International Journal of Social Sciences and Management Review

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

ENHANCING TRANSACTION THROUGHPUT IN PUBLIC BLOCKCHAIN NETWORK USING NESTED CHAINS

VILMA MATTILA, PRATEEK DWIVEDI, PRATIK GAURI & MD AHBAB
5ire (Sustainable Distributed Computing)
160 City Rd, London, United Kingdom
Corresponding Author: ahbab@5ire.org

https://doi.org/10.37602/IJSSMR.2022.5218

ABSTRACT

In this paper, we propose a new transaction sharding technique that aims to improve the scalability of the 5irechain network. This sharding technique called "nested chain" can be easily implemented in a decentralized setting like the blockchain. The sharding technique reduces cross-chain dependencies by ensuring that all the transactions sent from the same address will be found in blocks along the same chain and there will be no possibility of double-spending of the same token in two different chains. Our technique does not require additional tokens and it operates only on the same token across multiple shards.

1.0 INTRODUCTION

The total market cap of cryptocurrencies stood at US$3 trillion in November 2021 before shedding a third of its total value by early 2022. There are well over 6,000 different coins and tokens trading on cryptocurrency exchanges today. The underlying structure of these cryptocurrencies is based on blockchains – a data structure that serves as a decentralized ledger that records past transactions in an immutable and transparent manner.

Scalability is one of the crucial issues concerning a blockchain ecosystem. The scalability of any blockchain platform determines the peak number of transactions it can handle. Decentralized ledgers must address the scalability issue in order to be adopted by large financial farms. There are a few techniques currently being explored by researchers to deal with the scalability issue. One of them relies on having parallel chains with dedicated coins. In those ecosystems, each parallel chain has its own coin, thereby eliminating the possibility of double-spending across different chains.

In this paper, we propose a new design for maintaining parallel chains in 5ire network. Our design, henceforth called "nested chain" does not rely on additional coins, rather the same token can be used across all the parallel chains. Our technique increases the scalability of 5ire chain by many folds while preventing double spending of a token.

2.0 LITERATURE REVIEW

Sharding in distributed ledgers was conceptualized by Danezis et al. [2], and Croman et al. [1]. The goal of sharding is to split up the task of consensus among different nodes, for improving network throughput and reducing computational and storage overhead on a single node. Luu et al. [5] maintains one single chain but divides mining with respect to different

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

shards to different committees. Kokoris-Kogias etal. [4] presented Omniledger that uses ByzCoinX, a sharding approach that parallelizes block commitments by analyzing and handling dependencies between transactions. Zamani et al. [7] proposed Rapidchain, the first sharding-based blockchain protocol requiring a sublinear number of bits exchanged in the network per transaction. They propose a novel technique to partition the blockchain so each node would need to store only a fraction of the entire blockchain. In RapidChain, committees discover each other via an efficient routing mechanism that reduces latency and storage. Martino et al. [6] proposed chain web, a parallel chain PoW architecture that combines thousands of individually mined peer chains into a single network and in which each chain mines the same cryptocurrency that can be transferred to other chains via a trustless, two-step Simple Payment Verification at the smart contract level. In [3] proposed by Forestier et al., each block cross-references the most recent block of every chain.

3.0 DESCRIPTION OF NESTED CHAIN

We aim to solve the scalability issue by introducing a new concept called nested chains. In 5irechain, a transaction pool corresponds to a chain. When the number of transactions in a pool exceeds a certain threshold, the transaction pool splits, giving rise to two different pools. There are certain actors called joiner nodes in the network. When a particular transaction pool gets overloaded with transactions, the joiner nodes send a split transaction into the network. If the transaction is included in a block, the current chain splits, and from that point, two different chains emerge. Each of the chains has its own transaction pool with transactions taken from the parent pool. We use a hash function to classify trans-actions for inclusion into either pool. The sender address in every transaction is hashed, and depending upon the first few bites of the hash output, a unique transaction pool is chosen as the destination pool for that transaction. Let us assume that at time t = t0, there is one transaction pool P in the network. The transactions in P are denoted as T1, T2, . . . , Tn. If n is higher than the threshold value, then a split transaction is sent by the joiner nodes in the network. We assume that the 'split transaction' is included in the current block B0. Then P gets divided into two pools P0 and P1. Let Hash be an SHA-256 hash function. For each $i \in [1, n]$, if $Hash(PKTi) = 0||\{0, 1\}*$, then Ti will be included into pool P0, else it will be included into P1. This is depicted in Figure 2. Here, PC is the public key of the sender of the transaction Ti. Each of them will have its own chain that will emerge from the original chain. Let us say that the original chain was C, and the two chains emerging from C are denoted as C0 and C1. C0 and C1 correspond to pool P0, and P1 respectively. So, the first block in both C0, and C1 will include the hash of B0. Hence, B0 will be the predecessor of the first blocks of both chains. The two chains C0, and C1 will progress independently to each other. If any of the two chains need to be split the joiner nodes again send a 'split' transaction intended for the particular chain. This transaction is included in a block on that chain which causes the chain to split again as is shown in Figure 1. Every chain and its corresponding transaction pool have an identifier. The original chain has identifier X which is same as the original transaction pool. If the chain gets split, two new chains are created. They have ids X0 and X1 respectively. The corresponding transactions pools also have the same ids. If chain/pool X0 gets split again, there will be two new chains/pools having ids X00, and X01. In general, if chain/pool $X||y1||y2|| \cdot \cdot \cdot ||yc$ gets split,we get two new chains/pools with ids $X||y1||y2|| \cdot \cdot \cdot ||yc||0$, and $Xy1y2 \cdot yc1$, where $y1, y2, \cdot \cdot \cdot, yc \in \{0, 1\}$. If a transaction T appears in the network, it will go to the transaction pool $X||y1||y2|| \cdot \cdot \cdot ||yc$, where $Hash(PkT) = y1||y2|| \cdot$

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

· · ||yc||{0, 1}∗. PkT is the public key of the sender of T . This will ensure that all the transactions send by the same node will go to the same pool. The method of including transactions into different pools is shown in Algorithm 1.

If the load on the network drops, then chains that emerged from a parent chain has to be merged again. Let us assume that at some point in time there are two transaction pools X||y1||y2|| · · · ||yc||0, and X||y1||y2|| · · · ||yc||1 that came into existence after the division of the pool X||y1||y2|| · · · ||yc. The number of transactions into the two pools X y1 y2 yc 0, and X y1 y2 yc 1 have dropped below a certain threshold. Now, the joiner nodes send a 'merge' transaction aimed at the two pools. When the transaction is included in blocks of both the chains, mining stops on these two chains. Then the joiner nodes create a joiner block that causes the chains to merge into one.

Algorithm 1 Algorithm to include transactions into pools.

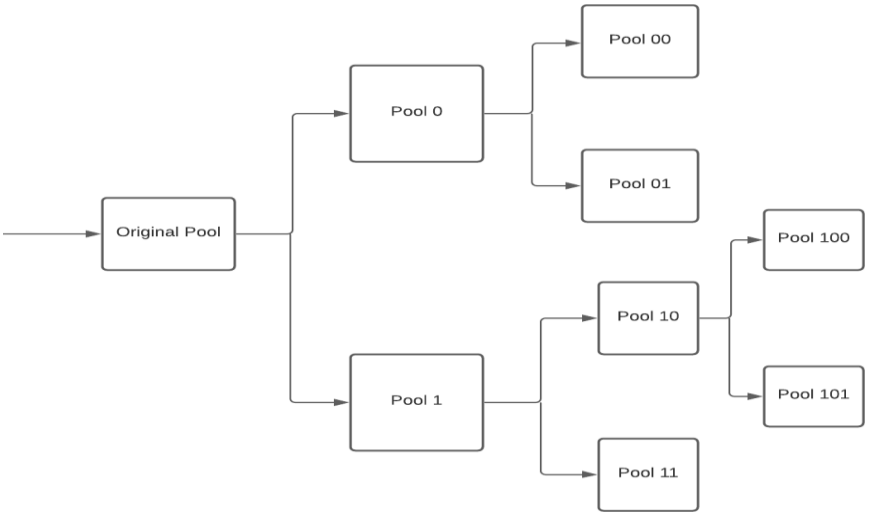
Require: a transaction T, the set P of ids of transaction pools. $SenderWallet()$ returns the sender address of a transaction. $Hash$ is a hash function.

Ensure: $i : i \in P$ the index of transaction pool where T is sent.

$x \leftarrow SenderWallet\ (T)$
$y \leftarrow$
$Hash(x)\ z$
$= |y|$
while $P \models \emptyset$ **do**
  $t \xleftarrow{\$} P$
  if $y_{z-1}y_{z-2} \cdots y_{z-|t|} = t$ **then**
    Return t
  end if
  $P = P \setminus t$
end while

4.0 JOINER NODES

There are certain actors called joiner nodes in the 5irechain ecosystem. The joiner nodes manage the creation, maintenance, and termination of nested chains. First,



Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

Fig. 1. Nested Chains

These nodes send split instruction that causes a transaction pool to split into two. The joiner nodes come from the set of block assemblers. The responsibility of a joiner node is to assemble a joiner block using a consensus mechanism and record the information of all preceding nested chains into it. A joiner block is created after each epoch in order to synchronize the state of 5irechain across the nodes. It is possible to increase or decrease the number of nested chains after each joiner block depending on the load on the network.

Figure 2 shows the different parameters of a split transaction transmitted by the joiner nodes. The transaction contains the public keys of the joiner nodes and the signatures along with other necessary parameters. Note that joiner nodes produce the split/join transaction in a unified manner. That is, the nodes send a common transaction together, instead of each node sending a separate transaction.

The joiner nodes are chosen from the set of block assemblers. There are many nodes chosen as block assemblers in an epoch. A group of block assemblers can create a split/join transaction in an epoch if the total weight of the block assemblers is at least 50% of the total weight of all the block assemblers in that epoch. The block assembler who includes a split/join transaction in a block must verify that this condition holds for the signers of the said transaction.

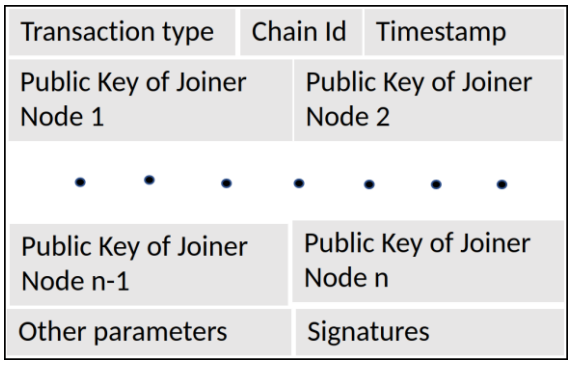


Fig. 2. Split transaction

5.0 DOUBLE SPENDING

One of the factors that determine the stability of any cryptocurrency network is its ability to prevent double-spending. Double spending is the event where the same coin is used more than once by the sender. This has to be prevented in order to maintain the consistency of the system. Cryptocurrencies that support parallel chains or shards need to ensure that the same coin does not get spent in two parallel chains. This could happen when two different transactions on the same coin get included in two different parallel chains. That is why conventional cryptocurrencies use different coins in distinct chains. Our 5irechain network prevents double-spending by making sure that all the transactions from the same sender go to the same transaction pool. If there are two different transactions having the same sender key, they will be included in the same pool. So, it is up to the block assembler to include either of them that would cause the other one to be discarded automatically. In our 5irechain, the block assembler of any slot only needs to scan the previous blocks along the same chain in order to

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

prevent double-spending, it does not need to read through the blocks in other parallel chains as all the transactions from a particular sender will be found on the blocks along the same chain. Thus, our 5irechain protocol evades the use of multiple coins in multiple chains.

6.0 ATTACKS ON THE PROTOCOL

In this section, we shall discuss some of the possible attacks on the nested chain protocol. We shall also discuss possible counter-measures against these attacks.

6.1 Selective wallet attack

As discussed above, our nested chain protocol classifies transactions depending upon the hash value of the senders' addresses. A powerful adversary may try to use this functionality to blow up one transaction pool by sending transactions to that pool from selected wallets. Let us assume that at some time there is one transaction pool A in the network. An increased influx of transactions causes the pool and the corresponding chain to split into two, namely pool A0, and

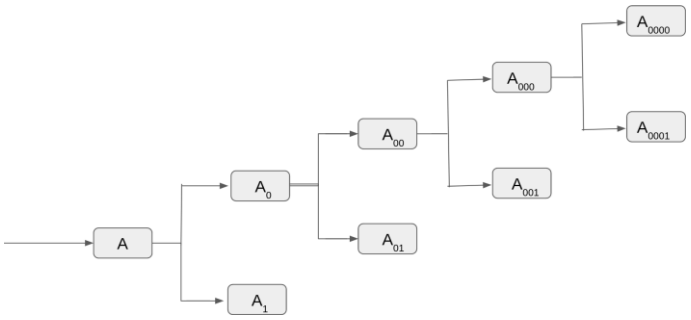


Fig. 3. Selective wallet attack

A1. Now attacker A wants to launch a selective wallet attack on the network. It sends many transactions to pool A0 from selected wallets that cause the transactions to go into pool A01. After a sufficient number of transactions are shoved into pool A01, it is split again into two pools A010, and A011. This way, the attacker can target one transaction pool and can flood it with transactions while other pools run dry. This scenario has been depicted in Figure 3. Under normal circumstances, the load on each of the transaction pools should be uniform as we determine the destination pool of a particular transaction depending upon the first few bites of the wallet address. However, a powerful attacker can overwhelm a particular pool by creating many wallets and sending tokens only from those wallets that would make the transactions go into that particular pool. This is achievable due to the fact that the number of transaction pools is always limited. It is needless to say that the attacker cannot continue with this attack for a long time as she would need to keep spending money for creating transactions. At some point in time, the attack would cease and chains would start merging back. It is also noteworthy that during the time of the attack, other chains might sit idle as they run out of transactions for assembling blocks. Keeping many chains idle is not desirable as it would increase the time to confirm a transaction that goes to an idle pool. Hence, it would be desirable not to have idle chains for a long time. One measure to achieve this will be to allow the out-of-norm merger of transaction pools and chains. That is, the joiner nodes can be allowed to merge any two of the existing pools/chains instead of merging only those

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

pools/chains that emerged from the same pool/chain. This has been depicted in Figure 4. In this figure, two transaction pools that did not originate from the same pool are merged together. These pools are A1 and A001. Once they are joined, all the transactions that were destined for A1, and A001 will now go into the combined pool A1 +A001. In other words, now the combined pool will contain all the transactions T, such that if WT is the sender's address of T, then Hash(WT) will start with either the bit 1 or the bit string 001. The joiner nodes need to merge the corresponding chains so the block assemblers and other nodes can know which pools are being merged, so the block assembly can continue smoothly.

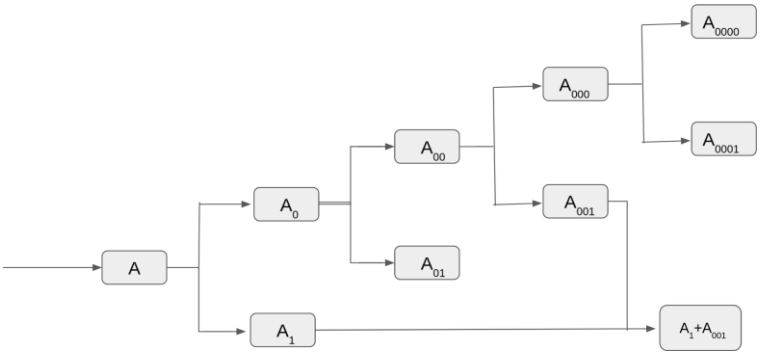


Fig. 4. Out-of-norm merger of transaction pools.

7.0 CONCLUSIONS

In this paper, we propose a new technique to scale up the number of transactions handled by the 5ire network. Our technique does not require additional tokens and at the same time prevents double-spending by ensuring that the transactions sent by a particular sender will not be included in blocks on two different chains. Our technique is compatible with blockchains like 5irechain and can be adopted by any blockchain in existence. Our technique eliminates cross-referencing between parallel chains.

REFERENCES

Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gu¨n Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, Financial Cryptography and Data Security, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. IACR Cryptol. ePrint Arch., 2015:502, 2016.

Sebastien Forestier. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. ArXiv, abs/1803.09029, 2018.

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, 2018.

Volume: 05, Issue: 02 " March - April 2022"

ISSN 2582-0176

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, CCS '16, page 17–30, New York, NY, USA, 2016. Association for Computing Machinery.

Will Martino, Monica Quaintance, and Stuart Popejoy. Chain web: A proof-of-work parallel-chain architecture for massive throughput. 2018.

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, CCS '18, page 931–948, New York, NY, USA, 2018. Association for Computing Machinery.

