational and time overhead, making it an impractical approach for large-scale networks. The DAG-based approach is posited as an alternative

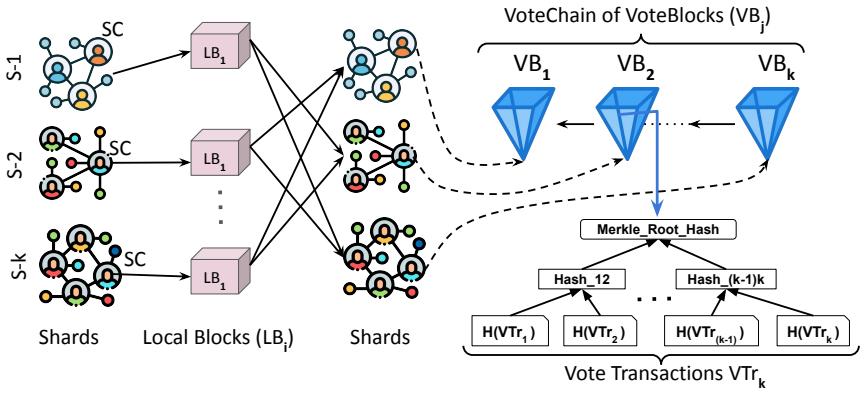


Figure 3.5: This figure illustrates the generation of local blocks (LB_i) and VoteChain of VoteBlocks (VB_j). A Shard Coordinator (SC) is elected in each shard, as depicted here.

to traditional methods for forming global blocks. DAG allows for a nonlinear arrangement of transactions. For each transaction T_{ij} in a local block B_i, a node is created in the DAG. If T_{ij} references another transaction T_{mn} in a different local block B_m, a directed edge is drawn from T_{ij} to T_{mn}. If two transactions, T_{ij} and T_{xy}, reference the same input transaction T_{mn} (i.e., if $T_{ij} \rightarrow T_{mn}$ and $T_{xy} \rightarrow T_{mn}$ exist in the DAG), it indicates potential double-spending. The system flags T_{ij} and T_{xy} for further validation. Each flagged transaction is then individually validated. The validation of each transaction is done by checking timestamp(T_{ij}) and timestamp(T_{xy}). If timestamp$(T_{ij}) <$ timestamp(T_{xy}), then T_{ij} is considered valid, and T_{xy} is marked as invalid (double spend), and vice versa. If timestamp(T_{ij}) and timestamp(T_{mn}) have the same timestamp, secondary criteria such as transaction ID and nonce value are used to determine the original transaction. The transaction with the lower lexicographical order or lower nonce value is considered the original. Additionally, if T_{ij} has a lower timestamp than T_{mn} but is broadcasted by an adversary much later than T_{mn}, nodes track the broadcast time and validate transactions within a predefined time window to ensure timely detection of adversarial delays. Initially, T_{mn} would be accepted due to its earlier broadcast time. If T_{ij} is later detected within the allowable time window and validated as the original transaction based on the lower timestamp, T_{mn} would be marked as invalid.

Once the original (valid) transaction is identified, the conflicting (invalid) transaction is discarded or marked as a double spend. The DAG is then updated after the invalid transaction is removed. This update might propagate through the DAG, affecting how other transactions are connected. The updated status on valid and invalid transactions is broadcasted to the entire network, ensuring that all nodes and shards have the latest and accurate state of the ledger. The global block is finalized with the remaining valid transactions and committed to the global ledger. Efficient data structures (i.e., DAG) and the sharding mechanism ensure that the global block remains practical in size. By assuming a maximum of 40-45 shards, each handling 500-800 transactions per local block, UnoShard can manage a global block size of up to 36,000 transactions. This approach leverages proven methodologies from high-throughput blockchains like EOS [140] and Solana [141], ensuring an effective and scalable block size even when aggregating transactions from multiple shards.

The *Algorithm* 6 describes the process of compiling transactions from local blocks of individual shards into a unified global block. It starts by constructing a DAG where each transaction from the shards acts as one of the nodes of DAG construction. This DAG reflects the dependencies between transactions on different shards. The algorithm then identifies possible double-spending situations by looking for transactions

referencing the same input in the DAG. These suspect transactions are validated based on their timestamps, determining which transaction came first and, therefore, is valid. Transactions found to be double spends are marked as invalid and removed from the DAG. After resolving these conflicts, the algorithm compiles all valid transactions into a global block, which is then broadcasted for network-wide verification and subsequently committed to the global ledger. This approach efficiently maintains the integrity and consistency of the blockchain ledger in a sharded architecture.

Algorithm 6: Global Block Generation

Input: LocalBlocks (Array of local blocks from each shard),
DAG (Directed Acyclic Graph representing transactions)
Output: GlobalBlock (Finalized global block for the ledger)
Function `globalBlock(`*block, nonce*`)`**:**

```
Initialize an empty DAG;
for Every LB_i in LocalBlocks do
    for Every T_ij in LB_i do
        Create a node in the DAG for T_ij;
        if T_ij references another T_mn in a different LB_m then
            Draw a directed edge from T_ij to T_mn in the DAG;
for Every ((T_ij, T_mn)) in the DAG that reference the same input transaction do
    Flag (T_ij, T_mn) for validation;
    Validate each flagged transaction by comparing timestamps;
    if (T_ij == T_mn) then
        Compare Transaction IDs;
        if (T_ij < T_mn) then
            Mark T_ij as valid and T_mn as invalid;
        else
            Mark T_mn as valid and T_ij as invalid;
    else
        if (T_ij < T_mn) then
            Mark T_ij as valid and T_mn as invalid;
        else
            Mark T_mn as valid and T_ij as invalid;
    Update the DAG by removing links associated with invalid transactions;
Broadcast(Tr_update) ;
create(GlobalBlock) ;
commit(GlobalBlock)
```

3.6 PERFORMANCE ANALYSIS

The performance analysis of BlockTree is performed based on four parameters: consensus delay, throughput, fairness, and block congestion control. We found these four parameters are very crucial for the perfor-

mance of the blockchain. First, we provide the simulation environment of BlockTree, which includes some important assumptions about the simulation.

3.6.1 Simulation Environment

We evaluate BlockTree with 1000-node experiments on a customized simulation with parameters like transaction frequency, consensus delay, fairness, and leader selection time based on different block sizes (512 KB to 9MB). Our simulation assumes a network delay of 100ms with a variance of 10ms to reflect typical conditions in a large-scale distributed blockchain network. We also assume a message loss rate of 0.1% to represent a reliable network environment and utilize a mesh network topology, where each node is connected to multiple other nodes, ensuring robustness and resilience. We compared BlockTree with Bitcoin, Bitcoin Cash, Bitcoin-NG, Algorand, Obscuro, and Prism. In our analysis, we referenced the empirical performance data presented in the original studies of state-of-the-art protocols, enabling a consistent and fair evaluation against BlockTree. We endeavored to conduct the most thorough comparative analysis possible, given the resources available, ensuring that our evaluation was aligned with established standards in the field. The evaluation results show that it is possible to improve consensus delay and throughput (number of transaction executions) by modifying the structure and consensus algorithm while maintaining the same level of security. BlockTree outperforms almost all the metrics across all the measured parameters. Fork generation, a common drawback in most previous cryptocurrencies, has been addressed in the BlockTree protocol. We run the protocol with different block sizes to test throughput, scalability, consensus delay, fairness, and block congestion control, as seen in the comparative graphs below. We vary the block sizes from 512 KB to 9MB. It takes more time to verify and propagate as the block size increases.

3.6.2 Consensus Delay

Figure 3.6a shows the consensus delay with varying block sizes. The results indicate that as the block size increases, the consensus delay also increases because a larger block takes more time to verify and propagate. However, in BlockTree, the consensus delay is significantly lower than that of state-of-the-art schemes such as Algorand, Bitcoin-NG, and Prism. This demonstrates that BlockTree can perform transactions within a few seconds, even as block sizes grow. One of the key advantages of BlockTree is its m-way tree structure, which allows for parallel processing and efficient transaction management. This structure helps maintain lower consensus delays as the network scales. During our tests, we found that the m-way nature of BlockTree ensures that the consensus delay remains minimally affected as the number of miners increases.

When the number of miners surpasses the defined limit m, the network automatically splits and creates a new shard. This dynamic shard creation ensures that BlockTree does not face scalability issues that typically plague other blockchain protocols. Additionally, the efficient leader selection algorithm and the streamlined verification process further contribute to maintaining low consensus delays, even with increasing network size. As a result, BlockTree outperforms other schemes not only in scalability but also in maintaining rapid transaction processing times, making it a robust solution for high-throughput blockchain applications.

3.6.3 Throughput

Figure 3.6b shows the transaction frequency with varying block sizes. The results indicate that BlockTree performs exceptionally well compared to other schemes. As the block size increases, the transaction frequency also increases, thereby enhancing the throughput of BlockTree. This superior performance can be attributed to the m-way tree structure of BlockTree, which allows for efficient parallel processing and management of transactions across multiple shards. Additionally, we tested the throughput of BlockTree with varying time intervals between leader nodes, ranging from 30 seconds to 100 seconds. The results show that as the time interval between leader nodes increases, the throughput also increases. However, it is important to note that if the interval increases excessively, it may lead to a situation similar to monologues, where a single leader node generates an excessive number of data blocks. Therefore, the interval should not exceed a reasonable limit to ensure fairness and maintain optimal performance.

Unlike other state-of-the-art schemes like Algorand, Bitcoin-NG, and Prism, BlockTree consistently achieves higher throughput. This is due to its innovative consensus mechanism and efficient block verification process. The ability to dynamically create new shards as the network grows further ensures that BlockTree can handle a significant increase in transaction volume without compromising performance. As a result, BlockTree improves scalability and maintains high throughput, making it a suitable solution for large-scale blockchain applications. The throughput analysis clearly demonstrates that BlockTree can handle a high volume of transactions efficiently, even with increasing block sizes and varying leader selection intervals. This capability makes BlockTree a robust and scalable blockchain protocol suitable for diverse and demanding applications.

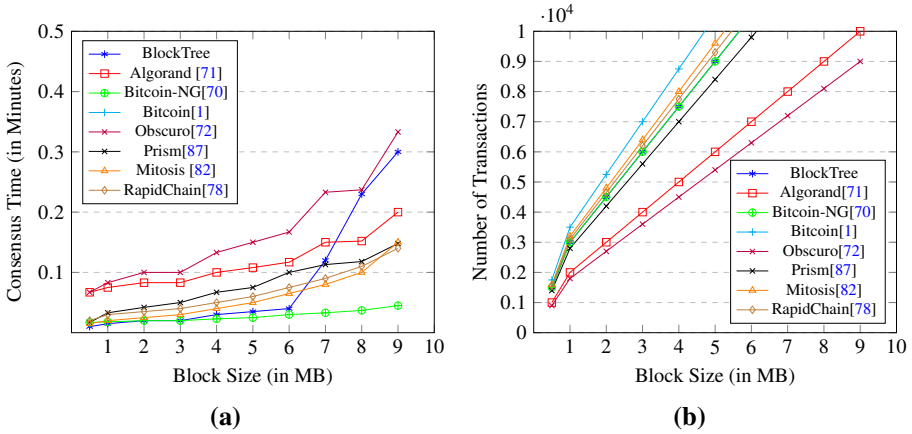


Figure 3.6: Comparative analysis of BlockTree with state-of-the-art schemes based on the (a) consensus delay and (b) throughput

3.6.4 Fairness

Figure 3.7a illustrates the fairness of block proposers in BlockTree compared to other state-of-the-art schemes. As the number of miners increases, the likelihood of different miners generating blocks decreases in many traditional schemes. However, BlockTree demonstrates superior fairness across varying numbers of miner nodes, from 100 to 1000, maintaining a more equitable distribution of block generation opportunities. This enhanced fairness in BlockTree is primarily due to its efficient sharding mechanism. BlockTree

maintains a fixed, defined size for each shard, ensuring that every node within a shard has an equal opportunity to become a leader node. This approach contrasts with other schemes where the competition among all miners can lead to disproportionate chances for block generation, often favoring more powerful nodes. In BlockTree, as the number of miners increases and surpasses the defined limit for a shard, the network splits, and a new shard is created. This dynamic creation of shards helps distribute the mining power more evenly across the network, preventing any single node or a small group of nodes from dominating the block generation process. The fixed shard size ensures that each shard remains manageable and that the nodes within each shard operate under fair conditions.

Unlike Algorand, Bitcoin-NG, and Prism, BlockTree's sharding mechanism provides a significant advantage in maintaining fairness. By ensuring that each node has a uniform chance to become a leader, BlockTree mitigates the risk of centralization and promotes a more decentralized and equitable blockchain environment. This makes BlockTree fairer but more resilient and secure, as it prevents the concentration of mining power that could otherwise lead to potential vulnerabilities and attacks. Overall, the fairness analysis underscores BlockTree's capability to provide a balanced and just platform for block generation, even as the network scales up, reinforcing its position as a leading blockchain protocol regarding fairness and decentralization.

3.6.5 Block Congestion Control

In cryptocurrencies like Bitcoin, Bitcoin-NG, Algorand, and others, every miner competes with the rest of the nodes to win block creation compensation. This competition often leads to block congestion as many nodes generate blocks simultaneously, resulting in illegitimate fork generation. BlockTree effectively addresses this issue by creating multiple fixed-size shards. Instead of having all miners compete for the same block, these shards allow different groups to create different blocks for separate transactions. This approach significantly reduces block congestion and improves throughput. BlockTree's sharding mechanism ensures that each shard operates independently, minimizing the competition within a shard and distributing the workload evenly across the network. As a result, the likelihood of block congestion and forks is greatly reduced.

Figure 3.7b illustrates the block congestion control capacity of BlockTree compared to state-of-the-art schemes. The results indicate that BlockTree outperforms other schemes by maintaining lower levels of congestion and higher throughput. This performance improvement is primarily due to the efficient sharding mechanism, which distributes transactions across multiple shards, preventing bottlenecks and ensuring a smoother transaction flow. Additionally, the fixed-size shard structure in BlockTree ensures that each shard remains manageable and operates under optimal conditions. This prevents any single shard from becoming overwhelmed with transactions, further contributing to the overall efficiency and scalability of the network. By reducing congestion, BlockTree not only enhances throughput but also improves the overall user experience by ensuring faster transaction confirmations and reducing the risk of network slowdowns. Consequently, BlockTree's approach to block congestion control through sharding provides a significant advantage over traditional blockchain protocols. It effectively mitigates the issues of block congestion and fork generation, leading to a more robust, scalable, and efficient blockchain network.

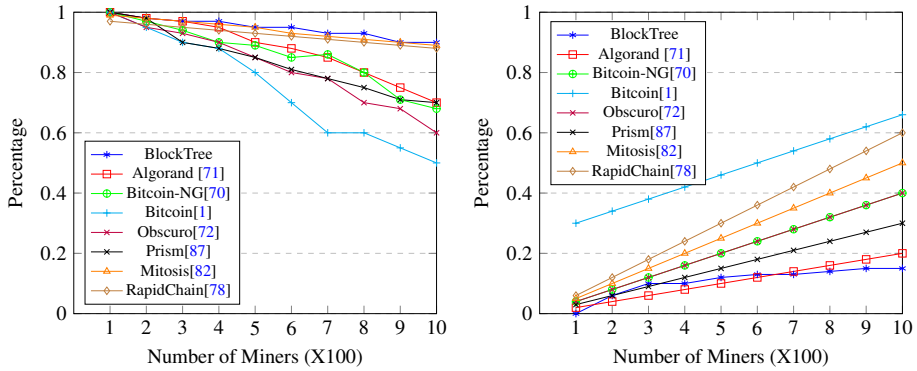


Figure 3.7: Comparative analysis of BlockTree with state-of-the-art schemes based on (a) fairness and (b) block congestion control

3.7 SECURITY ANALYSIS

We have analyzed the security of BlockTree based on three parameters: double-spending, illegitimate fork, and scalability. Below is the qualitative discussion on how BlockTree has prevented these attacks and maintains high scalability.

Proposition 1. *In the UnoShard system, the probability $P_{takeover}$ of 1% attack or shard takeover attack, defined as more than half of the nodes in any shard being malicious, is negligibly small.*

Proof. Let N Total number of nodes in the network, S: Number of shards in the network, $N_S = \frac{N}{S}$: Number of nodes per shard, assuming an even distribution, T: Threshold for shard takeover, defined as the majority ($> \frac{N_S}{2}$) of malicious nodes in a shard, and P_M: Probability that any given node is malicious. The probability of a shard takeover in the UnoShard blockchain network can be modeled as a binomial probability problem. We calculate the probability that more than half of the nodes in a shard are malicious. The binomial distribution gives the probability of exactly k nodes in a shard being malicious:

$$P(X = k) = \binom{N_S}{k} P_M^k (1 - P_M)^{N_S - k} \tag{3.10}$$

To find the probability of a shard takeover, we need to calculate the cumulative probability of more than half the nodes in a shard being malicious. This is given by:

$$P_{takeover} = \sum_{k=\frac{N_S}{2}+1}^{N_S} \binom{N_S}{k} P_M^k (1 - P_M)^{N_S - k} \tag{3.11}$$

where the sum accumulates the probabilities for all scenarios where more than half of the nodes are malicious. However, in all the scenarios affecting the value of $P_{takeover}$, the P_M is the most vulnerable parameter because, in a public blockchain, where any node can join and participate in the consensus process with the ware minimum authentication process, it is hard to say P_M could be typically low. Apart from the P_M, the other factors are fair shard allocation based on DPoS and trust-based intra-shard consensus.

In UnoShard, the shard allocation process is governed by DPoS, where only a selected group of nodes (delegates) participate in consensus. These delegates are elected based on their reputation or stake, which reduces the likelihood of malicious nodes gaining significant influence, thereby effectively lowering P_M over time. Furthermore, The fair shard allocation mechanism ensures an even distribution of nodes across shards, making it difficult for any single shard to be dominated by malicious nodes. Even if the initial P_M is higher, the distribution helps keep $P_{takeover}$ low. Along with fair shard allocation using DPoS, the trust-based consensus mechanism, which dynamically adjusts the influence of nodes based on their behavior, plays a big role in lowering the probability of shard takeover. Malicious nodes quickly lose trust, limiting their impact on the network's decision-making process. This mechanism acts as a secondary layer of security, ensuring that even if P_M is higher initially, its influence on shard takeover risk is mitigated. Given these mechanisms, the cumulative effect is that $P_{takeover}$ remains low, though it may not be "negligibly small" in the strictest mathematical sense. The assertion of "negligible" should be understood in the context of these mitigating factors rather than as an absolute statement. □

Proposition 2. *The UnoShard integrates a fair shard allocation method, trust-based intra-shard consensus, and cross-shard consensus with the DAG to secure from majority attack. This is defined by the system's ability to manage node failures or malicious activity while maintaining the integrity and operation of the whole network.*

Proof. UnoShard's resilience against majority attacks is achieved through the integration of fair shard allocation, a trust-based Practical Byzantine Fault Tolerance (PBFT) consensus mechanism within each shard, and a cross-shard consensus process that utilizes a Directed Acyclic Graph (DAG). This combination of mechanisms ensures that UnoShard can handle node failures or malicious activities while maintaining the network's integrity and operational continuity.

Initially, the UnoShard begins by evenly distributing nodes across multiple shards. Let S represent the number of shards and N the total number of nodes, resulting in each shard containing $N_S = \frac{N}{S}$ nodes. This fair shard allocation minimizes the likelihood of any single shard being disproportionately controlled by malicious nodes, as it prevents the concentration of such nodes within one shard. In the next phase, the UnoShard employs the PBFT consensus mechanism within each shard, which requires at least $\frac{2}{3}$ of the nodes to agree for a transaction to be validated. This consensus model allows the system to tolerate up to $\frac{1}{3}$ of the nodes being faulty or malicious. The resilience of each shard against majority attacks is represented by $R = 1 - \frac{1}{3N_S}$ indicating that as long as fewer than one-third of the nodes in a shard are compromised, the shard will remain secure and operational. Lastly, the cross-shard consensus process in UnoShard leverages a DAG to manage and validate transactions across shards, ensuring consistency and resolving any conflicts that arise from interactions between shards. The use of DAG enhances the fault tolerance of the entire network by synchronizing the state of all shards, thereby maintaining the overall network's integrity even if some shards encounter issues.

The combined fault tolerance of UnoShard can be expressed as $P_{\text{tolerance}} = P_f \times R$ where P_f is the probability that a single shard failure impacts the network. Given that P_f is proportional to $\frac{1}{S}$, the likelihood of a shard failure affecting the entire network decreases as the number of shards increases. This, combined with the shard-level resilience R, ensures that UnoShard is robust against majority attacks, providing a secure and reliable blockchain network even in the presence of malicious nodes. □

3.7.1 Prevention of Double Spending

BlockTree: Double spending means spending the same money twice. It is not an issue in physical currency, but it is a big problem in digital currency. In digital currency, a transaction carrying the digital cash is broadcasted to all the nodes in the network for verification. It is possible that another node copies the transaction. Before confirmation, it was rebroadcast into the network. Double spending attacks are a big concern for Bitcoin, Bitcoin-NG, and other cryptocurrencies. In the case of Bitcoin, Bitcoin-NG, this issue remains there due to the long confirmation time. However, in BlockTree, all the transactions are bunched in the data block via the leader node only. While the leader node generates the data block, no other node can create the data block. Therefore, the possibility of rebroadcasting the same transaction again in the network is eliminated. Hence, the possibilities of double-spending are negligible.

Proposition 3 (UnoShard)**.** *The cross-shard consensus in UnoShard uses a Directed Acyclic Graph (DAG) to record transactions in a global block. Let T_{ij} and T_{xy} be transactions in UnoShard which references the same input transaction T_{mn}. The DAG structure ensures that UnoShard validates at most one of T_{ij} and T_{xy}, thereby preventing double spending attacks.*

Proof. Each transaction in the DAG model of UnoShard is a vertex. When two vertices T_{ij} and T_{xy} relate to a common input T_{mn}, directed edges towards T_{mn} are produced, implying the possibility of double-spending. A time-based validation is the solution: If $Timestamp(T_{ij}) < Timestamp(T_{xy})$, then T_{ij} is considered as valid, while T_{xy} is considered as invalid; otherwise, vice versa. The DAG then undergoes pruning to remove invalid transactions, denoted as $\mathscr{F} : DAG \rightarrow DAG'$, where DAG' is the updated graph. Valid transactions from DAG' are aggregated into the global block, denoted as $\mathscr{G} : DAG' \rightarrow GB$, where GB represents the finalized global block. This mechanism guarantees that only transactions without double-spending issues are included, preserving the blockchain system's integrity. □

3.7.2 Illegitimate Fork Prevention

In the BlockTree protocol, several mechanisms ensure that the leader cannot create forks and that any attempts to do so are effectively managed. The leader selection algorithm guarantees that only one leader is active at any given time. This selection process involves solving a cryptographic puzzle and incorporates checks on the status and age of nodes. Nodes with a "bad" status are excluded, and among the remaining nodes, the one with the lowest age is preferred, ensuring fair and reliable leader selection. The selected leader's identity is broadcast to the network, ensuring transparency. BlockTree distinguishes between leader blocks and data blocks. Once a leader is selected, a leader block is first created, marking the beginning of the term. During the specified epoch, the leader generates data blocks. This distinction helps maintain order and prevent unauthorized block creation. This concept is similar to the Bitcoin-NG protocol, where the blockchain is divided into epochs managed by a leader who generates key blocks and microblocks. Key blocks establish the leader for an epoch, while microblocks contain transactions. This separation ensures efficient transaction processing and prevents issues like stale blocks common in traditional blockchain protocols. Unlike Bitcoin-NG, where the current leader node continues generating microblocks until a new key block arrives, BlockTree restricts the leader's activity to a specific time interval. After this interval, all nodes switch to leader selection mode to choose a new leader. This approach prevents the problem of unnecessary fork generation, ensuring no overlap in leader activities. Thus, the possibility of parallel data block generation by multiple nodes in the same group is minimized, effectively eliminating forks within the same group. The BlockTree consensus algorithm follows the 2K+1 rule, ensuring that the number of honest

nodes is always greater than the number of malicious nodes. For a block to be accepted, it must receive votes from at least 2K+1 nodes.

If a leader attempts to create a fork by generating two data blocks with the same parent, the protocol's verification processes detect this. Nodes continuously verify blocks, and any discrepancies in hash values indicate a fork attempt. Honest nodes will vote only for the valid block, rejecting the conflicting one. The block with the highest number of valid confirmations is added to the main chain, and the conflicting block is discarded. Malicious leaders are penalized by being marked with bad status, excluding them from future leader selections.

3.7.3 Scalability Enhancement

Apart from the fork and double-spending, the high transaction load is also a limitation of blockchain implementation, where every transaction is validated and executed on every node. To maintain security and improve scalability, public blockchains impose upper limits that manage the rate at which they can process transactions and the rate at which new blocks are created. In Bitcoin and Bitcoin-NG, as they are public blockchains, there might be any number of nodes who race to mine the block as soon as possible to get rewarded. Due to this race amongst nodes, much of the mining power goes in vain. Also, this leads to the scalability issue. To deal with the scalability issue and utilize the mining power, BlockTree modified the blockchain structure into the m-way tree-like structure. BlockTree creates the shards when the number of miners becomes greater than the defined size m, and a new fork is created in the current chain from that instant. This fork is the legitimate fork in the chain. A newly created shard starts mining in the new fork of the chain. In this way, the mining competition is reduced, and now miners in all the shards mine the blocks parallelly. For example, if 1000 miners were mining the same set of transactions and were now divided into ten shards of 100 miners, the blockchain's throughput would increase by ten times.

Concluding Remark

In this chapter, we have addressed the critical issue of scalability in public blockchains by introducing two novel consensus algorithms: BlockTree and UnoShard. Through an in-depth exploration of their architectures and mechanisms, we have demonstrated how these protocols significantly enhance the scalability and throughput of blockchain networks while maintaining high levels of security. BlockTree's innovative structure allows for efficient block validation and propagation, mitigating common bottlenecks that hinder scalability. Similarly, UnoShard distributes the workload across multiple shards, enabling the system to scale effectively without compromising security. UnoShard ensures fair shard allocation and fair load distribution, along with the prevention of shard takeover attacks and double spending attacks. Our comparative analysis reveals that both BlockTree and UnoShard outperform state-of-the-art scalability solutions, offering promising directions for the future of scalable public blockchains. Building on the advancements made in this chapter, we now turn our focus to the challenge of interoperability in blockchain networks in Chapter 4. As blockchain technology continues to evolve, the ability to seamlessly communicate and transact across different blockchain platforms has become increasingly important. Chapter 4 introduces the *CrossLedger* protocol, designed to enable secure and efficient cross-chain asset transfers. *CrossLedger* paves the way for a more interconnected and versatile blockchain ecosystem by addressing the complexities of interoperability. This transition from scalability to interoperability reflects the broader goal of enhancing blockchain technology to meet the diverse needs of its users across different platforms.