TomoChain Lab Pte. Ltd.
219 Trung Kinh, Hanoi, Vietnam
10 Anson Road #22-15 International Plaza, Singapore

TomoChain Proposal for Decentralized Applications-Oriented Proof-of-Stake Sharding Architecture

Preliminary draft, comments are welcomed

Cam Pham at campv@tomochain.com

PREFACE

TomoChain’s Vision and mission

Our mission is to be a leading force in building the Internet of Value, and its infrastructure. We are working to create an alternative, scalable financial system which is more secure, transparent, efficient, inclusive and equitable for everyone.

TomoChain is an innovative solution to the scalability problem with the Ethereum blockchain, and other current blockchain platforms. TomoChain relies on a system of 150 Masternodes with Proof of Stake Voting (POSV) consensus that can support near-zero fee, and 2-second transaction confirmation time. Security, stability and chain finality are guaranteed via novel techniques such as double validation, staking via smart-contracts and "true" randomization processes.

TomoChain supports all EVM-compatible smart-contracts, protocols, and atomic cross-chain token transfers. New scaling techniques such as sharding, EVM parallelisation, private-chain generation, hardware integration will be continuously researched and incorporated into TomoChain's Masternode architecture which will be an ideal scalable smart-contract public blockchain for decentralized apps, token issuances and token integrations for small and big businesses.

Scope

This document describes TomoChain’s initial proposal for decentralized applications-oriented Proof-of-Stake Sharding Architecture. This preliminary draft is not our final design specification and it is subject to change. Comments are welcomed and appreciated.

ABSTRACT

This paper proposes a sharding architecture solution for the TomoChain public blockchain infrastructure. Besides the aims at significantly improving the transaction processing performance in the current TomoChain design with the Proof-of-Stake Voting (PoSV) consensus, the presented sharding is designed in order to support decentralized applications' performance. The solution uses PoSV for intra-shard consensus because PoSV provides two-second block-time and fast confirmation time. The paper identifies and solves cross-shard transactions for smart contract message calls chain and data availability problem in shard reshuffling. Specifically, we propose two cross-shard smart contract transaction schemes combined with a new smart contract deployment strategy that places dependent smart contracts onto the same shard. Moreover, we also address the data availability problem in state sharding when shard reshuffling takes place. In addition, to be able to provide security and safety for shards, we provide an incentive-driven mutual verification game that aims at offering fast transaction confirmation time and fast detection of a malicious behavior that creates fraudulent blocks. Masternodes of a shard that create a fraudulent block will be detected and penalized by losing all of their deposits to the node that detects the invalidity of the block.

Keywords

Sharding, Blockchain, Smart contract, TomoChain, cross-shard transactions, randomization, data availability, security, PoSV, consensus.

I. INTRODUCTION

Blockchain has become one of the most disruptive technologies that enable many decentralized applications, including but not limited to cryptocurrencies, smart contract, voting and supply chain management. Blockchain proponents have been trying to inject it into this Industry 4.0 revolution era. In order to reach this goal, the current blockchain infrastructure must at least be able to compete with many mainstream technologies, such as Visa and MasterCard processors in financial services. Nevertheless, the reality is disappointing these blockchain proponents: Bitcoin [1] and Ethereum [2] only can process around 7 and 15 transactions per second, respectively. Remember that, these numbers in Visa and MasterCard are around 5000-6000 transactions per seconds.

Several scaling solutions have been proposed by the proponents, including on-chain and layer 2 scaling solutions. Off-chain scaling solutions are Lightning [3] network for Bitcoin, State Chanel [4] and Plasma [5] for Ethereum, just to name a few. Regarding on-chain scaling, Sharding and Proof-of-Stake-based consensus are of potential that can significantly improve transaction processing performance while trying to maintain security requirements of the system.

Sharding challenges: Sharding is a technique that is inspired by the concept of database sharding in which the whole database is divided into sub-databases which will then be deployed on different servers. The goal is to parallelize the transaction processing by dividing the blockchain network into sub-networks each of which stores a portion of the whole blockchain and processes a subset of non-overlapping transactions. The application of sharding to decentralized blockchain systems has the following technical challenges that will be addressed in this paper:

- Assignment of masternodes to specific shards: This process is critical to ensure that masternodes are assigned to shards in a randomized manner to avoid an adaptive attack. In the latter, malicious masternodes can all join the same shard in order to collude it.
- Cross-shard transaction: Cross-shard transactions enable every account to transact with any other accounts. The challenge is how to execute cross-shard transactions securely and safely knowing that a cross-shard transaction involves more than two shards. Furthermore, the problem is even more challenging if the receiver is a smart contract.
- Cross-shard smart contract message calls chaining: The problem of cross-shard transaction becomes very difficult if a smart contract in one shard calls another smart contract in another shard, which in turn invokes another contract in another shard. This cross-shard smart contract call chaining if not properly processed will be easy to attack and violate security requirements and harmful to the performance of the whole system.
- Shard reshuffling through randomization and data availability: The problem with state sharding is when the network reshuffles the set of masternodes for each shard after a period of time (often called *epoch*), the masternodes that switch from one shard to a new shard need to synchronize the current data of the new shard in order to execute new transactions for that shard. If not carefully treated, the data synchronization takes a significant amount of time and decreases the performance of the system.

Previous works on sharding: Sharding has its origin from permissioned or closed distributed databases in which a centralized database is partitioned into smaller databases that store information related to a subset of users. The smaller databases are then maintained by trusted infrastructures provided by infrastructure providers such as Google or Amazon. However, the sharding in these closed distributed databases cannot be applied to public/permissionless blockchains in which one masternode does not have to trust in other masternodes in the Internet.

Several blockchain-related works have been trying to propose a sharding solution that can leverage the transaction processing performance. In [6], Elastico is proposed as the first sharding solution for public blockchains. Elastico partitions the blockchain network into smaller committees, each of which then processes a disjoint set of transactions, called a shard. However, Elastico offers horizontal scaling but with high failure probabilities [12]. Zilliqa [7] inherits the

sharding architecture solution from Elastico. Both Elastico and Zilliqa support network sharding and utilizes Practical Byzantine Fault Tolerance [8] as intra-shard consensus. In these approaches, micro blocks, which are created by the shards, are then aggregated into a final block by another consensus round run among the final committee members. OmniLedger [9] using the Unspent Transaction Output (UTXO) provides atomic cross-shard transactions. A common missing feature of these aforementioned approaches is the lack of smart contract support. If two transactions from the senders in different shards, which execute the same smart contract, would cause concurrent conflict updates in the different shards to the state of the same smart contract.

In this paper, we propose a sharding architecture for TomoChain but can be applied to any Proof-of-Stake-based blockchain systems to address the aforementioned issues. Furthermore, to be able to provide security and safety for shards, we provide an incentive-driven mutual verification game that aims at offering fast transaction confirmation time and fast detection of a malicious behavior that creates fraudulent blocks. Masternodes of a shard that create a fraudulent block will be detected and penalized by losing all of their deposits to the node that detects the invalidity of the block.

The remainder of this paper is structured as follows: Section II describes the sharding architecture and the consensus protocol of the solution addressing the previously mentioned problem. Section III provides our solutions to different cross-shard transaction schemes. Section IV presents an incentive-driven mutual game verification game for fast transaction confirmation time and fast detection and penalization of malicious behaviors of attacking maternodes for strengthening security and safety. Section V details the reshuffling and data availability problem and solution. Finally, we conclude the paper and show some perspectives in Section VI.

II. SHARDING ARCHITECTURE

This section describes the sharding architecture and the used consensus protocol. We assume readers of this paper have basic understanding of our Proof-of-Stake Voting (PoSV) consensus protocol previously released and presented in [10]. However, to better describe the sharding architecture, we summarize some important information about TomoChain's PoSV.

Specifically, TomoChain features a voting-based system to elect 150 Masternodes that are responsible for creating, verifying and finalizing the blocks created within a period, called epoch, which lasts for 900 of two-second block-times. Each coin-holder must deposit at least 50 000 TOMO to a Voting smart contract and must satisfy a set of infrastructure requirements. The set of Masternodes is dynamically selected through votes made by coin-holders that send their tokens to the Voting smart contract.

Sharding architecture overview

Figure 1 shows the proposed sharding architecture. Without loss of generality, we suppose there are *N* masternodes that are selected among the candidates. The network of masternodes is assigned to specific shards using randomization sampling. This latter randomly divides the set of *N* masternodes into sub-groups of size *c* (either 10 or 15), namely the shard size, thus the total number of shards *S* in the network is *N/c*. This step is called network sharding. This step is repeated every epoch. Each shard also has an identifier numbered from 0 to (*S – 1*).

Each shard/subgroup only stores a portion of the whole blockchain, instead of the whole blockchain as in existing blockchains such as Bitcoin and Ethereum. The processing of a transaction assigned to a specific shard is based on the address of the sender of the transaction. A transaction *tx* is assigned to shard *id* if and only if *log2(N)* rightmost bits of the transaction sender's address is equal to *id*. That means, each external owned account is managed by a specific shard, based on its address. Therefore, simple double-spending attacks by sending transactions to different shards are avoided.

For example, transactions from address *0xabc..def* and *0xabc...ded* are processed by shard 15 (16th shard) and 13 (14th shard), respectively, if the total number of shards *N = 16*.

There is also a root chain that interacts with shard chains. The root chain is used for securing the transactions in the shard chains. The root chain does not store details of transactions assigned to shards, but block hashes and the smart contracts used for voting and block finalization.

TomoChain features a Voting smart contract that is for masternode candidates and coin-holders to deposit and vote for masternodes, respectively. In the sharding architecture, this smart contract is deployed on the root chain. Coin-holders interact with the Voting smart contract in the current TomoChain as they interact with the root chain in the sharding architecture. Every vote and deposit are recorded in the root chain.

The blocks in the root chain are created and verified by a larger number of masternodes than shard chains' for securing the whole system (see Consensus protocol for more detailed discussion). Ideally, all masternodes are responsible for the root chain, just as they are maintaining TomoChain in PoSV. An interesting point is that the set of masternodes for the root chain does not need to do computation for transactions of specific shards, but only for creating blocks containing transactions to the consensus smart contracts (see [11] for detailed discussion). That means, the computation throughput is much smaller than that of the current PoSV because most of transactions happen in the shard chains. All masternodes will be creating and verifying

blocks in the root chain and a shard chain, but there is no slot of two-second within which a masternode must create blocks for both the root chain and a shard chain. This is because it is very hard for a masternode to create two blocks within 2s.

It is worth noting that, in order for attackers to create another longer shard chain than an existing shard chain, the attackers need to successfully revert both the shard chain and the root chain, which are unlikely to happen since the root chain is secured by all masternodes.

Shard assignment through randomization sampling

As a reminder, masternodes in TomoChain in an epoch is selected at the end of the previous epoch and there is also a randomization process during the previous epoch in order to select block verifiers for double validation. The addresses of the set of masternodes for next epoch is recorded in the Voting smart contract in a decreasing order of total votes for them. Each masternode is identified by its deposit wallet address. The goal of the shard assignment is to divide the address-identified *N* masternodes into subsets of masternodes, each of which process a disjoint set of transactions. To do that, an additional randomization process is executed during the previous epoch for shard assignment.

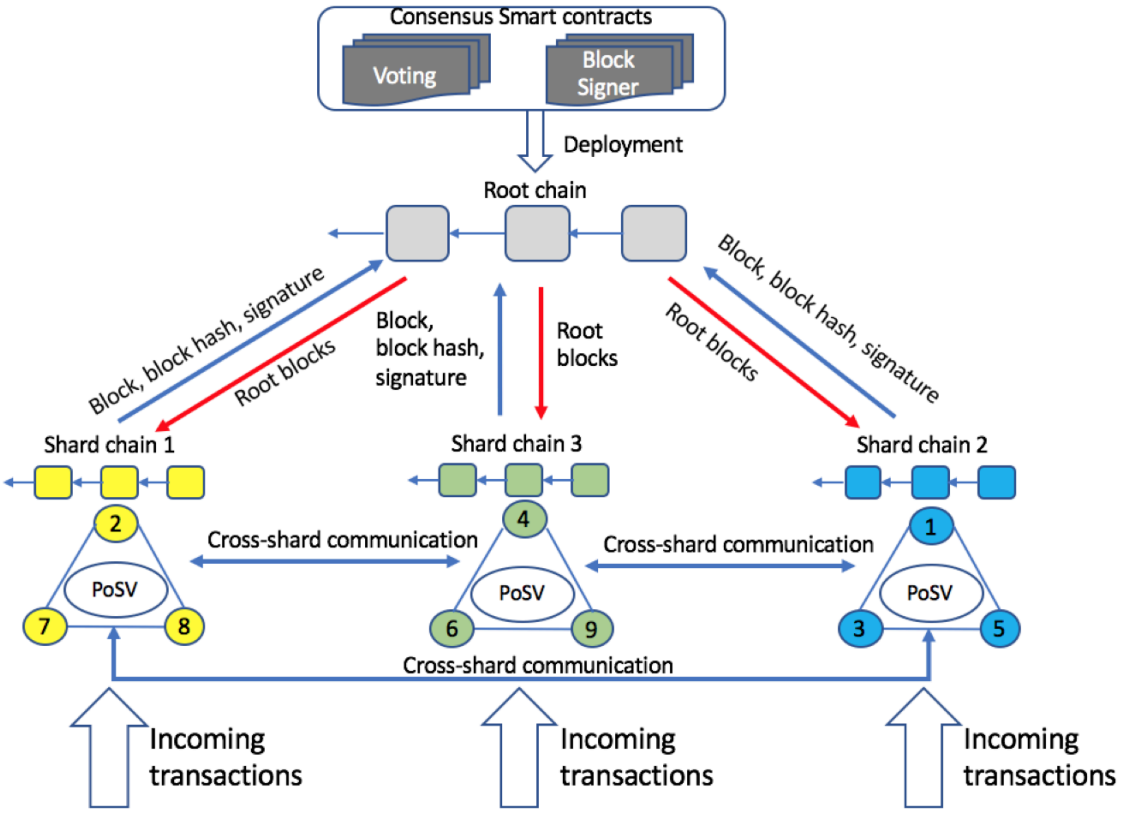

Figure 1: The proposed sharding architecture solution

The shard assignment is realized through randomization sampling that executes a pseudo random number generator function with a random seed agreed between the masternodes. We utilize the decentralized randomization algorithm that was previously introduced in our TomoChain Proof-of-Stake Voting consensus.

The shard assignment has the following steps:

- **Randomization**: This step follows our previously proposed decentralized lock-recovery randomization scheme to compute a random seed which is unpredictable and not biased by any masternode.
- **Shard assignment computation**: Based on a random seed, we generate *N* random numbers using a pseudo random number generator. The *N* numbers are used in a permutation algorithm for sampling *S* sets of *c* masternodes.

Figure 1 shows an example of the result of the decentralized randomization sampling process. The shards are communicated with each other through a cross-shard communication scheme, which is presented in the next section. Each shard sends their created blocks, block hashes and block signatures to the root chain that stores them in the TomoChain block signer smart contract (see Section Consensus protocol for more information).

Consensus protocol

In PoSV, there are currently two consensus smart contracts deployed onto TomoChain:

- Voting smart contract: This contract allows coin-holders to deposit (to become a masternode candidate), to vote for masternodes, to resign from the candidates list, to un-vote for masternodes, and to withdraw voted and deposited tokens.
- Block signer smart contract: Once a masternode verifies a block, the masternode signs off the block and sends the signature to the block signer smart contract.

In the sharding architecture, these two smart contracts are deployed onto the root chain. Since the latter is verified by many masternodes, the smart contracts are secured.

Proof-of-Stake Voting (PoSV) consensus with double validation and randomization is run among the set of masternodes of a shard. As a reminder, PoSV provides fast block-time and confirmation time, and a double validation technique for security enhancement.

A block in PoSV is verified by both block creator, which creates the block, and block verifier, which verifies the block before adding it to the blockchain (see [10] for more information about block creator and block verifier in double validation). Other masternodes will then check whether the block verifier signs off on the block. Each masternode verifies a block by sending its signature

for the block to the block signer smart contract on the root chain. A block in a shard is confirmed if ¾ masternodes of that shard verifies and signs it off. The confirmation time for intra-shard transactions can be almost instant, just as what is provided by PoSV without sharding.

The root chain does not need to do re-computation for verifying blocks created by shard chains. The root chain is secure since it is maintained by all masternodes in the network. A double-spending attack to a shard needs to revert both the shard chain and the root chain, which is unlikely to happen.

There is a small probability that a shard is colluded, in which more than its ¾ masternodes are attackers that do an adaptive attack. These attackers can create invalid blocks to create, for example, money out of thin air. In order to deal with this issue, we provide a game theory incentive-driven approach that is similar to Plasma. This approach is presented in Section IV.

III. CROSS-SHARD TRANSACTION SCHEMES

Cross-shard transaction

A cross-shard transaction involves sending some coins from a sender in one shard to a receiver in another shard. It enables any account in a shard to transact with any other account in other shards. Our **Lock-Commit** cross-shard transaction requires cross-shard communication and is processed in two steps as follows:

- **Lock at sending shard**: The transaction is first processed by the sender shard which decreases the balance of the sender account. At the end of this phase, a lock transaction receipt is generated which is signed by the masternode which produces the block containing the transaction. The receipt is only generated if the transaction has been finalized[1] by the shard (the block is verified and signed by ¾ the number of masternodes of the shard sending their signatures to the Block Signer smart contract on the root chain). It plays the role of a **Proof-of-Acceptance** which is then sent to the receiver shard for processing. The **Proof-of-Acceptance** consists of the transaction, the Merkle proof of the transaction in the block at the sender shard, and the finality proof of the block.
- **Commitment at receiving shard**: The leader of the receiver shard sends the **Proof-of-Acceptance** for the transaction at the lock phase to the receiver shard. This latter processes the proof as a transaction which might increase the balance of the receiver. An issue of this commitment phase is that the malicious masternodes in the receiver shard might ignore the commit transaction, thus leaving the whole transaction stuck (because the sender's balance is decreased while the receiver's balance is not

[1] A block is considered as "finalized" meaning that it is irreversible and any attempt to modify it is refused by the network.

increased). However, because masternodes take turns to create blocks in a round-robin manner following the PoSV consensus, the proof will eventually be added to a block created by an honest masternode, even though one or several malicious masternodes might ignore the proof transaction. This liveness property is discussed in details in our previous technical paper for PoSV consensus.

Figure 2 shows an illustrative example of a smart contract call cross-shard transaction. One of the complications of this transaction scheme is how to refund the left gas after the execution of the smart contract at the receiver shard completes. The left gas should be returned back to the sender (User) of the transaction in the sender shard. With that being said, in order to completely confirm a smart contract call cross-shard transaction, one must wait until all these back-and-forth phases between these shards are confirmed one-by-one. Therefore, the transaction confirmation latency increases, which is not expected from user perspectives, despite the significant increase of transaction volume.

To deal with this issue, we propose another cross-shard transaction scheme, namely **transfer-first cross-shard transaction scheme** (TFC). In this latter, instead of directly doing a smart contract call cross-shard transaction, the sender at one shard can:

- Send the expected amount to another external account that is managed by herself/himself at the receiver shard through a simple balance transfer. Note that, this latter still requires the **Lock-Commit** scheme to make simple balance transfer. However, **Lock-Commit** scheme for simple balance transfer does not involve the complexity of gas refund as for smart contracts. This is because the **Lock-Commit** scheme for cross-shard simple balance transfer will consume a fixed amount gas (the gas for two balance transfers or 2300*2 gas).
- Use the controlled external account to interact with all smart contracts in the receiver shard. All transactions started by this external account within the receiver shard will then be confirmed almost instantly as provided by the PoSV consensus.

While this transfer-first cross-shard transaction scheme helps maintain the fast transaction confirmation property of PoSV, it raises another issue: a user must manage as many accounts as the number of shards in the system in order to be able to instantly interact with all smart contracts and decentralized applications deployed on all shards. In order to alleviate this complicated account management issue, we intend to integrate an account management layer into our TOMO wallet. This layer is to help users manage their accounts in different shards as a single account in the current TomoChain design. It brings transparency to users as they are interacting with a one-shard system. When a user wants to make a transaction, the wallet does as follows:

- The account management layer specifies the target shard ID, based on the address of the receiver of the transaction.
- If the receiver is an external owned account, a simple transfer transaction is created and broadcast to the masternodes network.
- If the receiver is a smart contract, the account management layer verifies whether the address *A0* corresponding to the target shard has the expected amount of token for the transaction.
- If the balance is not sufficient, a transfer transaction from one of the accounts in other shards to the address *A0*. Then, the transaction to the smart contract is created and sent to the masternodes network.

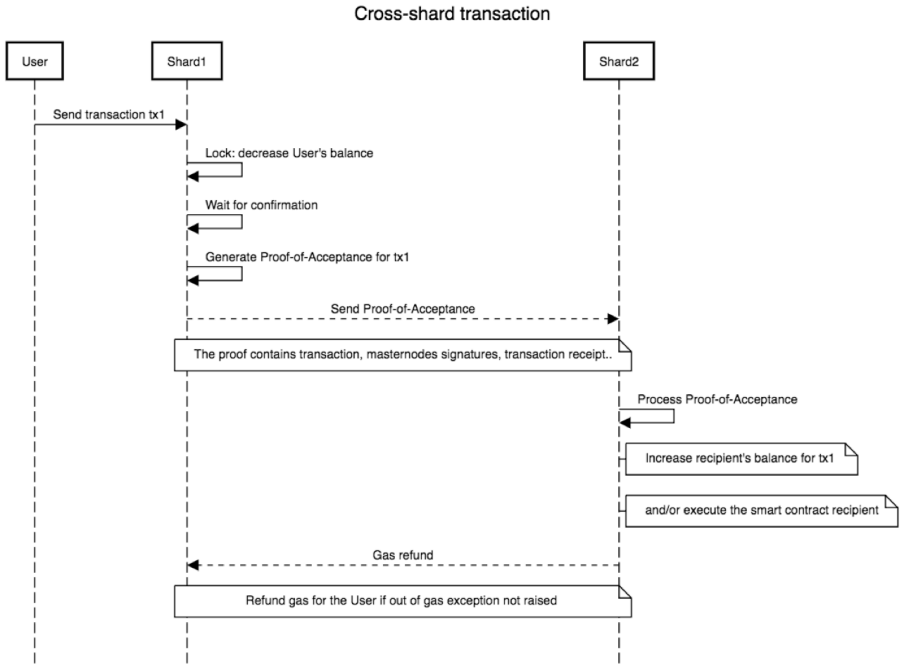

Figure 2. Smart contract call cross-shard transaction

One issue with this cross-shard transaction scheme is that the transaction fee would be higher than the non-sharding blockchain system since any cross-shard transaction must pay an additional fee at the sender shard of the transaction for the **Lock** at sending shard phase. However, the issue can be alleviated if the scheme is implemented along with a near-zero fee transaction system such as TomoChain.

Smart contract message calls chain

A smart contract, executed within a shard, might call another smart contract managed by another shard in the middle of its execution. Many smart contracts can be involved in a smart contract

message calls chain. This is a very challenging problem because the execution of a caller smart contract in a caller shard cannot suspend and wait for the completion and finality of the callee smart contract in a callee shard. If that is the case, both the caller and callee shards would be suspended, thus significantly decreasing the performance of the whole system. The problem might be more harmful if the smart contract message calls chain involves many shards.

Let's illustrate this challenge by the example in Figure 3 with the corresponding solidity source code in Figure 4. Suppose a user sends a transaction to a smart contract *SC1* in order to call method *m1* of *SC1*. The execution of the latter in Shard1 executes the method *m2* of *Shard2*, which in turn makes another message call to *SC3* in *Shard3*. For simplification, we assume that the user's address is managed by *Shard1*. Otherwise, the user can use the *TFC* cross-shard transaction scheme presented previously.

The *TFC* cross-shard transaction scheme previously presented in Cross-shard transaction is not applicable to the illustrative example. The reason is that, the method calls between smart contracts are not transactions, but message calls. Message calls are not directly written to the blockchain. Even if we can wisely consider message calls as transactions in this case, the message call *m1* to *m2* executed in Shard1 must wait for the message call in Shard2 to be confirmed, which in turns needs to wait for the message call in *Shard3* to be confirmed. This **wait-for-confirmation** chain critically decreases the performance of the system since the operations of waiting shards are blocked.

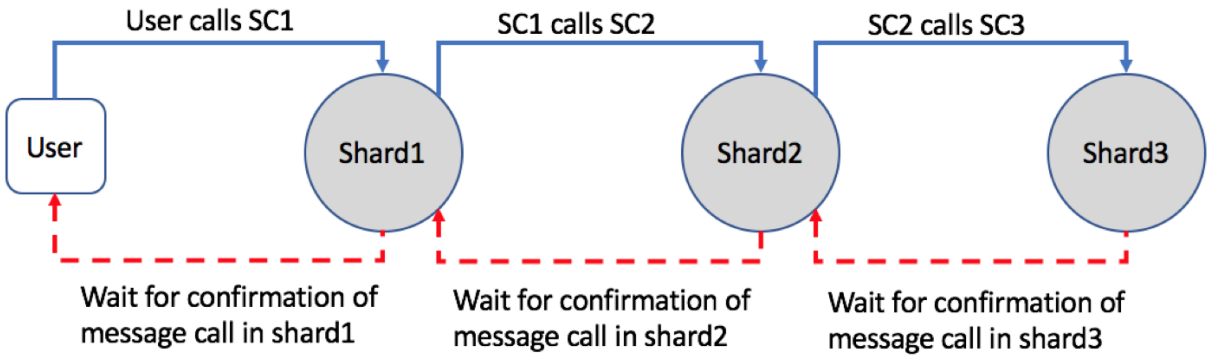

Figure 3. Smart contract message calls chain example

```
contract SC1 {
    bool callableSC2;
    SC2 sc2;
    constructor (SC2 _sc2) public {
        sc2 = _sc2;
    }
    function m1() external {
        if (callableSC2) {
            sc2.m2();
        }
    }
}
```

```
contract SC2 {
    SC3 sc3;
    constructor (SC3 _sc3) public {
        sc3 = _sc3;
    }
    function m2() external {
        sc3.m3();
    }
}
contract SC3 {
    function m3() external {
        //Do something
    }
}
```

Figure 4. Solidity example for smart contract message calls chain

In order to deal with this problem, we propose to group all smart contracts in a message calls chain and deploy them onto the same shard. With that being said, for the illustrative example, the smart contracts *SC1*, *SC2* and *SC3* are all deployed onto the shard onto which *SC3* is deployed. This is because *SC3* is the first contract among the three to be deployed.

By this way, any smart contract message calls chain transaction is always executed within one single shard. Therefore, the aforementioned problem is eliminated.

There is one issue that this solution cannot deal with is when a contract *A* calls two independent contracts *B* and *C*, which are respectively previously placed onto different shards. We approach this issue by early detecting this problem through an off-chain smart contract analysis when the user wants to deploy this smart contract *A* to the network. Then, an exception is raised that does not allow to place *A* onto the network.

IV. INCENTIVE-DRIVEN MUTUAL VERIFICATION GAME FOR SAFETY AND SECURITY

Sharding is usually considered when the system has many participating nodes. This is because higher number of nodes per shard decreases the probability that a shard is colluded meaning that invalid blocks created by a colluded shard can be finalized.

In order to make sharding applicable to blockchain systems with smaller number of nodes per shard, i.e. 10 nodes per shard, while still maintaining its security, we propose to use a game theory incentive-driven mechanism which is currently utilized by Plasma. Concretely, a shard is

monitored by one or multiple challengers. The latter can be either independent nodes or masternodes in the root chain. They act as full nodes and receive all transactions and blocks processed by the monitored shard and verify all of these blocks. If a processed transaction is detected as invalid, i.e. the transaction is mal-formed or creates money out of thin air, the challenger sends a proof-of-invalidation or **fraud proof** to the root chain. The incentive for the challenger is that if a transaction is successfully challenged, the challenger gets rewarded as the total deposited tokens of the masternodes that have validated the invalid block. In contrast, all of the masternodes that have verified the invalid block are penalized by losing all of their deposited tokens to the challenger. Note that, non-masternode challengers do not have to deposit any amount of tokens.

There are two safety concerns about this incentive-driven approach:

- **Motivation for challengers**: Non-masternodes challengers are expected to actively monitor the network to get incentives for invalid blocks. However, in the long term, non-masternodes are only motivated for monitoring the network if there are occasional invalid blocks so that the non-masternodes challengers can compensate for the cost of operating and monitoring the network. If there are no invalid blocks for a long period, the challengers will eventually be discouraged for this monitoring task, thus leaving a safety issue for the attackers in a colluded shard to create invalid blocks. The latter then can be falsely finalized because all non-masternodes challengers turn off their monitoring task since they cannot pay operational costs for monitoring the network.
- **Data availability**: Data availability is a strict requirement for the challengers to be able to detect invalidity. Without data, it is impossible to claim fraud proofs. For example, when a new shard block is created, any challenger needs the shard block, the previous state of the shard chain, and the signatures of the attacker masternodes verifying the invalid block, in order to have enough evidence to penalize the attackers. Therefore, all attackers would choose to refuse to broadcast their created blocks to honest nodes. This is because if a shard is colluded, all of the attackers within that shard can finalize an invalid block without sending any block data to honest nodes.

We tackle the two aforementioned concerns as follows:

- Masternode candidates can become challengers to monitor the network and get incentives if some masternodes verify an invalid block. These candidates have incentives to monitor the network to detect the malicious behavior since whether having monitored or not, they still have to pay operational and monitoring costs. Furthermore, monitoring the network is also one of the ways to show its performance for coin-holders to vote it to become a masternode. In addition, if a masternode candidate chooses to not receive shard blocks and later on it is elected as a masternode assigned to a shard, the elected masternode will not be able to verify or create any shard blocks, which in turn lowers its

performance. Eventually, the elected masternode will be quickly voted out of the masternode list by the coin-holders.

- Shard blocks and masternodes' signatures produced by maternodes need to be propagated to the root chain in order to finalize the blocks. All masternodes then have chance of verifying the validity of the shard blocks to detect the malicious behavior and get incentives. We call this mechanism an incentive-driven mutual verification game. If one shard is colluded and an invalid block is produced by attackers, whether the attackers in the shard want to or not, they have to send the invalid block to the root chain to finalize. Exposing an invalid block to the root chain is extremely risky for the attackers to lose all of their deposits[2]. It is worth noting that, the attackers have no incentives to not propagate invalid blocks to the root chain. This is because, by doing so, the attackers have spent computation resources and operational costs for only locally postponing the transactions within one shard. Furthermore, the attackers will be voted out of the masternode list because of their low performance and these postponed transactions will then be processed by the same shard but with a different set of masternodes once network reshuffling has been executed.

Fraud proofs: If a block with an invalid state transition is signed off and propagated throughout the network, any other participant who receives the block can submit a merkleized fraud proof to the Voting smart contract[3] on the root chain and the shard chain rejects the invalid block and is rolled back.

Fraud proofs ensure that all state transitions are validated. Example fraud proofs are proof of transaction spendability (funds are available in the current shard), proof-of-state transition (including checking the signature for the ability a transaction can be validated and executed, proof of inclusion/exclusion across blocks, and deposit/withdrawal proofs.

In order for this construction to have minimal proofs, though, all blocks must provide a commitment to a merkleized trie of the current state, a trie of outputs spent, a merkle tree of transactions, and a reference to the prior state being modified.

The fraud proofs and the incentive-driven mutual verification game combined with double validation ensure that a coalition of participants is not able to create invalid blocks that can be finalized without getting penalized. This ensures that the challengers always have access to

[2] Note that, masternodes are not required to verify blocks submitted to the root chain (that makes computation throughput on the root chain significantly less than the shard chains'). Masternodes can propagate the shard blocks submitted to the root chain to the masternode candidates or any challenger that will verify the validity of the shard blocks, which solves the data availability problem in this case.

[3] Note that, the Voting smart contract contains all deposits and voted tokens, therefore, the penalty for masternodes creating invalid blocks will be executed by a function within the contract. The penalty function is executed once a fraud proof for an invalid block is submitted by any nodes.

shard blocks, thus are able to prove (and therefore discourage) invalid state transitions in shard blocks.

Compared to TrueBit and Plasma: TrueBit [11] and Plasma [5] both use an incentive-driven verification game approach to provide scalable and secure solutions. Similar to our argument on the incentives for challengers to observe the network, the authors of TrueBit argue that, if there is no chance of finding a bug in the blocks, challengers will likely refuse to continue the monitoring task. In order to motivate the challengers to actively observe the network, TrueBit proposes to provide "*forced errors*". These errors are produced at random and unpredictable time. The main objective of blocks with forced errors is to incentivize the challengers and keep them motivated to secure the network. These blocks are special and the creators and validators of these blocks are not penalized.

Plasma provides an "*exit*" mechanism that aims at resolving the data availability problem. Specifically, in Plasma, when a participant cannot access block data of a Plasma chain, the participant can move her asset/token from the Plasma chain to the root chain (or the parent Plasma chain) by executing the "*exit*" mechanism. The participant then needs to wait for an amount of time, i.e. 1 week, in order for other participants to submit a fraud proof if the exit execution is falsified.

In order to maintain fast confirmation time as in the current TomoChain PoSV design and to have the shards' liveness, our approach is driven by an incentive and mutual verification game. The latter aims at providing both fast confirmation and detection of Byzantiness of a block.

By utilizing the very strong incentivizing-penalizing incentive-driven mutual verification game, challengers are strongly encouraged to work hard in order to keep the system safe and secure and shard masternodes are discouraged to act maliciously.

V. RESHUFFLING AND DATA AVAILABILITY

To enhance further the security of shards, it is very important that shards should be dynamic for resilience against attacks and failures of nodes in a shard. For example, one shard might be stuck in a situation where blocks are valid but cannot be finalized because attacking masternodes do not validate these blocks. A technique for dealing with this problem is to reshuffle the masternodes assigned to the shards. This means the set of masternodes for each shard is changed dynamically every epoch. At the end of each epoch, the randomization sampling is repeated for a different set of masternodes elected to participate into shards.

Reshuffling enhances the security and resilience of the system. However, it introduces a new challenge: the data availability problem. In the latter, every masternode joining to a new shard

must synchronize the blockchain portion of this new shard in order to be able to verify transactions belonging to it. If the portion is too large and the synchronization time is too short, the entire shard cannot verify any transaction, thus renders the shard useless for the whole synchronization time.

This problem can be tackled by the following strategies::

- Shard masternodes storing all states of other shards: It requires that all masternodes, even non-shard member nodes, must receive and store all of the portions of state in all shards. This solution allows for any masternodes to smoothly switch from one shard to or join any other shard without needing a data synchronization time.
- Gradual reshuffling: Instead of resampling the whole set of masternodes assigned to each shard at the end of each epoch, the reshuffling might only pull a subset of masternodes out of a shard while leaving the remainder intact, i.e. 1/10 of the masternodes are kicked out from a shard while the other masternodes still stay in the shard and the resampling takes *c/10* (c is the shard size) masternodes randomly from the set of masternode candidates. Since the latter have been synchronizing all blocks (see Section IV for more information), the system can smoothly continue the operations without interrupting time for data synchronization.
- Randomization sampling in advance: Still using the strategy of giving masternodes joining new shards some amount of time to download the blockchain data of a shard, this approach lets the network execute the randomization sampling in the middle of an epoch. Therefore, assigned masternodes will know ahead of time to which shard they will contribute in the next epoch. In this fashion, these in-advance assigned masternodes will have enough time to download the blockchain data of the assigned shard to prepare for the next epoch.

Currently, we have been leaning towards the first and second strategy because these solutions are more implementable and analyzable for security. These strategies will be rigorously analyzed before making a final decision since both have its own advantages and limitations. While the first strategy has advantage of simplicity and masternodes and candidates can switch from one shard to any other shard smoothly, the storage and network bandwidth of masternodes might be more needed. On the other hand, the second strategy can save more storage and network bandwidth, it raises a question: which masternodes should be kicked out of a shard if the performance of all masternodes in that shard is efficiently equal to each other's.

VI. CONCLUSION

The paper has presented a new solution to sharding architecture for public blockchains. It aims at being transparent to users and significantly improving the transaction processing performance,

while still maintaining basic security requirements of the system. The paper uniquely identified and solved cross-shard transactions for smart contract call chaining and data availability problem in shard reshuffling. The proposed solution uses the Proof-of-Stake Voting (PoSV) efficient consensus protocol proposed by Tomochain for intra-shard consensus. Specifically, we relied on our decentralized randomization algorithm for randomized shard assignment of masternodes and proposed two cross-shard smart contract transaction schemes combined with a new smart contract deployment strategy. The latter places dependent smart contracts onto the same shard, which eliminates smart contract message calls chain problem involving multiple shards. In order to ensure safety and security of shards, we proposed an incentive-driven mutual verification game. In this latter, masternodes that create and/or verify an invalid block are penalized by losing all of their deposits to any participant that detects the invalidity by providing fraud proofs. Moreover, we also discuss the data availability problem in state sharding when shard reshuffling takes place and propose several possible solutions.

We are currently analyzing rigorously the safety and security properties of the proposed architecture to show its soundness. In future, once the analysis will have been done, we will implement the proposed architecture on top of the TomoChain blockchain.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” p. 9.

[2] G. Wood, “Ethereum: a secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, pp. 1–32, 2014.

[3] J. Poon and T. Dryja, “The Bitcoin Lightning Network:,” p. 59.

[4] P. Mccorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, “Pisa: Arbitration Outsourcing for State Channels.”

[5] J. Poon and V. Buterin, “Plasma: Scalable Autonomous Smart Contracts,” 2017.

[6] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A Secure Sharding Protocol For Open Blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*, 2016, pp. 17–30.

[7] P. Barrett, “Zilliqa Technical Whitepaper,” *Zilliqa*, pp. 1–8, 2017.

[8] M. Castro, M. Castro, B. Liskov, and B. Liskov, “Practical Byzantine fault tolerance,” *OSDI {’}99: Proceedings of the third symposium on Operating systems design and implementation*, no. February, pp. 173–186, 1999.

[9] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and E. Syta, “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding,” *IEEE Symposium on Security & Privacy*, 2018.

[10] Tomochain R&D Team, “Tomochain: Masternodes Design Technical White Paper Version 1.0,” Technical White Paper, 2018.

[11] J. Teutsch and C. Reitwießner, “A scalable verification solution for blockchains,” p. 50, 2017.

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