

A Community Detection-based Blockchain Sharding Scheme*

Zixu Zhang¹, Xu Wang¹, Guangsheng Yu², Wei Ni², Ren Ping Liu¹, Nektarios Georgalas³, and Andrew Reeves³

¹ GBDT, University of Technology Sydney, Sydney, Australia

`zixu.zhang@student.uts.edu.au`

`xu.wang-1@uts.edu.au`

`renping.liu@uts.edu.au`

² Data61, CSIRO, Sydney, Australia

`saber.yu@data61.csiro.au`

`wei.ni@data61.csiro.au`

³ Applied Research, British Telecom, Martlesham, UK

`nektarios.georgalas@bt.com`

`andrew.reeves@bt.com`

Abstract. Sharding has been considered a promising approach to improving blockchain scalability. However, multiple shards result in a large number of cross-shard transactions, which require a long confirmation time across shards and thus restrain the scalability of sharded blockchains. In this paper, we convert the blockchain sharding challenge into a graph partitioning problem on undirected and weighted transaction graphs that capture transaction frequency between blockchain addresses. We propose a new sharding scheme using the community detection algorithm, where blockchain nodes in the same community frequently trade with each other. The detected communities are used as shards for node allocation. The proposed community detection-based sharding scheme is validated using public Ethereum transactions over one million blocks. The proposed community detection-based sharding scheme is able to reduce the ratio of cross-shard transactions from 80% to 20%, as compared to baseline random sharding schemes, and retain the ratio of around 20% over the examined one million blocks.

Keywords: blockchain, sharding, community detection

1 Introduction

Scalability is one of the well-known bottlenecks in the development of blockchain technology. The scalability of a blockchain refers to its ability to process more transactions per unit of time. Many blockchain platforms, such as Bitcoin [1] and Ethereum (ETH) [2] (about seven transactions per second for Bitcoin and

* Supported by BT Group Plc

about fifteen for Ethereum), are inefficient in processing transactions per unit of time. Blockchain platforms process transactions much slower than VISA, which processes 56,000 transactions per second [3]. The scalability of blockchain has been studied extensively to increase transaction rates [4]. Sharding technology is the most potential technology to improve blockchain scalability among many scaling schemes. For blockchains to become scalable, various sharding solutions have been introduced in [5–9].

Sharding is implemented on-chain, making it part of the layer-1 scaling solution [4]. Scaling on layer 1 has two directions, vertical scaling and horizontal scaling [10, 11]. Vertical scaling of a chain involves increasing the transaction capacity within each block or shortening the block period. Vertical scaling increases transaction throughput, but it requires more bandwidth from nodes. As a result, vertical scaling methods are not feasible for nodes with limited bandwidth. Sharding is a horizontal scaling scheme that divides the blockchain into multiple parts. Each independent chain only responds to its transaction mining. As the number of shards increases, transaction throughput increases linearly [12].

Blockchain sharding increases transaction throughput, but cross-shard transactions also increase during sharding. Due to excessive cross-shard transactions, the sharding technique cannot scale efficiently in the blockchain [13]. Transactions between senders and recipients on different shards are termed cross-shard transactions, while intra-shard transactions are transactions within the same shard. Cross-shard transactions require multi-phase protocols to verify their authenticity. A multi-phase protocol typically consists of a prepare and a commit phase [14]. Since cross-shard transactions take longer to process and require more processing power than intra-shard transactions, sharding in the blockchain is not as effective in improving scalability when cross-shard transactions are involved. In [6], more than 95% of transactions are cross-shard transactions. Thus, reducing cross-shard transactions with efficient shard allocation strategies is a direct way to optimize sharding.

This paper intends to reduce the transactions between shards by allocating frequent blockchain node pairs to the same shard. We transform the problem of obtaining the least number of cross-shard transactions into a graph partition problem. A weighted and undirected graph is constructed using the number of transactions between nodes. Then, we propose generating the community result for assigning nodes to shards using a community detection algorithm. We demonstrate our scheme by testing its reliability and determining that cross-shard transactions are reduced to 20%. The following is a summary of our contributions.

- We convert the blockchain sharding challenge into a graph partitioning problem. The sharding optimization can be transformed into the community detection problem, where nodes are tightly linked with nodes in the same community and loosely connected with nodes in other communities.
- We propose a new sharding scheme using community-detection algorithms. The scheme is achieved by designing an undirected and weighted graph to capture transactions where the weight of an edge represents the number of

transactions between the endpoints. The graph is then divided into communities by running the community-detection algorithm, and each community forms a shard.

- Validated by comprehensive experiments on Ethereum transaction data, the proposed sharding scheme can significantly reduce the number of cross-shard transactions, e.g., from 80% to 20%, compared with popular random sharding schemes. Experiment results also confirm the stability of the proposed sharding scheme.

This paper is organized as follows. Section II introduces related blockchain sharding works. Section III depicts the proposed community detection-based sharding scheme. The experiment results are presented in Section IV, followed by conclusions in Section V.

2 Related Work

Assigning nodes to shards is commonly accomplished by adding Distributed Randomness Generation (DRG) protocols to enhance the randomness of the assignment process. Elastico [5], Omniledger [6], and RapidChain [7] all use DRG protocols to randomly allocate nodes to shards. In [5], nodes are assigned to a shard according to the last few bits of the solution of the Proof of Work (PoW) puzzle. The PoW puzzle is derived from the combination of the epoch randomness, which is generated with a DRG protocol [15], node identity, and nonce. Kokoris *et al.* [6] develop Omniledger, a distributed ledger where validators are randomly assigned to shards and synchronize the previous shard state. Each shard has a leader who is elected by a verified random function. The leader generates a randomness output for deriving a random permutation with other nodes by initiating the *RandHound* DRG [16] protocol. In subsequent allocations, no more than $\frac{1}{3}$ nodes at the beginning of the permutation are shuffled randomly to other shards. As with [5,6], Zamani *et al.* [7] propose a sharding-based blockchain protocol RapidChain. RapidChain uses different random allocation methods for nodes. In [7], the *reference committee* (a special shard) uses the Feldman Verifiable Secret Sharing (VSS) DRG [17] protocol to generate unbiased random outputs. The PoW puzzles are generated by random outputs and need to be solved by the nodes to join the system. Each node is then randomly assigned to each shard by a member of the *reference committee* executing *Commensal Cuckoo* rule [18]. With the *Commensal Cuckoo*, each node is mapped in a random position of Interval $[0, 1)$ based on its identity by a hash function. Interval $[0, 1)$ are then divided into regions representing shards. As new nodes are added, existing nodes in the same region are moved to new random regions.

Without a DRG protocol, node allocation to shards can also be achieved according to addresses' prefixes, such as Monoxide [8]. In addition to allocating nodes to shards based on addresses, nodes can be divided by voting from other nodes. For instance, Chainspace [9] permits nodes to move between shards

based on the votes of other nodes. Through a smart contract called Manage-Shards, voting is carried out. By deploying different shards on different channels, Fabric [19, 20] address sharding can be achieved. A trusted entity is used for cross-shard transactions.

Nodes can be assigned to shards using effective partitioning methods for reducing cross-shard transactions. Classical graph partitioning algorithms like Kernighan-Lin [21] can be used to optimize the sharding process. The Kernighan-Lin algorithm divides a graph into two communities of known size and exchanges any two vertices between them to obtain two communities with the smallest cut set size. The Kernighan-Lin algorithm causes multiple partitions since it divides two communities at once. Nevertheless, a multi-partitioned system will increase algorithm complexity.

Existing blockchain sharding schemes, including random sharding, voting-based sharding, and channel-based sharding, have not tried to improve the scalability of sharded blockchains by reducing the number of cross-shard transactions. Graph partitioning and community detection algorithms are promising to reduce cross-shard transactions but have yet to be developed for blockchain sharding schemes, especially for blockchains craving high scalability.

3 Proposed Community Detection-based Sharding Scheme

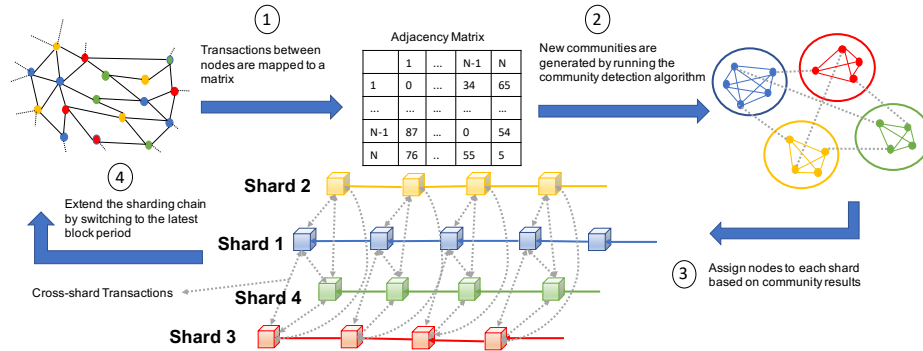


Fig. 1. The flow diagram of the proposed Re-sharding system comprises the following four stages. ① Graph generation: generate an adjacency matrix according to the number of transactions between nodes. ② Community detection: run the community detection algorithm to identify node communities so that nodes frequently exchange with nodes in the same community and transact less with nodes in different communities. ③ Community-based sharding: allocate nodes to shards according to detected communities, where nodes in the same community are in the same shard, and implement the sharding result. ④ Chain extension: extend the chains in parallel.

We propose a novel blockchain sharding scheme using the community detection algorithm. An adjacency matrix representing transactions (TXs) between node pairs is used as the input parameter for the system model we presented. Our model uses the Louvain algorithm in community detection [22] to obtain communities that respond to the sharding result. All nodes are assumed to be trustworthy in the model. Under our assumption, cross-shard transactions are reduced to a lower frequency based on our model. Therefore, our community detection-based sharding model is more suitable for permissioned chains since security issue is out of scope. Notations used in this paper are collected in Tab. 1.

The blockchain sharding network is reviewed as an undirected weighted graph $G = (V, E)$. The set of vertices V represents node addresses, and the set of edge E represents the transaction number between node pairs. This G is incorporated into the community detection algorithm using the adjacency matrix format.

Table 1. notation and definition in the blockchain sharding system.

Notation	Description
\mathcal{N}	Set of nodes
\mathcal{C}	Set of community
\mathcal{S}	Set of shard
\mathcal{B}_p	An epoch of blocks
\mathcal{E}_t	t -th block epoch
r	The cut weight ratio
\mathcal{A}	The adjacency matrix of the transaction graph
s	The number of shards
ϕ_i	The number of Intra-shard transactions
ϕ_c	The number of Cross-shard transactions
ρ	Threshold for the cross-shard transaction ratio

Fig. 1 illustrates the sharding system we use to reduce R_c . This system performs four stages: Graph generation, Community detection, Community-based sharding, and Chain extension. The following Eq. 1 calculates the ratio of cross-shard transactions.

$$R_c = \frac{\phi_c}{\phi_i + \phi_c}. \quad (1)$$

STAGE 1: Graph generation (Alg. 1 lines 1-7). The model generates an adjacency matrix \mathcal{A} based on the number of transactions between nodes. As shown in Alg. 1, the adjacency matrix is initialized with zero. The algorithmic flow starts with obtaining the dataset of the node \mathcal{N} that sends and receives transactions from the selected block epoch. Then, node dataset \mathcal{N} are sorted based on the number of transactions to a sorted node dataset \mathcal{N}_s . An adjacency matrix \mathcal{A} is generated after traversing all node sets. The size of the \mathcal{A} depends on the node number. The i -th row and the j -th column in matrix $a_{i,j}$ represents

Algorithm 1: Community-based sharding

```

▷ Sharding( $\mathcal{B}_p, s$ )
Input:
   $\mathcal{B}_p$ : An epoch of blocks;
   $s$ : The number of shards.
Output:
   $\mathcal{S}$ : Shards with nodes;
   $r$ : The cut weight ratio.
1  $\mathcal{A} = \mathbf{0}$ 
2  $\mathcal{N} \leftarrow \text{GetAllNodes}(\mathcal{B}_p)$ 
   // Sort nodes according to the number of transactions.
3  $\mathcal{N}_s \leftarrow \text{Sort}(\mathcal{N}, \text{TX})$ 
4 for  $i \leq |\mathcal{N}_s|$  do
5   for  $j \leq |\mathcal{N}_s|$  do
6      $a_{i,j} \leftarrow \text{CountTX}(\mathcal{B}_p, N_i, N_j)$ 
7    $\mathcal{C} \leftarrow \text{CommunityDetection}(\mathcal{A}, s)$ 
   // Calculate the cut weight ratio, which is the sum of weights of the edges crossing
   // the communities divided by the sum of weights of all edges.
8  $r \leftarrow \text{CalculateCutRatio}(\mathcal{A}, \mathcal{C})$ 
9  $\mathcal{S} \leftarrow \text{MapNodesToCommunities}(\mathcal{N}_s, \mathcal{C})$ 
10 return  $\mathcal{S}, r$ 

▷ ShardsExtension()
Input:
   $\rho$ : The threshold of cross-shard TX ratio;
   $s$ : The number of shards.
12 while True do
13   ShardsBlockMining()
14   if Shard heights reach an epoch then
15      $\mathcal{B}_p \leftarrow \text{Blocks in the latest period}$ 
16      $\phi_i \leftarrow \text{CountIntraShardTX}(\mathcal{B}_p)$ 
17      $\phi_c \leftarrow \text{CountCrossShardTX}(\mathcal{B}_p)$ 
18     if  $\frac{\phi_c}{\phi_i + \phi_c} > \rho$  then
19        $\mathcal{S}, r = \text{Sharding}(\mathcal{B}_p, s)$ 
        // Reallocated nodes to new shards.
20       ReAllocateNodes( $\mathcal{S}$ )
21        $\rho \leftarrow \max(\rho, r)$ 

```

the number of transactions between nodes N_i and N_j . Matrix diagonal elements represent the number of transactions between nodes and themselves.

STAGE 2: Community detection (Alg. 1 lines 8-9). The community detection algorithm generates a community set \mathcal{C} using an adjacency matrix \mathcal{A} and a fixed number of shards s as input parameters. An algorithm for detecting communities facilitates more frequent exchanges between nodes within a community and less exchange between nodes from different communities. To measure the quality of the sharding result, we define a cut weight ratio r . Cut weight ratio is calculated by dividing the sum of edge weights crossing communities by the sum of all edge weights.

STAGE 3: Community-based sharding (Alg. 1 lines 10-11). Based on the optimized outputs of the community set \mathcal{C} from previous stages, sorted nodes are assigned to the new communities and the filled shard set \mathcal{S} is returned. Filled shards set \mathcal{S} with sorted nodes and a cut weight ratio r are returned at the end of each epoch.

STAGE 4: Chain extension (Alg. 1 lines 12-21). As the new shards return, each chain mines blocks and extend in parallel until it reaches the end of an epoch. According to returned sharding results, the number of intra-chain transactions ϕ_i and the number of cross-chain transactions ϕ_c are separately counted. According to Eq. 1, the statistical results of both transactions are used to calculate a ratio of cross-shard transactions R_c . We set a threshold ρ for cross-shard transactions in our system. The threshold determines whether or not a node needs to be reassigned during a block epoch \mathcal{E}_t . A node is not re-allocated if R_c is less than the threshold. Otherwise, nodes are re-allocated to other shards based on the community detection algorithm. Additionally, the value of ρ is based on the largest R_c .

4 Experiments

4.1 Experiment Settings

An experimental framework is implemented in a local environment (MacBook Pro with 2.5 GHz Quad-Core Intel Core i7 and 16GB memory) to evaluate a proposed blockchain sharding scheme. We create a virtual machine using Conda on Visual Studio Code and implement our framework in Python 3.10.44. Ethereum block data ranging from 13.7 million to 15.04 million are downloaded and sorted from the Ethereum public endpoint [23]. There are 1.34 million blocks in the captured block range. Experiments are conducted on consecutive 100,000 blocks randomly selected from captured block ranges. Our randomly selected block period is between 14 million and 14.9 million. We divide the selected block period into ten equal parts, each containing 10,000 blocks.

4.2 Data Overview

We have 762,203 node addresses and 3,735,641 transactions during our first test block epoch. The community detection algorithm uses the top 90 addresses¹ sending the most transaction numbers out of 762,203 addresses. We define K as the number of addresses with the most transactions in subsequent tests. After sorting the first 90 nodes, we see that the address with the most transactions belongs to OpenSea². In our first tested 10,000 blocks, OpenSea generated 154,879 transactions. Besides OpenSea, big companies like Coinbase³ and Uniswap⁴ also have large numbers of transactions on Ethereum. According to the adjacency matrix, 31 nodes do not have any transactions with any of the first 90 nodes. The row sum of all the 31 sender addresses in the matrix is zero. Thus we remove these 31 addresses, and the top 90 most transacted addresses form a 59×59 adjacency matrix.

4.3 Sharding Analysis

The comparison between a community detection-based sharding method and a random sharding method is shown in Fig. 2. In Fig. 2(a), the vertical axis represents the number of cross-shard transactions, while the horizontal axis represents the Ethereum block sequence number. Fig. 2(b) differs from Fig. 2(a) in that the vertical axis represents the ratio of cross-shard transactions, while the horizontal axis remains the same. During the first block epoch, we divide the 59 addresses into four shards, which generate 76,369 transactions. As a result of adopting the community detection algorithm, 65,703 transactions are intra-shard transactions, and 10,666 are cross-shard transactions.

To compare with the community detection-based sharding scheme, we simulate a process of randomly assigning addresses, similar to the process of nodes randomly assigned to varying shards in Omniledger [6]. Nodes can be randomly allocated into shards of the same size. Alternatively, nodes can be randomly divided into shards of different sizes. Therefore, we develop two methods for randomly allocating nodes to shards in the experiments. Keeping each shard with the same number of nodes is balanced sharding. Sharding with more or fewer nodes in each shard is unbalanced.

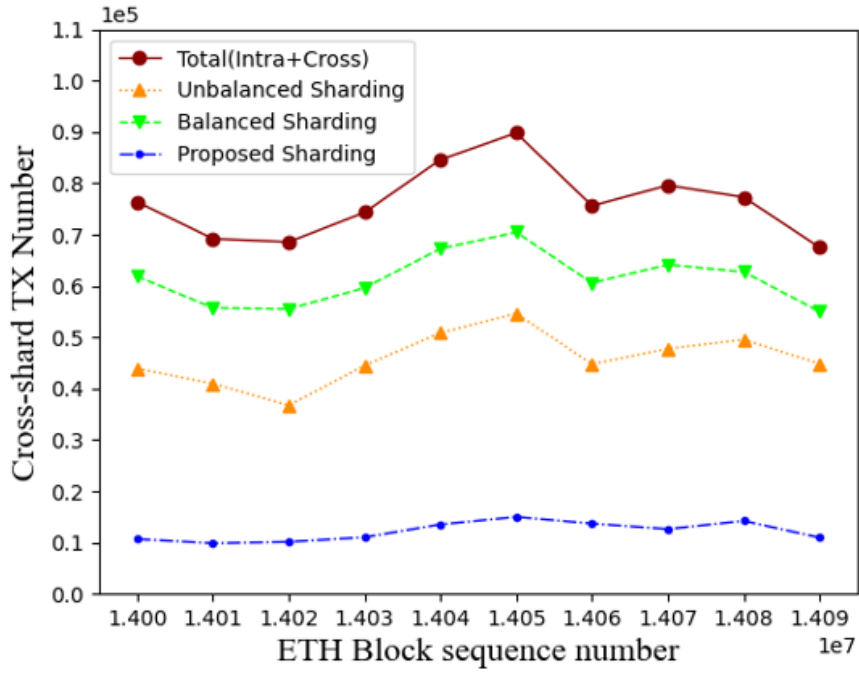
As shown in Fig. 2(a), the proposed sharding scheme keeps the number of cross-shard transactions around 10,000. Additionally, there is a transaction peak of around 14.05 million blocks. In the peak block, the random balanced sharding method generates 70,455 cross-shard transactions, while the unbalanced sharding method generates 54,629 cross-shard transactions. Community detection-based sharding generates only 14,972 cross-shard transactions, approximately one-fifth

¹ Due to the limited computational capacity, the local computer can only handle up to 90 addresses.

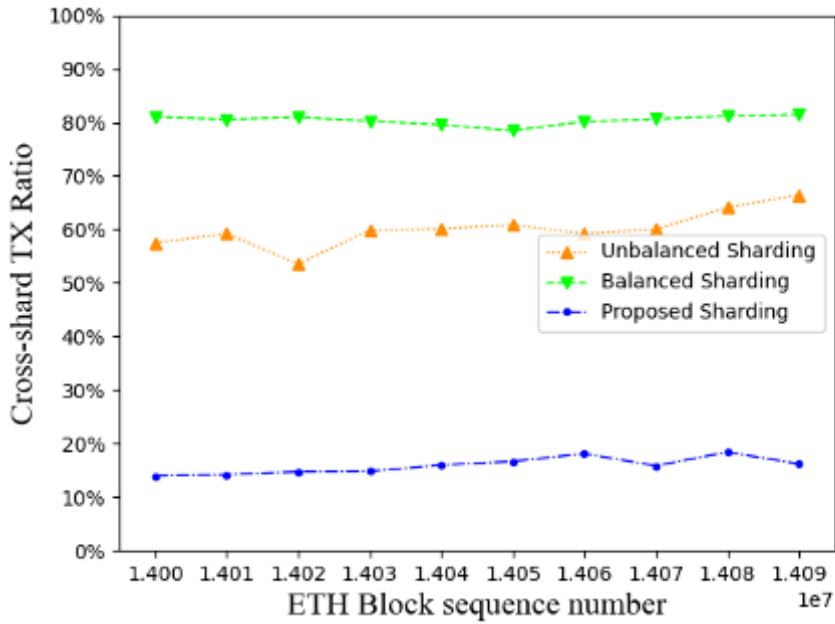
² 0x7be8076f4ea4a4ad08075c2508e481d6c946d12b, <https://opensea.io/>

³ 0x503828976d22510aad0201ac7ec88293211d23da, <https://www.coinbase.com/>

⁴ 0xd3d2e2692501a5c9ca623199d38826e513033a17, <https://uniswap.org/>



(a) Cross-shard TX Number, Community Detection VS Random Allocation



(b) Cross-shard TX Ratio, Community Detection VS Random Allocation

Fig. 2. Testing Ethereum address in sharding framework under unbalanced random allocation method, balanced random allocation method, and community detection-based allocation method while fixing tested addresses number $K = 90$ and shard number $s = 4$.

of the number generated by random sharding. With our proposed sharding technique, the peaks of shard lines appear relatively flat because it is based on a community detection algorithm. Fig. 2(b) shows that random balanced and unbalanced sharding leads to a high ratio of cross-shard transactions, i.e., around 80% and 60% respectively. Compared with random sharding with balanced shards, the ratio of cross-shard transactions is reduced from 80% to 20% by implementing the community-detected sharding rather than random sharding.

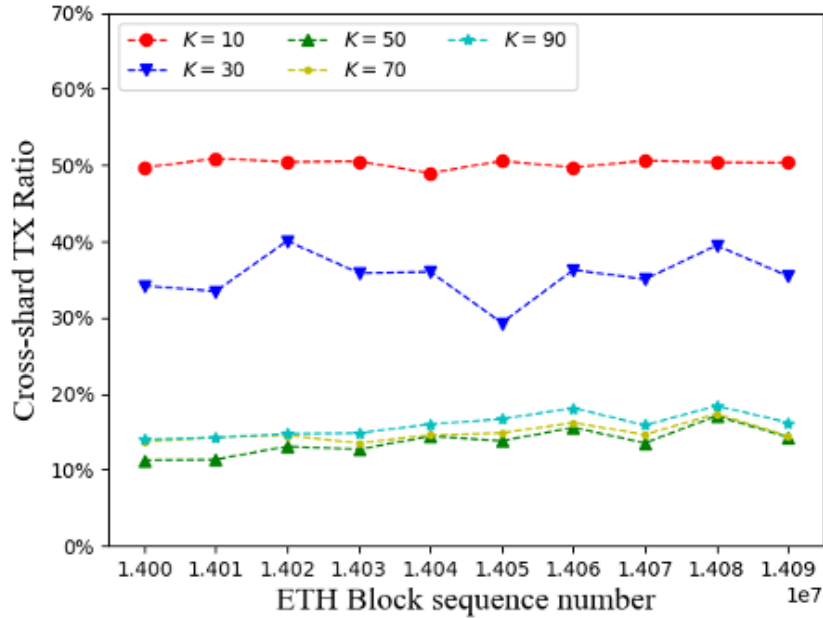


Fig. 3. Cross-shard TX ratio, the top 10 to 90 most traded addresses. Varying K within $\{10, 30, 50, 70, 90\}$, and fixing $s = 4$.

We explore our proposed scheme by changing the numbers of addresses or numbers of shards. According to Fig. 3, we demonstrate changes in the ratio of cross-shard transactions when the number of addresses varies. Tab. 2 presents the number of deleted addresses and the number of remaining addresses for each K value.

We evaluate our proposed community detection-based sharding approach by varying ETH most traded top K addresses from the range $\{10, 30, 50, 70, 90\}$ and fixing shard number $s = 4$. After traversing ten epochs, experiments on the ratio of ϕ_c in Fig. 3 reveal an interesting pattern of change. We see that a decrease in the ϕ_c occurs with an increase in K from 10 to 50, and an increase in the ϕ_c occurs with an increase in K from 50 to 90. The ratio of cross-shards is lowest when $K = 50$.

Table 2. Testes ETH top K Addresses and Remaining Addresses

Tested ETH address number	$K=10$	$K=30$	$K=50$	$K=70$	$K=90$
Deleted address number	3	9	14	21	31
Remaining address number	7	21	36	49	59

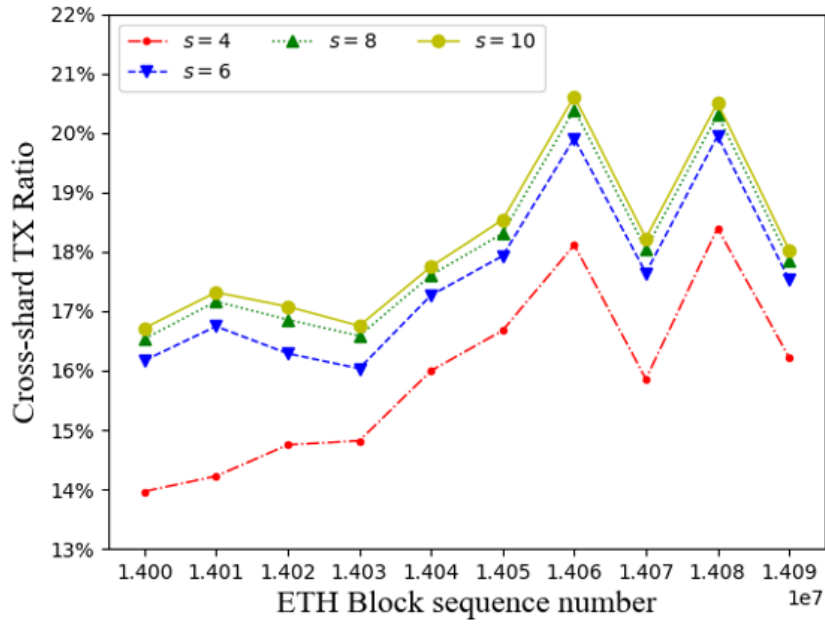


Fig. 4. Cross-shard TX ratio, while varying shards number within $\{4, 6, 8, 10\}$, and fixing $K = 90$.

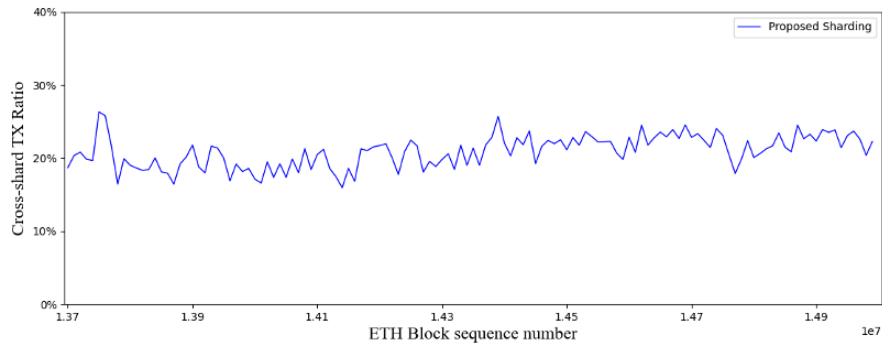


Fig. 5. Variation of cross-shard TX ratios in 1.3 million blocks.

In addition to varying the top K , we test sharding performance by changing the shard number s from the range $\{4, 6, 8, 10\}$ and fixed $K = 90$. The observation in Fig. 4 indicates that an increase in s leads to an increase in ϕ_c . Although the s increases with ϕ_c , the s and the ϕ_c do not follow a linear relationship. A significant change in the ratio of ϕ_c occurs when the s is increased from 4 to 6, followed by a continually smaller change as the s increases. Sharding results show that fewer shards result in a lower ratio of cross-shard transactions. Also, cross-shard transactions will reach upper bounds as more shards are added.

Fig. 5 shows the test results of our proposed sharding method from the block range of 13.7 million to 15.04 million. We display the overall ratio of cross-shard transactions during the tested period. Community detection-based sharding reduces transactions between shards to 20% and stays stable.

5 Conclusion

We presented a community detection-based sharding scheme to reduce cross-shard transactions. We converted blockchain data into an undirected and weighted graph format and executed a community-detection algorithm to get communities. The assignment of nodes to communities represented the division of nodes into shards. The superiority of this sharding approach was demonstrated by evaluating it on a real-world Ethereum dataset. The community detection-based sharding scheme reduced the ratio of cross-shard transactions to 20% and maintained stability.

Acknowledgement

This work was supported by BT Group plc through the Project “*Blockchain based Workflow and Policy Management Platform*”.

References

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.
2. Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
3. David Gerard. *Attack of the 50 foot blockchain: Bitcoin, blockchain, Ethereum & smart contracts*. David Gerard, 2017.
4. Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *Ieee Access*, 8:16440–16455, 2020.
5. Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30, 2016.

6. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.
7. Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018.
8. Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, pages 95–112, 2019.
9. Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hryczyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.
10. Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.
11. Guangsheng Yu. Scaling-out blockchains with sharding: an extensive survey.
12. Mearian, L. Sharding: What it is and why many blockchain protocols rely on it. online, accessed, 2019. <https://www.computerworld.com/article/3336187/sharding-what-it-is-and-why-so-many-blockchain-protocols-rely-on-it.html>.
13. Mengqian Zhang, Jichen Li, Zhaohua Chen, Hongyin Chen, and Xiaotie Deng. An efficient and robust committee structure for sharding blockchain. *arXiv preprint arXiv:2112.15322*, 2021.
14. Yizhong Liu, Jianwei Liu, Jiayuan Yin, Geng Li, Hui Yu, and Qianhong Wu. Cross-shard transaction processing in sharding blockchains. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 324–339. Springer, 2020.
15. Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. In *International Conference On Principles Of Distributed Systems*, pages 275–289. Springer, 2006.
16. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.
17. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.
18. Siddhartha Sen and Michael J Freedman. Commensal cuckoo: Secure group partitioning for large-scale services. *ACM SIGOPS Operating Systems Review*, 46(1):33–39, 2012.
19. Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.
20. Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*, pages 111–131. Springer, 2018.
21. Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

22. Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.
23. Ankr.com. Build on Ethereum With Instant RPC Endpoint. online, accessed, 2022. Available at: <https://www.ankr.com/protocol/public/eth/>.