Boson Consensus: A Scalable Blockchain Consensus Algorithm

Qi Zhou

QuarkChain Foundation LTD., Singapore, Singapore

qizhou@quarkchain.org

v0.1.0

Abstract

One of the major challenges for blockchain networks is to achieve scalability while maintaining security and decentralization. Among the existing solutions to enhance scalability, blockchain sharding is one of the promising solutions by running multiple chains concurrently. However, the major issue that blockchain sharding faces is the vulnerability to a single-shard attack, where the security guarantee of a shard can be much lower than a single chain. In this paper, we introduce Boson consensus algorithm - a blockchain sharding consensus algorithm that scales the network dramatically while keeping security guarantee. We discuss how Boson consensus increases blockchain capacity by increasing the number of shards and enables both efficient in-shard and cross-shard transactions. Moreover, to ensure security, we demonstrate that all shards could share the same security guarantee as the root chain, whose security level is much stronger than any single shard in multi-chain networks. Finally, we illustrate the first realization of Boson consensus - QuarkChain, and show that a community-run testnet could achieve more than $55,000$ transactions per second.

Index Terms

Consensus, proof of work, proof of stake, sharding, cryptocurrency, scalability

I. Introduction

Since the debut of Bitcoin [1], Bitcoin and blockchain technology have received great attentions due to the merits such as decentralization, immunity, transparency, etc. However, one major drawback of the Bitcoin network is the low capacity in terms of transactions per second (TPS), where Bitcoin offers about 6 to 7 TPS. Compared to centralized systems such as VISA or Alipay,

which could achieve more than tens of thousands of TPS, the low TPS of Bitcoin results in high transaction fee and thus greatly limits its wider applications.

The performance of Bitcoin network is mainly determined by two parameters - maximum block size and block interval, where Bitcoin limits the block size to 1 megabytes (MB) and adjusts the difficulty of its proof-of-work (PoW) consensus so that the expected block interval is 600 seconds. To improve the performance, several blockchain networks are proposed to optimize the two parameters. Litecoin [2] decreases the block interval to 150 seconds and claims 4 times performance improvement. Bitcoin Cash [3] increases the block size limit to 32 MB and thus claims 32 times performance improvement. However, decreasing the block interval and/or increasing block size can significantly increase block stale rate [4]. As a result, a portion of the hashpower of the network does not contribute to the canonical chain and is wasted, and then the security of the network against double-spending attack, a.k.a, 51% attack, is weakened. To alleviate the security issue of the wasted hashpower, Ethereum adopts greedy heaviest observed subtree (GHOST) [5], [6] algorithm to collect the hashpower of the stale blocks and reduces its block interval to about 15 seconds. To further reduce the block interval and improve performance, delegated proof-of-stake (dPoS) consensus [7] is proposed to limit the network size and allows only 10-30 block producers (BPs) in the network. Although the block interval can be decreased to seconds or sub-seconds, dPoS is criticized to be prone to centralized due to the limited number of block producers.

Another approach to scale blockchain networks is to run multiple blockchains simultaneously, which is sometimes referred to as blockchain sharding. By increasing the number of chains (or shards) running in parallel, the scalability of the network increases linearly. However, this approach also faces several challenges. One challenge is that a single shard is more vulnerable to attack, namely, single-shard attack. For example, suppose there are N shards in the network and all shards run PoW with the same hash algorithm. Then there exists a shard with at most $1/N$ hashpower. As a result, an attacker could perform a double-spending attack with only $1/2N$ hashpower, which is much lower than that of a single-chain network. Besides the security concern, another challenge is to enable cross-shard interoperability efficiently and securely.

In this paper, we present Boson consensus algorithm - a novel approach to scale blockchain network dramatically. Boson consensus algorithm employs blockchain sharding so that the network capacity increases as the number of shard chains increases. In addition, to counter single-shard attack, Boson consensus algorithm runs a root chain that describes the canonical

chain of each shard and is designed to have strong security guarantee, e.g., high hashpower of the network if the root chain runs PoW consensus. By exploiting a root-chain-first fork-choice rule on each shard, we prove that the security of the double-spending attack on shard chains can be efficiently guarded by the root chain, and thus the double-spending attack on a shard becomes much harder than the conventional blockchain sharding design. In addition, we illustrate how to add a new shard dynamically in a Boson network and how to perform cross-shard transactions efficiently and securely.

Besides great scalability, Boson consensus algorithm also allows different shards running different combinations of consensus, ledger, transaction model, and even token economics. This gives Boson network a great flexibility and is able to employ different techniques of existing blockchains such as Bitcoin's unspent transaction output (UTXO) and scripting transaction model, Ethereum account-based model and virtual machine (EVM), privacy-preserve functions in a Boson network. Furthermore, these shard chains can easily inter-operate, e.g., an owner of a UTXO can call the smart contract on another shard, or a balance in an EVM-like shard could perform a confidential transaction on a shard with privacy-preserve functions. With the ability to add new shards, the network can be further upgraded to adopt emerging blockchain innovates such as new virtual machines, ledger models, etc.

The rest of papers are organized as follows. Section II introduces the system model of conventional blockchain network. Section III proposes Boson consensus algorithm. Section IV illustrates the first realization of Boson Consensus - QuarkChain. Section V concludes the paper.

Notations:

Notation	Description
$\mathbf{B} = [B_0, ..., B_l]$	A list of chained blocks
$\mathcal{C} = \{(N_i, N_j)\}$	Connectivity set of the network
$\mathbb{C}$	Consensus algorithm
$d(\cdot)$	Difficulty of a block
$h(\cdot)$	Cryptographic hash function
$\mathbb{L}$	Ledger of Boson network
$\mathbb{N}$	A blockchain network
$\mathcal{N} = \{N_i\}$	The set of nodes in the network
P	P2P network protocol
$\mathbb{S}$	State of a ledger
$T(\cdot, \cdot)$	State transfer function
$V(\mathbf{B})$	Validity function
$W(B)$	Simplified validity function

TABLE I

NOTATIONS USED IN THE PAPER

II. SYSTEM MODEL OF CONVENTIONAL BLOCKCHAIN NETWORK

Consider a blockchain network

$$\mathbb{N} = (\mathcal{N}, \mathcal{C}, \mathbb{C}, P), \tag{1}$$

where the nodes of the network $\mathcal{N} = \{N_i\}$ are inter-connected by a P2P protocol P, $\mathcal{C} = \{(N_i, N_j)\}$ is the connectivity set for all pairs of nodes N_i and N_j that are connected in the network, and $\mathbb{C}$ is the consensus algorithm. Each node N_i maintains a list of blocks

$$\mathbf{B}^{(i)} = [B_0^{(i)}, B_1^{(i)}, ..., B_l^{(i)}], \tag{2}$$

namely, a ledger, where $B_0^{(i)} = B_0$ the genesis block that is known and fixed for all nodes, $B_j^{(i)}$ is the block with index j, and the number of blocks in the ledger is $l + 1$.

The consensus $\mathbb{C}$ contains two components

$$\mathbb{C} = (V(\cdot), F(\cdot, \cdot)) \tag{3}$$

where $V(\cdot)$ is the validity function, which determines a ledger $\mathbf{B}$ is correct or not, and $F(\cdot, \cdot)$ is the fork-choice rule function. Note that the consensus we defined here is in a general form, which can describe most blockchain consensus algorithms such as proof of work (PoW), proof of stake (PoS), delegate proof of stake (dPoS), etc.

To incentivize each node to operate the network, each block is associated with a block reward assigned to the producer, and any node could produce a block as long as it is defined validly by $V(\cdot)$. Once a node produces a block, it will immediately broadcast the block to its neighbors via the P2P protocol P so that the block (and the producer's reward) could be accepted by others. Meanwhile, if another block is produced at the same height and results in different ledgers, namely, forks, the network reaches an inconsistent state temporarily, and we need a fork-choice rule $F(\cdot,\cdot)$ to determine which fork to survive. We will explain the details of the validity function and fork-choice rule in the following subsections.

A. *Validity Function*

The validity function $V(\cdot)$ accepts a list of blocks as input and outputs the validity of the blocks as

$$V(\mathbf{B}) = \begin{cases} 1, & \text{if } \mathbf{B} \text{ constitutes a valid ledger} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

A node will reject the blocks if $V(\mathbf{B}) = 0$. In addition, by properly designing the blocks, we could further simplify the validity function as follows.

- In a blockchain network, the validity function requires that the blocks in a ledger are chained via hash pointers. This means that, for each block, it contains a field *prev_hash*(B_i) that returns the hash value of the previous block and a valid ledger $\mathbf{B} = [B_0, B_1, ..., B_l]$ must satisfy

  $$h(B_{i-1}) = \textit{prev\_hash}(B_i), \forall i > 0 \tag{5}$$

  where $h(\cdot)$ is a cryptographic hash function defined by $V(\cdot)$. Assuming the hash function is collision-resistant, we could uniquely determine the history of a new block B_i, i.e., $[B_0, B_1, ..., B_{i-1}]$ by looking up the previous blocks associated with the hash pointers from the source of the new block (local or neighbor) recursively. In addition, given two blocks B_i and B_{i+n}, we called them n-hash-linked, i.e., $B_{i+n} \xrightarrow{n} B_i$, if and only if $B_i, B_{i+1}, ..., B_{i+n}$ are hash linked.
- The validity function only validates a list of blocks, which can be computationally expensive over time if the list is long. In most cases, we only need to validate whether a new block B_i is valid or not if the previous block is already validated. This can be done by using the

intermediate result of a ledger, namely, the state of the ledger with a state transfer function $T(\cdot,\cdot)$:

$$\mathbb{S}_0 = T(\emptyset, B_0) \tag{6}$$

$$\begin{aligned}\mathbb{S}_i &= T(\mathbb{S}_{i-1}, B_i) \\ &= T(T(\mathbb{S}_{i-2}, B_{i-1}), B_i) \\ &= T(T(...T(T(\emptyset, B_0), B_1)), ..., B_{i-1}), B_i),\end{aligned} \tag{7}$$

where $\mathbb{S}_i$ is the state associated with block B_i, $\mathbb{S}_0$ is the genesis state, and we could redefine the validity function as

$$\begin{aligned}V([B_0, B_1, ..., B_i]) &= W(B_i) \\ &= \begin{cases} 1, & \text{if } W(B_{i-1}) = 1 \text{ and} \\ & \quad B_i \text{ can be applied to } \mathbb{S}_{i-1}; \\ 0, & \text{otherwise.} \end{cases}\end{aligned} \tag{8}$$

where B_{i-1} and $\mathbb{S}_{i-1}$ are uniquely determined by the hash pointer *prev_hash*(B_i). This allows a node to quickly determine a validity of a block by applying it to the previous state instead of evaluating over all history of the block.

B. Fork-Choice Rule

The fork-choice rule function accepts two valid ledgers $\mathbf{B}^{(i)} = [B_0^{(i)}, B_1^{(i)}, ...B_l^{(i)}]$ and $\mathbf{B}^{(j)} = [B_0^{(j)}, B_1^{(j)}, ...B_m^{(j)}]$, and outputs which one should be chosen as the ledger of the network:

$$F(\mathbf{B}^{(i)}, \mathbf{B}^{(j)}) = \begin{cases} > 0, & \text{then } \mathbf{B}^{(i)} \text{ should be chosen,} \\ 0, & \text{anyone could be chosen,} \\ < 0, & \text{then } \mathbf{B}^{(j)} \text{ should be chosen.} \end{cases} \tag{9}$$

Note that in case of the fork-choice rule makes no preference ($F(\mathbf{B}^{(i)}, \mathbf{B}^{(j)}) = 0$), the node could choose any ledger, and a common practice is to choose the one that is firstly known to the node.

In addition, to incentivitize the nodes to produce blocks, we assume the fork-choice rule must have the following property

$$F(\mathbf{B} + [B], \mathbf{B}) > 0, \tag{10}$$

$$F(\mathbf{B}, \mathbf{B} + [B]) < 0, \tag{11}$$

where $+$ is the list concatenation operator and block B is a valid block appended to the ledger $\mathbf{B}$.

C. Example with PoW-Based Blockchain Network

In this subsection, we illustrate our blockchain abstractions using PoW-based networks such as Bitcoin and Ethereum as examples. In a PoW-based network, to produce a block, each producer first constructs a template block, which is validly defined by $W(\cdot)$ in Eq. (8) except that the block does not reach the associated difficulty. The difficulty is used to control the block produce rate so that the expected block rate is about 15 minutes and 15 seconds for Bitcoin and Ethereum networks, respectively. The difficulty of a block is adjusted for every 2016-block in Bitcoin network or every block in Ethereum. Then the block producer starts to mine the block by increasing a nonce field in the block until the hash of the block reaches the difficulty, i.e.,

$$W(B_i) = \begin{cases} 1, & \text{if } B_i \text{ is a valid template block and} \\ & \quad h_m(B_i) \times d(B_i) < 2^{256}, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

where $d(B_i)$ returns the difficulty of block B_i and $h_m(\cdot)$ is the hash function for mining, which is double SHA2 for Bitcoin and Ethash for Ethereum.

The fork-choice rule of the PoW-based network uses a term called "total difficulty", which is defined as

$$td(\mathbf{B}) = \sum_{\forall i} d(B_i), \tag{13}$$

and the fork-choice rule function becomes

$$F(\mathbf{B}^{(i)}, \mathbf{B}^{(j)}) = td(\mathbf{B}^{(i)}) - td(\mathbf{B}^{(j)}). \tag{14}$$

Note that with total difficulty as the fork-choice rule, to maximize the mining efficiency, a block producer will be expected to mine the block that could be appended to the ledger with the highest total difficulty - otherwise, the miner will race with other block producers that are working on the ledger with the highest total difficulty and may waste its hashpower.

1) Double-Spending Attack on PoW-Based Blockchain Network: A double-spending attack aims to revert a block B_j of current ledger $\mathbf{B} = [B_0, B_1, ..., B_{j-1}, B_j, B_{j+1}, ..., B_l]$ by producing a list of blocks $[B'_j, B'_{j+1}, ..., B'_{l'}]$, namely, the attacking fork, such that

1) The attack fork constitutes a valid ledger $V(\mathbf{B}') = V([B_0, B_1, ..., B_{j-1}, B'_j, B'_{j+1}...B'_{l'}]) = 1$; and
2) The attacking fork has higher total difficulty compared to the canonical one, i.e., $\sum_{i \geq j} d(B'_i) > \sum_{i \geq j} d(B_i)$.

Therefore, we will have a valid ledger $\mathbf{B}'$ with a greater total difficulty than the current ledger $\mathbf{B}$, $td(\mathbf{B}') > td(\mathbf{B})$. According to fork-choice rule in Eq. (14), the network will choose $\mathbf{B}'$ and thus the blocks $B_j, B_{j+1}, ..., B_l$ will be reverted and transactions in the reverted blocks will be rolled back. As a result, the spent crypto-currencies in the transactions of the reverted blocks in $B'_j, B'_{j+1}, ..., B'_{l'}$ can be spent again. In a PoW-based blockchain network, an attacker could always create such attacking fork as long as the attacker has more than 51% of the hashpower of the network.

III. Boson Consensus

A network running Boson consensus is a tuple

$$\mathbb{N} = (\mathcal{N}, \mathcal{C}, \mathbb{C}, P), \tag{15}$$

where the nodes of the network $\mathcal{N} = \{N_i\}$ are inter-connected by a peer-to-peer (P2P) protocol P, $\mathcal{C} = \{(N_i, N_j)\}$ is the connectivity set for any pairs of nodes N_i and N_j that are connected in the network. However, different from the existing blockchain networks, whose ledger is a list of chained blocks, the ledger of a Boson network is

$$\mathbb{L} = (\mathbf{B}_\mathrm{r}, \mathbf{B}_0, \mathbf{B}_1, ..., \mathbf{B}_{N-1}) \tag{16}$$

where $\mathbf{B}_\mathrm{r} = [B_{\mathrm{r},0}, B_{\mathrm{r},1}, ..., B_{\mathrm{r},l_r}]$ is the list of blocks of the root chain, N is the number of shard chains, and $\mathbf{B}_i = [B_{i,0}, B_{i,1}, ..., B_{i,l_i}], \forall 0 \leq i \leq N-1$, is the list of shard chains. Since we have $N+1$ chains, each chain could have their own consensuses, i.e.,

$$\mathbb{C}_\mathrm{r} = (V_\mathrm{r}(\cdot), F_\mathrm{r}(\cdot, \cdot)), \tag{17}$$

$$\mathbb{C}_i = (V_i(\cdot), F_i(\cdot, \cdot)), \forall 0 \leq i \leq N-1. \tag{18}$$

In addition, the root block describes the canonical chain of each shard chain by including the hash pointers or headers of the last shard block observed

$$B_{i,o_i(B_{\mathrm{r},j})} = observed_i(B_{\mathrm{r},j}), \tag{19}$$

where $o_i(B_{\mathrm{r},j})$ returns the highest index of the block of the ith shard chain included by root block $B_{r,j}$. In practice, this can be done by including the hash values of new shard blocks when producing a root block. Given function $o_i(\cdot)$, we define the validity function of a ledger as:

Definition 1 (Boson Consensus Validity Function)**.** *The validity function of a ledger generated by Boson consensus is:*

$$V(\mathbb{L}) = \begin{cases} 1, & \textit{if } V_i(\mathbf{B}_i) = 1, \forall i \in \{\mathrm{r}, 0, ..., N-1\} \textit{ and} \\ & \quad o_i(B_{\mathrm{r},j-1}) \leq o_i(B_{\mathrm{r},j}),\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & \quad observed_i(B_{\mathrm{r},j}) \in \mathbf{B}_i,\ \forall 0 \leq i \leq N-1, j \geq 1 \\ 0, & \textit{otherwise}. \end{cases} \tag{20}$$

With the validity function defined in (20), we define the fork-choice rule of Boson network as follows.

Definition 2 (Boson Consensus Fork-Choice Rule)**.** *Given two different ledgers $\mathbb{L}^{(j)}, \mathbb{L}^{(k)}$, the fork-choice rule first compares the root chains of the ledgers:*

$$\begin{aligned} F(\mathbb{L}^{(j)}, \mathbb{L}^{(k)}) &= F_{\mathrm{r}}(\mathbf{B}_{\mathrm{r}}^{(j)}, \mathbf{B}_{\mathrm{r}}^{(k)}) \\ &= \begin{cases} > 0, & \textit{then } \mathbb{L}^{(j)} \textit{ should be chosen}, \\ 0, & \textit{see the following cases}, \\ < 0, & \textit{then } \mathbb{L}^{(k)} \textit{ should be chosen}. \end{cases} \end{aligned} \tag{21}$$

If $F_{\mathrm{r}}(\mathbf{B}_{\mathrm{r}}^{(j)}, \mathbf{B}_{\mathrm{r}}^{(k)}) = 0$, we have two possibilities.

- *The root chains are not equal, i.e., $\mathbf{B}_{\mathrm{r}}^{(j)} \neq \mathbf{B}_{\mathrm{r}}^{(k)}$. In this case, we have no preference on which ledger, and a common practice chooses the ledger that is firstly known to the node.*
- *The root chains are equal, i.e., $\mathbf{B}_{\mathrm{r}}^{(j)} = \mathbf{B}_{\mathrm{r}}^{(k)}$. This means that all the shard chains have common ancestors observed by the root chain, and thus each shard could apply their per-shard fork-choice rule*

$$F_i(\mathbf{B}_i^{(j)}, \mathbf{B}_i^{(k)}), \forall i \in \{0, ..., N-1\}, \tag{22}$$

where we denote $\mathbf{B}'_i$ *as the shard chain chosen*

$$\mathbf{B}'_i = \begin{cases} \mathbf{B}_i^{(j)}, & \textit{if } F_i(\mathbf{B}_i^{(j)}, \mathbf{B}_i^{(k)}) > 0 \\ \mathbf{B}_i^{(k)}, & \textit{if } F_i(\mathbf{B}_i^{(k)}, \mathbf{B}_i^{(j)}) > 0 \\ \mathbf{B}_i^{(j)} \textit{ or } \mathbf{B}_i^{(k)}, & \textit{otherwise.} \end{cases} \tag{23}$$

The resulting ledger becomes

$$\mathbb{L}' = (\mathbf{B}_{\mathrm{r}}^{(j)}, \mathbf{B}'_0, ..., \mathbf{B}'_{N-1}). \tag{24}$$

In summary, the root chain fork-choice rule overrides the per-shard fork-choice rule as the first step, and we call the fork-choice rule as ***root-chain-first fork-choice rule****.*

A. Simplified Validity Functions

Let us define a list of states for the root chain and shard chains, and the responding state transfer function as follows:

$$\mathbb{S}_{i,0} = T_i(\emptyset, B_{i,0}), \forall i \in \{\mathrm{r}, 0, ..., N-1\} \tag{25}$$

$$\begin{aligned} \mathbb{S}_{i,j} &= T_i(\mathbb{S}_{i,j-1}, B_{i,j}) \\ &= T_i(T_i(\mathbb{S}_{i,j-2}, B_{i,j-1}), B_{i,j}) \\ &= T_i(T_i(...T_i(T_i(\emptyset, B_{i,0}), B_{i,1})), ..., B_{i,j-1}), B_{i,j}), \forall i \in \{\mathrm{r}, 0, ..., N-1\} \end{aligned} \tag{26}$$

where $T_i(\cdot, \cdot)$ is the state transfer function of root chain or the ith shard chain defined by $V_i(\cdot)$, $\mathbb{S}_{r,0}$ and $\mathbb{S}_{i,0}$ are called genesis root state and the ith genesis shard state, respectively. In addition, we assume that every block has a hash pointer field *prev_hash*$(B_{i,j})$ containing the hash value of the previous block as

$$h(B_{i,j-1}) = \textit{prev\_hash}(B_{i,j}), \forall j > 1, i \in \{\mathrm{r}, 0, ..., N-1\}. \tag{27}$$

As a result, following Eq. (8), we could validate a new shard block by applying it to the previous state identified by the hash pointer of its previous block as

$$V_i([B_{i,0}, B_{i,1}, ..., B_{i,j}]) = W_i(B_{i,j}) = \begin{cases} 1, & \text{if } W(B_{i,j-1}) = 1 \text{ and} \\ & \quad B_{i,j} \text{ can be applied to } \mathbb{S}_{i,j-1}, \forall i \in \{0, ..., N-1\}. \\ 0, & \text{otherwise,} \end{cases} \tag{28}$$

and to validate a new root block, we have

$$W_{\mathrm{r}}(B_{\mathrm{r},j}) = \begin{cases} 1, & \text{if } W_{\mathrm{r}}(B_{\mathrm{r},j-1}) = 1 \text{ and} \\ & B_{\mathrm{r},j} \text{ can be applied to } \mathbb{S}_{\mathrm{r},j-1} \text{ and} \\ & observed_i(B_{\mathrm{r},j}) \xrightarrow{o_i(B_{\mathrm{r},j}) - o_i(B_{\mathrm{r},j-1})} observed_i(B_{\mathrm{r},j-1}),\ \forall 0 \le i \le N-1 \text{ and} \\ & W_i(B_{i,k}) = 1,\ \forall 0 \le i \le N-1, o_i(B_{\mathrm{r},j-1}) + 1 \le k \le o_i(B_{\mathrm{r},j}) \\ 0, & \text{otherwise.} \end{cases} \tag{29}$$

B. Double-Spending Attack

There are two types of block that an attacker may revert

- a root block $B_{\mathrm{r},j}$; or
- a shard block $B_{i,j}, 0 \le i \le N-1$.

According to the fork-choice rule in Eq. (21), it is straightforward that reverting a root block in a Boson network could be done by creating an attacking fork of the root chain. Meanwhile, reverting a shard block $B_{i,j}$ is less straightforward and depends on whether it is included by root chain, which is defined as follows.

Definition 3 (Root-Chain-Confirmed Shard Block)**.** *Given a Boson network ledger $\mathbb{L}$, the jth shard block of shard chain i, $B_{i,j}$, is **root-chain confirmed** if and only if $j \le o_i(B_{\mathrm{r},l_{\mathrm{r}}})$.*

Given the definition, we have the following propositions:

Proposition 1. *Given a Boson network ledger $\mathbb{L}$, if a shard block $B_{i,j}$ is not root-chain confirmed, the attacker could revert a shard chain block $B_{i,j}$ by creating an attacking fork $B'_{i,j}, B'_{i,j+1}, ..., B'_{i,l'_i}$ such that*

$$F_i(\mathbf{B}'_i, \mathbf{B}_i) > 0 \tag{30}$$

where $\mathbf{B}'_i = [B_{i,0}, B_{i,1}, ..., B_{i,j-1}, B'_{i,j}, B'_{i,j+1}, ..., B'_{i,l'_i}]$ and $V_i(\mathbf{B}'_i) = 1$.

Proof. By constructing another ledger $\mathbb{L}'$ as

$$\mathbb{L}' = [\mathbf{B}_{\mathrm{r}}, \mathbf{B}_0, \mathbf{B}_1, ..., \mathbf{B}_{i-1}, \mathbf{B}'_i, \mathbf{B}_{i+1}, ..., \mathbf{B}_{N-1}]. \tag{31}$$

Since $V_i(\mathbf{B}'_i) = 1$ and $B'_{i,j}, B'_{i,j+1}, ..., B'_{i,l'_i}$ are not root-chain confirmed, $V(\mathbb{L}') = 1$ holds true. Because both $\mathbb{L}$ and $\mathbb{L}'$ have the same root chain, by applying the fork-choice rule in Definition

2, local shard fork-choice rule will be applied and the result is $F_i(\mathbf{B}'_i, \mathbf{B}_i) > 0$, which means that $\mathbf{B}'_i$ will be chosen, and thus $B_{i,j}$ will be reverted. □

Proposition 2. *Given a Boson network ledger $\mathbb{L}$, if a shard block $B_{i,j}$ is root-chain confirmed, the attacker could revert a shard chain block $B_{i,j}$ only if the attacker also reverts all the root blocks $B_{\mathrm{r},k}$'s such that*

$$j \leq o_i(B_{\mathrm{r},k}). \tag{32}$$

Proof. Suppose the proposition is false, which means that there exists a root block $B_{\mathrm{r},m}$ in the attacking ledger $\mathbb{L}', F(\mathbb{L}', \mathbb{L}) > 0$

$$\mathbb{L}' = [\mathbf{B}'_{\mathrm{r}}, \mathbf{B}'_0, \mathbf{B}'_1, ..., \mathbf{B}'_i, ..., \mathbf{B}'_{n-1}]. \tag{33}$$

such that

$$j \leq o_i(B_{\mathrm{r},m}). \tag{34}$$

As the result, the root chain of the attacking ledger is of the form:

$$\mathbf{B}'_{\mathrm{r}} = [B_{\mathrm{r},0}, B_{\mathrm{r},1}, ..., B_{\mathrm{r},m}, ..., B'_{\mathrm{r},l'_{\mathrm{r}}}] \tag{35}$$

Following Eq. (20), this means that $\mathbf{B}'_i$ must be in the form of

$$\mathbf{B}'_i = [B_{i,0}, B_{i,1}, ..., B_j, ..., B_{i,o_i(B_{\mathrm{r},m})}, ..., B'_{i,l'_i}] \tag{36}$$

which means $B_{i,j}$ is not reverted and thus the proposition is true.

□

Proposition 2 has a very important implication on security: for any shard blocks that are confirmed by the root chain, the security of double-spending attack will be guarded by the root chain. If the root chain has strong security guarantee, e.g., the root chain is PoW-based and has very high hashpower, then all the blocks of the shard chains will share the same security guarantee. Note that for those shard blocks that are not confirmed by root chain, we could

- Incentivitize the root chain block producers to include new shard blocks as much as possible. A design is to use "tax" - a portion of coinbase rewards of shard blocks is re-allocated to the root chain block producers so that the root chain block producers would include new shard blocks as much as possible to maximize their returns.
- Improve the finality condition of a transaction. The finality of a transaction (and its block) is subjective - it depends on the recipient, and in a conventional blockchain network such

as Bitcoin or Ethereum, the recipient may require the confirmation of several blocks (e.g., 6 block confirmations for Bitcoin) before the recipient believes the transaction is finalized. In the case of Boson network, a recipient may require the confirmation of several root blocks rather than shard blocks before the transaction is assumed to be finalized.

C. Adding a Shard Chain Dynamically

Boson consensus allows adding a shard chain $\mathbf{B}_N$ with consensus $\mathbb{C}_N = (V_N(\cdot), F_N(\cdot,\cdot))$ at a specific root block height h_N dynamically. First of all, let us define

$$B_{i,-1} = \emptyset, \tag{37}$$

which denotes a block that does not exist. Therefore, we define

$$observed_N(B_{\mathrm{r},j}) = B_{N,-1} = \emptyset, \forall j < h_N \tag{38}$$

$$o_N(B_{\mathrm{r},j}) = -1, \forall j < h_N \tag{39}$$

which means that the root block should not include any shard blocks of shard chain N when root block height is smaller than h_N, and the genesis block of shard chain will be only created after root block height is greater than h_N. The validity function and fork-choice rule are the same as Definitions 1 and 2, where $\emptyset \in \mathbf{B}_n$ always holds true.

Note that if the current network does not support $\mathbb{C}_N = (V_N(\cdot), F_N(\cdot,\cdot))$, a network upgraded (hard fork) may be required. If $\mathbb{C}_N$ is already supported by the current network, an on-chain governance protocol (e.g., voting on root chain) can be employed to determine h_N and $\mathbb{C}_N$ without a network upgrade.

D. Running Different Types of Nodes in Boson Consensus

An advantage of Boson consensus is that, the node operator could run different types of nodes. Depending on

- the security level a node operator wants to achieve; and/or
- the full ledger or a specific shard or the root chain the node operator may be interested in; and/or
- the node resources that a node operator has;

, nodes can be classified into the following categories:

- **Full node**. A full node maintains the whole ledger of the network $\mathbb{L}$, and can fully verify the validity of the ledger and operate on the latest ledger of the network.
- **Shard node**. A shard node only maintains the ledger of one shard $\mathbf{B}_i$ and runs as a light client of the root chain by only validating the headers of the blocks of the root chain. By assuming $\mathbf{B}_{\mathrm{r}}$ valid, the shard node could fully validate the ledger of the shard $\mathbf{B}_i$ following the simplified per-shard validity function in Eq. (28) and operate on the latest shard ledger based on the fork-choice rule in Definition 2 in the absence of the ledger of the other shards.
- **Root node**. A root node only maintains the ledger of the root chain $\mathbf{B}_{\mathrm{r}}$ and runs as the light client of shard chains. It may maintain the metadata of shard chains such as headers, and by assuming $\mathbf{B}_i, \forall i \in \{0, ..., N-1\}$ valid, it could fully validate the ledgers of the root chain $\mathbf{B}_{\mathrm{r}}$ and operate on the latest ledger of the root chain.

A node operator is free to run any type of nodes: if the node operator requires significant security (e.g., miner on the root chain), then the operator could run a full node to make sure that the operator is working on the latest network ledger and every block produced is valid; or if the node operator is only interested in one shard and accepts weaker security (e.g., running a decentralized application just on one shard), then the operator could run a shard node with much lower resource cost than running a full node.

In addition, if the network capacity is increased by increasing the number of shards, the resource required by the full node increases, and it may become a significant burden if the full node is run by a powerful single machine. A way to alleviate the issue is to run multiple nodes that are trusted with each other and form a *cluster*, which runs at least one root node and shard nodes for all shards. Since the nodes within a cluster are trusted, the cluster could fully validate the whole ledger of the network $\mathbb{L}$ as a full node. As a result, by *scaling out* the full node via a cluster instead of running a powerful single machine, the burden of running a full node can be significantly lowered.

E. Cross-Shard Interoperability

In this subsection, we illustrate how multiple shards in a Boson network could interact with each other by performing cross shard transactions securely and efficiently. First of all, we classify the transactions into the following categories:

- In-shard transaction - a synchronous transaction that changes the state of a single shard.

- Cross-shard transaction - an asynchronous transaction that changes the state of multiple shards.

The cross-shard transaction will be initiated from a local shard, and the transaction may emit multiple asynchronous messages to remote shards and change the states of remote shards, where given the jth block $B_{i,j}$ of shard chain i, the messages generated by the transactions of the block are

$$M_k(B_{i,j}) = \mathbf{M}_{i,j,k} = [m_{i,j,k,0}, m_{i,j,k,1}, ..., m_{i,j,k,l_{\mathbf{M}_{i,j,k}}}], \tag{40}$$

where $m_{i,j,k,x}$ is the xth message sent from the jth block of shard chain i to shard chain k, and a list of messages can be directly applied to the remote shard state as

$$\mathbb{S}'_{y,z} = \sigma(\mathbb{S}_{y,z}, \mathbf{M}). \tag{41}$$

The examples of a message are

- Deposit some amount of a native token to a recipient in shard k; or
- Call a smart contract in shard chain k; or
- Create a smart contract in shard chain k.

To process the messages in a secure way, we need to ensure the *happen-before* relation between the emission of a message of a local shard and the processing of the message of a remote shard. To achieve that, the target shard contains an additional hash pointer that points to a root block

$$h(B_{\mathrm{r},p(B_{i,j})}) = prev\_root\_hash(B_{i,j}), \tag{42}$$

where $B_{\mathrm{r},p(B_{i,j})} = prev\_root(B_{i,j})$ returns the index of the root block that the shard block $B_{i,j}$ points to and it is a non-monotonic decreasing function over j, i.e., $p(B_{i,j-1}) \leq p(B_{i,j})$. By including a root block hash pointer with higher index, each shard could maintain a list of messages *ready-to-process* from other shards by

$$Q(B_{i,j}) = Q'(B_{i,j-1}) + \sum_{k=p(B_{i,j-1})+1}^{p(B_{i,j})} \sum_{n=0}^{N-1} \sum_{m=o_i(B_{\mathrm{r},k-1})+1}^{o_i(B_{\mathrm{r},k})} M_i(B_{n,m}) \tag{43}$$

where $Q(B_{i,j})$ is the pre-processing list of messages and $Q'(B_{i,j})$ is the post-processing list of messages as

$$Q'(B_{i,j}) = Q(B_{i,j}) \setminus P(B_{i,j}) \tag{44}$$

where $P(B_{i,j})$ is the list of messages that are processed in block $B_{i,j}$ and $\setminus$ is a set minus operator.

With the aforementioned definitions, we extend the state of a block $B_{i,j}$ as

$$\bar{\mathbb{S}}_{i,j} = (\mathbb{S}_{i,j}, Q'(B_{i,j})) \tag{45}$$

$$= \bar{T}(\bar{\mathbb{S}}_{i,j-1}, B_{i,j}) \tag{46}$$

$$= \bar{T}((\mathbb{S}_{i,j-1}, Q'(B_{i,j-1})), B_{i,j}) \tag{47}$$

where

$$\mathbb{S}_{i,j} = T(\sigma(\mathbb{S}_{i,j-1}, P(B_{i,j}))), B_{i,j}). \tag{48}$$

Summarizing all equations, we have

Definition 4 (Boson Consensus Validity Function with Cross-Shard Capability)**.** *The validity function of a ledger generated by Boson consensus with cross-shard capability is:*

$$\bar{V}(\mathbb{L}) = \begin{cases} 1, & \textit{if } \bar{V}_i(\mathbf{B}_i, \mathbf{P}_i) = 1, \forall i \in \{\mathrm{r}, 0, ..., N-1\} \textit{ and} \\ & o_i(B_{\mathrm{r},j-1}) \leq o_i(B_{\mathrm{r},j}),\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & observed_i(B_{\mathrm{r},j}) \in \mathbf{B}_i,\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & p(B_{i,j-1}) \leq p(B_{i,j}),\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & prev\_root(B_{i,j}) \in \mathbf{B}_{\mathrm{r}},\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & p(B_{i,o_i(B_{\mathrm{r}},j)}) < j,\ \forall 0 \leq i \leq N-1, j \geq 1 \textit{ and} \\ & Q'(B_{i,j}) = Q(B_{i,j}) \setminus P(B_{i,j}), \forall 0 \leq i \leq N-1, j \geq 1, \\ 0, & \textit{otherwise}, \end{cases} \tag{49}$$

where $\mathbf{P}_i = [P_{i,0}, P_{i,1}, ..., P_{i,l_i}]$ *with* $P_{i,j} = P(B_{i,j})$ *and* $\bar{V}_i(\cdot,\cdot)$ *is the extended version of validity function of shard* i *as*

$$\bar{V}_i(\mathbf{B}_i, \mathbf{P}_i) = \begin{cases} 1, & \textit{if } \bar{V}_i(\mathbf{B}_i \setminus [B_{i,l_i}], \mathbf{P}_i \setminus [P_{i,l_i}]) = 1,\ \textit{and} \\ & \quad B_{i,l_i}, P_{i,l_i} \textit{ can be applied to } \mathbb{S}_{i,l_i-1} \textit{ with } \mathbb{S}_{i,j} = T(\sigma(\mathbb{S}_{i,j-1}, P_{i,j}))), B_{i,j}), \\ 0, & \textit{otherwise}, \end{cases} \tag{50}$$

The fork-choice rule is still the same as Definition 2.

Given Definition 4, we could derive the simplified version of validity function as follows.

Proposition 3. *The simplified validity function of a ledger generated by Boson consensus with cross-shard capability is:*

$$\bar{V}_i([B_{i,0}, B_{i,1}, ..., B_{i,j}]) = \bar{W}_i(B_{i,j})$$

$$= \begin{cases} 1, & \textit{if } \bar{W}_i(B_{i,j-1}) = 1 \textit{ and} \\ & \bar{W}_{\mathrm{r}}(B_{\mathrm{r},p(B_{i,j})}) = 1 \textit{ and} \\ & p(B_{i,j-1}) \leq p(B_{i,j}) \textit{ and} \\ & prev\_root(B_{i,j}) \xrightarrow{p(B_{i,j})-p(B_{i,j-1})} prev\_root(B_{i,j-1}) \textit{ and } ,\forall i \in \{0, ..., N-1\}, \\ & B_{i,j} \xrightarrow{j-o_i(B_{\mathrm{r},p(B_{i,j})})} observed_i(B_{\mathrm{r},p(B_{i,j})}) \textit{ and} \\ & B_{i,j}, P_{i,j} \textit{ can be applied to } \bar{\mathbb{S}}_{i,j-1}, \\ 0, & \textit{otherwise}, \end{cases} \tag{51}$$

and

$$\bar{W}_{\mathrm{r}}(B_{\mathrm{r},j}) = \begin{cases} 1, & \text{if } \bar{W}_{\mathrm{r}}(B_{\mathrm{r},j-1}) = 1 \text{ and} \\ & B_{\mathrm{r},j} \text{ can be applied to } \mathbb{S}_{\mathrm{r},j-1} \text{ and} \\ & observed_i(B_{\mathrm{r},j}) \xrightarrow{o_i(B_{\mathrm{r},j})-o_i(B_{\mathrm{r},j-1})} observed_i(B_{\mathrm{r},j-1}), \forall i \in 0, ..., N-1 \text{ and} \\ & \bar{W}_i(B_{i,k}) = 1,\ \forall 0 \leq i \leq N-1, o_i(B_{\mathrm{r},j-1}) + 1 \leq k \leq o_i(B_{\mathrm{r},j}) \text{ and} \\ & B_{\mathrm{r},j} \xrightarrow{j-p_i(observed_i(B_{\mathrm{r},j}))} prev\_root(observed_i(B_{\mathrm{r},j})), \\ 0, & \text{otherwise.} \end{cases} \tag{52}$$

IV. QuarkChain - The First Realization of Boson Consensus

In this section, we present the first realization of Boson Consensus in QuarkChain. QuarkChain network is currently implemented in Python [8] while the implementation in other computer languages is also in progress at the time of writing. QuarkChain uses Ethereum's devp2p as the P2P protocol, implements the clustering feature, and thus allows running a full node with a root node and shard nodes trusted in the cluster. As a result, the performance of QuarkChain network can be easily scaled to thousands or even tens of thousands of TPS with a large number of shards in the network and sufficient number of machines in a cluster. As reported by the

Chain Name	Consensus	Block Interval	Mining Hash Alg.	Ledger/Transaction Model
Root Chain	PoSW	60s	Ethash	Account-based/EVM
Shard Chain 0	PoW	10s	Ethash	Account-Based/EVM
Shard Chain 1	PoSW	10s	Ethash	Account-Based/EVM
Shard Chain 2	PoSW	10s	Ethash	Account-Based/EVM
Shard Chain 3	PoSW	10s	Ethash	Account-Based/EVM
Shard Chain 4	PoSW	10s	Ethash	Account-Based/EVM
Shard Chain 5	PoSW	10s	Ethash	Account-Based/EVM
Shard Chain 6	PoSW	10s	Qkchash	Account-Based/EVM
Shard Chain 7	PoSW	10s	Qkchash	Account-Based/EVM

TABLE II

CONFIGURATIONS OF THE ROOT CHAIN AND SHARD CHAINS OF QUARKCHAIN MAINNET

community, the peak performance of QuarkChain testnet reaches about more than $55,000$ TPS with $1,024$ shards [9].

The mainnet of QuarkChain was officially launched on April 31, 2019 with 8 shards [10]. The root chain and most of shards run proof-of-staked work (PoSW) with enhanced security [11], and shard chains 6 and 7 uses Qkchash as the mining hash algorithm [12]. Table II summarizes the major configurations of all chains.

V. CONCLUSION

In this paper, we introduced a general mathematical model for blockchains and used the single blockchain model with Bitcoin and Ethereum as examples. Then we proposed Boson consensus algorithm - a blockchain sharding consensus algorithm that scales the network by running multiple shard chains for better performance and a root chain for better security. We proved that with root-chain-first fork-choice rule of each shard, all blocks of shard chains can be efficiently protected by the root chain, and thus the Boson consensus algorithm addresses the single-shard attack issues. In addition, we discussed different types of nodes running in the network with different requirements of node resources and security levels. Furthermore, we illustrated how to add a shard in a Boson network dynamically and how to perform cross-shard transactions securely and efficiently. Finally, we introduced the first realization of Boson consensus - QuarkChain and elaborated its configurations.

ACKNOWLEDGEMENT

We would like to thank Prof. Xiaoli Ma for reviewing the paper and offering numerous suggestions.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[2] C. Lee, "Litecoin." [Online]. Available: https://litecoin.org/

[3] "Bitcoin cash." [Online]. Available: https://www.bitcoincash.org/

[4] K. Croman, D. Christian, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[6] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[7] D. Larimer, "Delegated proof-of-stake white paper," 2014.

[8] "Python implementation of QuarkChain," https://github.com/QuarkChain/pyquarkchain, 2019, [Online; accessed 13-August-2019].

[9] "TPS Competition Final Leaderboard," https://medium.com/quarkchain-official/tps-competition-final-leaderboard-de6a8a8700a8, 2018, [Online; accessed 13-August-2019].

[10] "QuarkChain Mainnet," https://mainnet.quarkchain.io, 2019, [Online; accessed 13-August-2019].

[11] "Proof of Staked Work - A Simple PoW/PoS Hybrid Consensus," https://github.com/QuarkChain/pyquarkchain/blob/master/papers/posw.pdf, 2019, [Online; accessed 13-August-2019].

[12] "Irregular-Program-Based Hash Algorithms," https://github.com/QuarkChain/pyquarkchain/blob/master/qkchash/qkchash.pdf, 2019, [Online; accessed 13-August-2019; to be published in IEEE DAPPCON 2019].