A PoS-based sharding technique has also been designed for Ethereum [6]. At the initial stage, each validator has to pay some security deposit to obtain the right to create new blocks. A smart contract is used to assign validators and distribute transactions into shards. Each validator downloads all the block records in its own shard after the assignment is completed. In each shard, a block producer is independently selected among the validators in the shard to generate a block containing transactions that have not yet been processed. After the block generation phase is completed in each shard, a newly generated block data is transmitted to a smart contract. The smart contract merges the block data to the main blockchain ledger. The new block data is broadcasted to all nodes in the entire blockchain network to synchronize blockchain data. However, this sharding scheme has two weaknesses as follows:

- **Less mining power to attack:** When we divide validators in the entire network into several shards and process transactions independently in each shard, the mining power of each shard naturally becomes smaller than that of the entire network. Therefore, less mining power would be sufficient to attack a single shard in a targeted way effectively. For example, an attacker with 10% mining power of the entire network is not enough to attack the system. However, when validators are divided into six shards, 10% mining power of the entire network is the same as 60% mining power of a single shard. Therefore, the attacker can successfully perform attacks such as selfish mining.
- **Hardness to implement a secure and fair shard assignment scheme:** If validators and transactions are not partitioned into shards well in a secure and fair manner, an attacker can intentionally create nodes and/or transactions on a specific shard in a targeted way. In Ethereum smart contracts, however, it is hard to securely produce random numbers without external parameters because the function description in the smart contract can inherently be exposed to the attacker. Moreover, the use of a smart contract with external parameters can lead to a delay in the shard assignment process.

3 PROPOSED SYSTEM

An epoch is defined as the time period during which the updated shards and the selected block producer are used to create the next block of the main blockchain. For each epoch, we dynamically update shards and select their block producers. The operations performed during the time period of each epoch are shown in Figure 1 and can be divided into four stages:

(1) **Creating shards:** During the previous epoch, users registered as a *validator* with their information and a certain amount of deposit – this validator registration procedure is the same as that procedure in the Ethereum PoS mechanism [5]. When an epoch starts, validators, and *transactions* are divided into k shards according to the following rule. For a random shard assignment, the shard of each validator v is calculated as '$Hash(v\text{'s address} \,||\, Hash(b)) \,\%\, k$' where b represents the last block in the previous epoch. Without loss of generality, shards are indexed from zero to $k-1$. Transactions are also divided into k shards in a similar way – the shard of each transaction t is calculated as '$Hash(t\text{'s address} \,||\, Hash(b)) \,\%\, k$'.

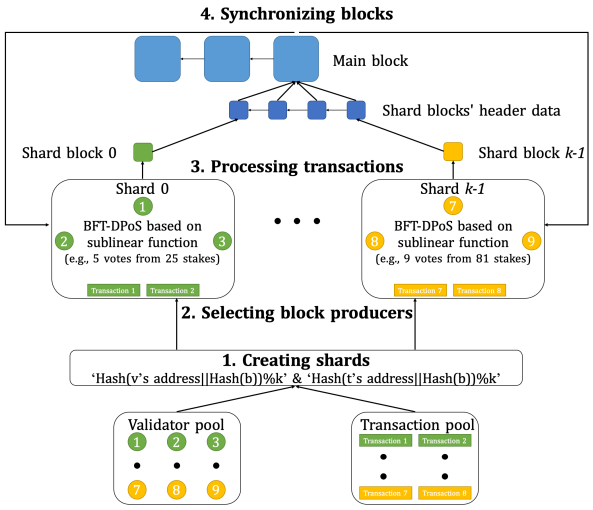

Figure 1: Overview of the proposed system.

(2) **Selecting block producers:** Block producers in each shard are selected in parallel using a Byzantine Fault Tolerance-Delegated Proof of Stake (BFT-DPoS) algorithm [4]. In each shard, m validators with the largest number of stakes are selected as the *block producers* in the shard during the epoch. Among them, a block producer p is chosen sequentially in a round-robin fashion. To discourage miners from holding a large number of stakes, we also suggest the use of a sublinear function so that miners can cast a number of votes which is proportional to a square root of their stakes, rather than linearly proportional to the stakes.

(3) **Processing transactions:** The producer p collects transactions in the shard, and generates a *shard block* to store those transactions. The shard block generated by p is distributed to the remaining $m-1$ block producers through its shard network. The shard block is accepted only if the sum of stakes of the other block producers (except p) who agree the block is greater than 2/3 of the total stakes of all block producers (except p) following the BFT theory. During this process, each validator can try to check the validity of the distributed shard block. In case p creates an invalid shard block, p loses its own deposit.

(4) **Synchronizing blocks:** When the shard block generation process is completed in all shards, the producer p creates the header data about the shard block (e.g., its index and hash value) and then broadcasts it to all network nodes through the blockchain protocol. Each node creates the next *main block* by merging all the received shard header data into it in the predefined order of the corresponding shard blocks.

4 EVALUATION

In this section, we analyze the security and the performance of the proposed system compared with Ethereum.

4.1 Security analysis

Sharding-based blockchain protocols can easily be attacked even by an attacker with a small number of stakes. When the number of shards is k, $(51/k)$ % of the total stakes would be sufficient to

perform 51% attack in the case of a single shard. Also, since the proposed protocol follows the BFT algorithm, $(1/3k)$ of the total stakes is sufficient for DoS attack, disabling a single shard. To mitigate such attacks, we suggest the use of a sublinear function that enforce users to divide their stakes into multiple accounts holding a small number of stakes to maximize their profits.

If we specifically use the square root function as a sublinear function, the security of the proposed system can be analyzed with k. Table 1 shows the minimum proportions of stakes to perform DoS attacks with k, assuming that all shards have an equal number of stakes. From Table 1, we found that the proposed system is secure against an attacker with 30% stakes when $k \leq 6$ while the Ethereum sharing system can be vulnerable even against an attacker with 6% stakes under the same conditions.

Table 1: Security analysis of proposed system with k.

k	Minimum mining power to perform DoS attack		Probability of succeeding DoS attack with 25% stakes
	Ethereum sharding [6]	Proposed sharding	Proposed sharding
2	16.67%	Not possible	4.75%
3	11.11%	Not possible	11.90%
4	8.33%	69.44%	16.85%
5	6.67%	44.44%	20.33%
6	5.56%	30.86%	22.66%
7	4.76%	22.68%	24.83%

To avoid the penalty derived by a sublinear function in stakes, the attacker can generate multiple accounts to perform Sybil attacks. Here, we are specifically interested in the security analysis of the proposed blockchain protocol against such an attacker with 25% stakes, which are the same as the minimum mining power needed to perform selfish mining attacks.

When the attacker divides his or her stakes into a massive number of Sybil accounts, the number of the attacker's stakes in a single shard follows the normal distribution with the mean of $25/k\%$ and the standard deviation of $5\sqrt{k-1}/k\%$. The probability that the attacker's stake in a shard exceeds 1/3 of the total stake in the shard can be calculated by z satisfying $\mu + z\sigma \geq 1/3k$. We can see that the success probability of DoS attacks is less than 25% when $k \leq 7$. Therefore, we recommend $k \leq 6$ to security requirements.

We specifically generate random values using a double hashing strategy to dynamically assign *validators* into shards every time. If we use fixed parameters only for a hash function, *validators* and transactions are always included in the same shard although their shard index can be changed over time. Bonneau and Clark [1] already proposed the idea using the upcoming block data as a public random source. We extend this idea to dynamically create shards for *validators* and transactions with the upcoming block data. We note that the proposed random shard assignment scheme can be implemented at the protocol level, unlike the existing system [6] using smart contracts.

4.2 Performance analysis

This section analyzes the performance of the proposed protocol using a fair and dynamic sharding management scheme compared with the PoS-based sharding scheme proposed by Ethereum [6]. For the performance analysis, we set $k = 6$ to provide sufficient security against several attacks discussed in Section 4.1.

The Ethereum's consensus protocol is designed to generate a block every 12 seconds because 12 seconds is known as the time to propagate the block in order to avoid the creation of forks and stale blocks [2]. Recently, Lee et al. [7] collected real-world Ethereum block propagation time records and found that a block can be distributed to about 37% and 95% of all nodes in the entire network, respectively, within 0.25 and 4 seconds on average.

Based on this observation, we expect that the synchronization of shard header data for the main blockchain would be completed within 4 seconds. Also, each shard block in a shard can be created and synchronized within 0.25 seconds because the size of each shard network is significantly smaller than 37% of the entire network when $k = 6$. If we assume that the creation of shards and the selection of block producer can be completed within 0.25 seconds, we can numerically generate 31 shard blocks for each shard.

In summary, the proposed system can generate 186 shard blocks in total for each epoch with the real parameter settings obtained by the Ethereum network.

5 CONCLUSION

In this paper, we propose a PoS-based blockchain protocol using fair and dynamic sharding management. To overcome the limitations of existing sharding-based protocols, we specifically design a sharding management scheme to dynamically divide miners into several groups over time by using the randomness of the blockchain itself at the protocol level. The proposed protocol also adopts the concept of sublinear function in stakes to reduce the attack possibility on a single shard. Finally, we show that the proposed protocol is 186 times faster than Ethereum in the same environment.

For future work, we plan to extend our idea to a more practical system for establishing shards using validators' geographical locations in the blockchain network.

ACKNOWLEDGMENTS

This research was supported in part by the ITRC program (IITP-2019-2015-0-00403), MSIP&IITP (2015-0-00914), and the ICT R&D program of MSICT/IITP (2017-0-00045, Hyper-connected Intelligent Infrastructure Technology Development).

REFERENCES

[1] Bonneau, Joseph and Clark, Jeremy and Goldfeder, Steven. 2015. On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*.
[2] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*.
[3] James C. Corbett et al. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems* 31, 3, Article 8 (2013), 8:1–8:22 pages.
[4] Daniel Larimer. 2017. DPOS Consensus Algorithm - The Missing White Paper. https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper Accessed on: 2019-08-26.
[5] Danny Ryan and Chih-Cheng Liang. 2018. EIP 1011: Hybrid Casper FFG. (2018). https://eips.ethereum.org/EIPS/eip-1011 Accessed on: 2019-08-26.
[6] Ethereum Foundation. 2019. Sharding-FAQ. https://github.com/ethereum/wiki/wiki/Sharding-FAQ Accessed on: 2019-08-26.
[7] Daehwa Rayer Lee, Yunhee Jang, Hanbin Jang, and Hyoungshick Kim. 2019. 80% of Block Propagation Rate is Enough - Towards Secure and Efficient PoW-based Blockchain Consensus. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*.
[8] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*.