# A Master-Slave Chain Model for Multiple Blockchains

Xinsen Hu, XH, Hu

Beihang University, School of Computer Science and Engineering

Kai Hu, KH, Hu

Beihang University, School of Computer Science and Engineering

Siyuan Wang, SW, Wang*

Beihang University, School of Computer Science and Engineering

Qinwei Tong, QT, Tong

Beihang University, School of Computer Science and Engineering

In recent years, blockchain has attracted great attention because of decentralization, tamper resistance and traceability. However, a single blockchain is less able to handle a large number of transactions. Besides, interoperability among different blockchains remains a continuing challenge. In this paper, we propose a master-slave chain model (MSCM) for multiple blockchains, which provides flexibility and scalability for the blockchain system. In the model, atomic cross-blockchain transactions are enabled by the proposed cross-blockchain protocol. We also introduce a blockchain router algorithm to optimize the communications among different blockchains. The experimental results support the usefulness of MSCM as a means of performance improvement.

**CCS CONCEPTS** • Computer systems organization ~ Architectures ~ Distributed architectures ~ Peer-to-peer architectures • Software and its engineering ~ Software creation and management ~ Designing software ~ Software design engineering

**Additional Keywords and Phrases:** Blockchain, Multiple Blockchains, Master-Slave Chain, Cross-blockchain, Performance

**ACM Reference Format:**

First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

* Corresponding author.

## 1 INTRODUCTION

Blockchain [1] has broad application prospects due to its advantages of decentralization, tamper resistance and traceability. The application scenarios include digital asset trading, open data [2], contract certificate, data authentication [3], electronic currency, etc. Blockchain technology is gradually developing into the infrastructure of the future digital society.

However, the performance of a single blockchain is largely limited. For example, Bitcoin [1] and Ethereum [4, 5] can separately handle about 7 and 20 transactions per second respectively. EOS can reach 2000 transactions per second due to the more centralized consensus algorithm delegated proof of stake (DPoS) [6]. In fact, in scenarios such as finance that require high transaction processing performance, existing blockchain systems cannot yet support practical applications. In addition, various business needs are difficult to meet within the same blockchain system. Therefore, as more and more blockchains are emerging, interoperability and cooperation among multiple blockchains are needed for more complex and richer applications.

In this paper, we propose a master-slave chain model (MSCM) for multiple blockchains and design a cross-blockchain protocol to enable cross-blockchain transactions. Taking the bank scenario as an example, in the banking system, there is the only digital asset issuance and management organization, which is responsible for the operation of the master blockchain. As the business of the banking system develops, the master blockchain will gradually become the bottleneck of the system. At this time, new slave blockchains can be dynamically created and added to the master-slave chain system to handle part of the transactions. For example, the branch institution creates a slave blockchain to handle transactions in the corresponding region, and the slave blockchain can be customized according to specific business needs. The master-slave chain system processes transactions according to certain rules, including local transactions and cross-blockchain transactions. In summary, our contributions are:

- We propose a master-slave chain model which provides flexibility and scalability for multiple blockchains.
- We design a cross-blockchain protocol which ensures the atomicity and termination of cross-blockchain transactions in the master-slave chain model.
- We propose a blockchain router algorithm to increase the query speed.

The remainder of this paper is divided into five sections. Section 2 presents the related work. Section 3 introduces our proposed master-slave chain model, followed by the cross-blockchain protocol in Section 4. Section 5 describes the experiments and analysis. And Section 6 concludes the paper.

## 2 RELATED WORK

Various cross-blockchain technologies are proposed by the industry to solve communication problems between blockchains. Pegged Sidechains [7, 8] use Simplified Payment Verification (SPV) [1] to verify the payments in the source blockchain, which, however, rely on the source blockchain using Proof of Work (PoW) [1] consensus algorithm, and the hashing power of the source blockchain being large enough to ensure security. The notary mechanisms introduce a trusted third party as an asset custodian in the cross-blockchain process, which therefore results in some degree of centralization. The relay schemes mainly use relay chains to coordinate the interconnection of blockchains and have different specific implementations.

### 2.1 Pegged Sidechains

The pegged sidechains solution was first proposed to allow Bitcoin to be transferred between the Bitcoin blockchain and other blockchains, so that Bitcoin users can use other blockchain systems. Liquid developed by the BlockStream team is a sidechain of Bitcoin, mainly to meet the needs of shorter transaction confirmation time. All Bitcoins transferred to Liquid will be frozen in the Bitcoin account. Once the Liquid blockchain notices that the corresponding Bitcoin transaction has taken place, the same amount of Liquid assets (called L-BTC) will be released on the Liquid blockchain. If users want to transfer L-BTC back to Bitcoin, they need to first destroy the L-BTC on the Liquid blockchain, and then receive the same amount of Bitcoin from the frozen Bitcoin account.

### 2.2 Notary Schemes

The notary mechanisms use one or a group of notaries to confirm the transactions between the two parties and complete the locking and unlocking of assets on different blockchains. They can be categorized into single signature notary scheme, multi-signature notary scheme and distributed signature notary scheme. The single signature notary scheme designates an independent node or institution as the notary, which is the simplest way. The multi-signature notary scheme selects a group of notaries. Each notary has its own key. Only when the number of notaries' signatures reaches a certain number or ratio can the cross-blockchain transactions be confirmed. The distributed signature notary scheme uses a distributed signature with multi-party computation. The system splits the unique key into key fragments and distributes them to each notary node. Only when signed by a certain percentage of notary nodes can the cross-blockchain transactions be confirmed.

### 2.3 Relays

The relay schemes are more flexible and do not rely on third parties to verify the transactions. The relay only forwards messages, and the receiving chain verifies the data by itself. In the BTC-Relay [9] solution, the relay node has an Ethereum account, and can obtain block headers from the Bitcoin network and submit them to the BTC-Relay smart contract [10] on Ethereum. The contract receives the block header and verifies it with SPV. When the corresponding payment behavior proves to have occurred in the Bitcoin network, it can trigger the execution of a specific Ethereum smart contract, thereby enabling the transfer of Bitcoin to Ethereum. Cosmos [11] is a cross-blockchain platform in which different blockchains are called Zones. Those Zones communicate transactions via a Hub blockchain using Inter-blockchain Communication (IBC) which relies on Tendermint [12] consensus algorithm. However, the IBC does not provide guaranteed atomic behavior. For example, there is no guarantee that an IBC packet's datagram is included in both the source and destination blockchains [13]. As a comparison, the cross-blockchain protocol in MSCM directly guarantees atomic behavior of cross-blockchain transactions. Polkadot [14] is another cross-blockchain platform which consists of Relay-chains, Parachains and Bridges. Relay-chains provide shared consensus for all Parachains. Parachains receive and process transactions. And Bridges are responsible for connecting existing blockchain systems. The shared consensus mechanism requires validators to be randomly assigned to Parachains, which therefore is not suitable for permissioned blockchains. In contrast, permissioned blockchains are well supported in MSCM.

## 3 MASTER-SLAVE CHAIN MODEL

### 3.1 Overview

In MSCM, the system comprises a master blockchain and at least one slave blockchain. As shown in Figure 1, the master blockchain interacts with slave blockchains while the slave blockchains do not interact with each other. Both the master and slave blockchains can receive and process transactions proposed by users. When a transaction is a cross–blockchain one, it needs to be forwarded by the master blockchain to the corresponding slave blockchain. The master blockchain is also responsible for the management of the slave blockchains. New slave blockchains can be created according to specific requirements.

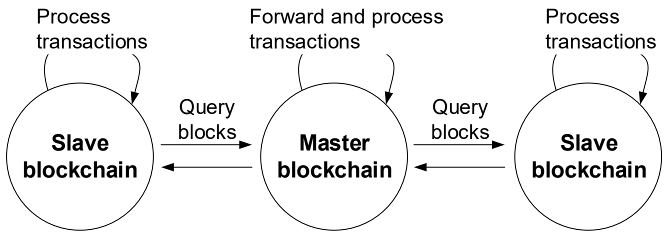

Figure 1: Master-slave chain model.

### 3.2 Account Address Structure

MSCM adopts the account-based accounting method. An account address in MSCM includes a chain ID and an address. The chain ID is the identifier of the blockchain to which the account belongs. The address is derived from the public key of the account. Using this structure, the account is uniquely identified in the multi-blockchain system.

### 3.3 Transaction Format

Blockchain is a distributed ledger which is updated via transactions. Users submit signed transactions to nodes of the blockchain. The nodes communicate those transactions to each other and propose a block containing them. Then the nodes reach a consensus on which block can be added to the end of the blockchain.

Transactions include local transactions and cross-blockchain transactions. We define the standard transaction format in Figure 2. The source blockchain and destination blockchain of a local transaction are the same while the ones of a cross-blockchain transaction are not. Cross-blockchain transactions involve updates on different blockchains, which indicates that the associated updates should be committed on both blockchains or discarded on both blockchains.

The details of fields in Figure 2 are as follows:

- **Type:** the type of the transaction, denoted by $T_e$. Normal transaction and coinbase transaction are the most common types.
- **From:** the source account address of the transaction, denoted by $T_f$. It can be described as $T_f = (cid, addr)$ where $cid$ is the blockchain identifier and $addr$ is the address of the account.
- **To:** the destination account address of the transaction, denoted by $T_t$. This field has the same structure as $T_f$.

| | |
|---|---|
| Type: 8bit | Extra |
| From: 176bit | To: 176bit |
| PubKey: 512bit | Value: 32bit |
| Nonce: 32bit | Signature: 520bit |
| LThreshold: 64bit | GThreshold: 64bit |

Figure 2: Transaction format.

- **Public key:** the public key of the source account, denoted by $T_p$.
- **Value:** the amount transferred of the transaction, denoted by $T_v$. This field can also be used to represent the service fee for a contract if smart contracts are supported.
- **Nonce:** the number of transactions sent from $T_f$, denoted by $T_n$. This field prevents double-spending, as it is the order in which transactions are sorted.
- **Signature:** the signature for the transaction, which is generated by the private key and can be verified by the public key, denoted by $T_s$.
- **Local threshold:** local block height threshold, denoted by $T_l$. If the transaction is not included in the block of height less than or equal to $T_l$ in the source blockchain, it is considered as invalid.
- **Global threshold:** global block height threshold, denoted by $T_g$. This field applies only for cross-blockchain transactions. The master blockchain forwards them and generate associated receipts by analyzing the blocks of slave blockchains. Until the block of height $T_g$ of the master blockchain, if the master blockchain is still unable to confirm that the cross-blockchain transaction is included in the block of the destination blockchain, it will generate a receipt of failure for this transaction. In this case, we say that the transaction has timed out globally. In this way, all the cross-blockchain transactions can be finished in a certain time.
- **Extra:** field for particular usage, denoted by $E$. Combined with $T_e$, this field can be used to fulfill specific needs, for example smart contracts.

## 3.4 Block Structure

In a blockchain system, the block serves as the carrier of all information, and is synchronized to all the blockchain nodes through a consensus algorithm. Each block records the hash of the previous block to form a chain structure, thereby maintaining the state of the system. Figure 3 shows the block structure of MSCM. Most fields apply for both the master blockchain and the slave blockchains.

- **Timestamp:** the time when the block was created, denoted by $B_t$.
- **Previous hash:** the hash of the previous block, denoted by $B_{pre}$.
- **Hash:** the hash of the block, denoted by $B_{hsh}$.
- **Height:** the height of the block, denoted by $B_h$. The genesis block has 0 height.

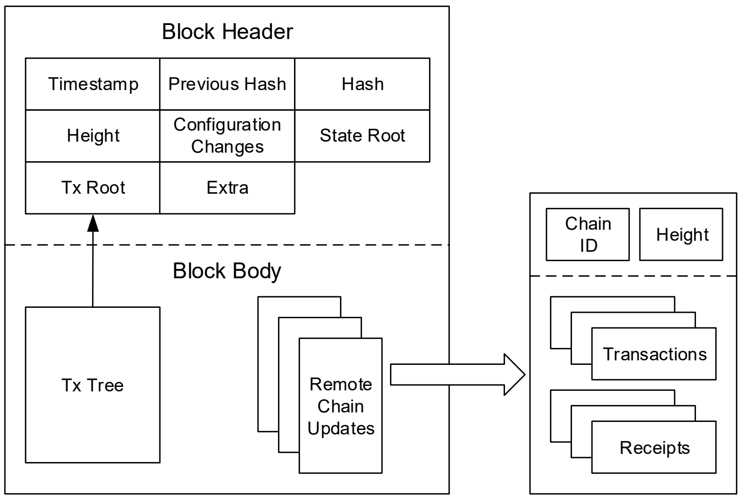

Figure 3: Block structure.

- **Configuration changes:** information for the management of the slave blockchains, denoted by $B_{cc}$. This field applies only for the master blockchain.
- **State root:** the hash of the root node of Merkle Patricia tree of accounts, denoted by $B_{sr}$.
- **Transaction root:** the hash of the root node of Merkle tree of transactions in the block, denoted by $B_{tr}$.
- **Extra:** field for particular usage, denoted by $B_e$. For example, when using Proof of Authority (PoA) [15] or Practical Byzantine Fault Tolerance (PBFT) [16] and other consensus algorithms with access permissions, nodes need to provide signatures for the block.
- **Transactions:** the transactions included in the block, denoted by $B_{txs}$.
- **Remote chain updates:** updates of other blockchains, denoted by $B_{us}$. Each update corresponds to a certain block of a certain blockchain. An update $B_u$ can be denoted as $B_u = (B_m, U_{cctxs}, U_r)$, where $B_m$ is meta information, $U_{cctxs}$ contains cross-blockchain transactions and $U_r$ contains receipts.

The master blockchain mainly uses $B_{us}$ to forward cross-blockchain transactions. For blocks of the master blockchain, cross-blockchain transactions sent by slave blockchains will be included into $U_{cctxs}$.

The slave blockchains mainly use $B_{us}$ to signal the consent to cross-blockchain transactions. For blocks of the slave blockchains, cross-blockchain transactions sent to the corresponding slave blockchain will be included into $U_{cctxs}$.

## 3.5 State Model

All nodes of each blockchain have a local state, and the consensus algorithm will ensure that the local state of all nodes of the same blockchain eventually converges to a consistent state. We use $\delta_c$ to denote the local state of a blockchain whose identifier is $c$, $\gamma_{Addr}$ to denote the account of which the address is $Addr$, and $\theta$ to denote the account address space. In addition, $B_{16}$ indicates that 16-bit storage space is occupied. So we have

$$\delta_c = \{\gamma_{Addr} \mid Addr^{cid} = c\} \tag{1}$$

where

$$\theta = \{(cid, addr) \mid cid \in B_{16}, addr \in B_{160}\} \tag{2}$$

$$\gamma_{Addr} = (Addr, Nonce, Balance) \tag{3}$$

$$Addr \in \theta \tag{4}$$

$$Nonce, Balance \in B_{32} \tag{5}$$

and $Addr^{cid}$ is the $cid$ of the address $Addr$.

Then the state of the whole system σ can be denoted as a set of states of the master blockchain and the slave blockchains, which is

$$\sigma = \{\delta_i \mid i \in B_{16}\} \tag{6}$$

According to the aforementioned transaction format, a transaction can be denoted as

$$T = (T_e, T_f, T_t, T_p, T_v, T_n, T_s, T_l, T_g, E) \tag{7}$$

where

$$T_f, T_t \in \theta \tag{8}$$

If we apply the valid $T$ transaction on σ, we can get the new state of the system $\sigma^{new}$, which is

$$\{(T_f, T_n, \gamma_{T_f}^{Balance} — T_v), (T_t, \gamma_{T_t}^{Nonce}, \gamma_{T_t}^{Balance} + T_v), \cdots\} \tag{9}$$

In (9), $\gamma_{T_f}^{Balance}$ is the balance of the account whose address is $T_f$; $\gamma_{T_t}^{Nonce}$ is the nonce of the account whose address is $T_t$; $\gamma_{T_t}^{Balance}$ is the balance of the account whose address is $T_t$.

## 4 CROSS-BLOCKCHAIN PROTOCOL

Transactions submitted by users need to be verified before being included in a block. If the signature $T_s$ is not valid, or $T_n \leq \gamma_{T_f}^{Nonce}$, or $T_v > \gamma_{T_f}^{Balance}$, or $\delta_{T_t^{cid}} \notin \sigma$ or $h \geq T_l$, where $h$ is the current block height of the source blockchain, the transaction will be ignored; otherwise, the transaction will be included in the new block.

The execution of a cross-blockchain transaction is more complex than that of a local transaction. Local transactions are executed directly while cross-blockchain transactions are processed by the cooperation of associated blockchains. Combining the two-phase commit and the timeout mechanism determined by block heights, we propose a cross-blockchain protocol for the master-slave chain model, which ensures the atomicity and termination of cross-blockchain transactions.

### 4.1 Transactions from Master Blockchain

Figure 4 shows the communications between master blockchain M and slave blockchain S1. We assume that a valid transaction $T1$ is submitted to the M blockchain, and the destination blockchain of $T1$ transaction is the S1 blockchain.

After the consensus of the block containing $T1$ transaction, nodes of the M blockchain pre-execute $T1$ transaction as the transaction may be rolled back later. The S1 blockchain includes $T1$ transaction in a block and then pre-executes $T1$ transaction. The M blockchain notices that the S1 blockchain has included $T1$ transaction, and will generate a receipt of success if the current block height of the M blockchain $h$ is less than $T_g$ of $T1$ transaction ($T1$ transaction has not timed out globally); otherwise the M blockchain will generate a receipt of failure. Upon receiving the receipt of failure, the M blockchain and the S1 blockchain will roll back $T1$ transaction.

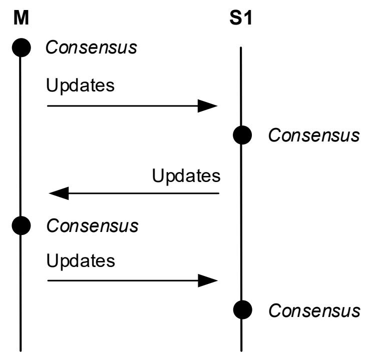

Figure 4: Communications between master blockchain M and slave blockchain S1.

## 4.2 Transactions from Slave Blockchain

Also in Figure 4, we assume that a valid transaction $T2$ is submitted to the S1 blockchain, and the destination blockchain of $T2$ transaction is the M blockchain.

Similarly, nodes of the S1 blockchain pre-execute $T2$ transaction. If the M blockchain notices $T2$ transaction before it times out globally, the M blockchain will include $T2$ transaction in a block of the M blockchain and generate a receipt of success, after which $T2$ transaction will be executed on the M blockchain; otherwise the M blockchain will generate a receipt of failure. If the receipt for $T2$ transaction is a receipt of failure, the S1 blockchain will roll back $T2$ transaction afterwards.

As shown in Figure 5, we assume that a valid transaction $T3$ is submitted to the S2 blockchain, and the destination blockchain of $T3$ transaction is the S3 blockchain.

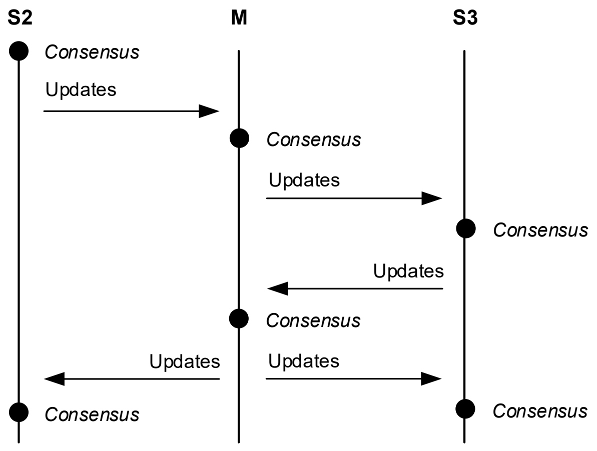

Figure 5: Communications between slave blockchain S2 and slave blockchain S3.

Similarly, nodes of the S2 blockchain pre-execute $T3$ transaction. If $T3$ transaction has timed out globally, the M blockchain will generate a receipt of failure; otherwise the M blockchain will include $T3$ transaction in a block of the M blockchain and wait for the destination blockchain S3 to respond to $T3$ transaction. The S3 blockchain includes $T3$ transaction in a block of the S3 blockchain and afterwards pre-executes it. If the M blockchain notices that the S3 blockchain has included $T3$ transaction before $T3$ transaction times out globally, it will generate a receipt of success; otherwise a receipt of failure will be generated. The S2 and S3 blockchains will roll back associated execution upon the receipt of failure.

## 4.3 Blockchain Router Algorithm

In MSCM, the master blockchain and the slave blockchains query blocks from each other. If all the queries are sent directly to the corresponding blockchain nodes, it will result in high query load on those nodes and long query time. We propose a blockchain router algorithm to increase the query speed and reduce the query load of corresponding blockchain nodes.

A blockchain router is a query agent for a blockchain miner node. Each miner node of the master blockchain will launch a corresponding blockchain router for each slave blockchain; each miner node of each slave blockchain will only launch a corresponding blockchain router for the master blockchain.

Miner nodes are allowed to set an appropriate trust list which includes the blockchain routers of other miner nodes in the same blockchain network, so that part of the query load is transferred to the blockchain routers in the trust list, which reduces the query load of nodes in the queried blockchains.

According to the properties of a queried blockchain, an appropriate query strategy needs to be adopted. For example, for PoW and other consensus algorithms that support blockchain forks, the strategy will require the queried block to meet certain confirmation conditions like several blocks have confirmed the queried block; for consensus algorithms like PoA with access permissions, the strategy will require nodes to sign the block.

The query process can be described by the Algorithm 1. First, the blockchain router queries the blocks from the routers in the trust list. If the corresponding blocks are successfully queried at this stage, the query process is directly ended. Second, according to the query strategy, the blockchain router concurrently queries the blocks of the corresponding blockchain. Multi-thread technology is used to increase the query speed.

ALGORITHM 1: Query Blocks Algorithm

```
Input: Chain ID cid, Height h, Limit l, Max Thread Number t
Output: A list of Blocks
begin
  routerTrustList ← getRouterTrustList()
  forEach router in routerTrustList do
    ok, blockList ← queryRouter(router, cid, h, l, t)
    if ok then
      return blockList
    end if
  end forEach
  queryStrategy ← getQueryStrategy(cid)
  tMax ← min(getMaxThreadNumber(queryStrategy), t)
  queryWorkers ← createWorkerSet(tMax)
  forEach worker in queryWorkers do
    queryHeight, queryLimit ← calculateQueryTask(h, l, tMax, worker)
```

```
        runQueryTask(worker, queryHeight, queryLimit, queryStrategy)
    end forEach
    queryResults ← []
    forEach worker in queryWorkers do
        gatherQueryResults(queryResults, worker)
    end forEach
    return queryResults
end
```

## 5 EXPERIMENTS AND ANALYSIS

Using Raft as the consensus algorithm, we implemented a prototype of MSCM in Go language. The experiments were conducted on 4 servers. When multiple blockchains are launched and each blockchain consists of 4 nodes, one server will run multiple nodes of different blockchains, but there is no resource sharing between these nodes. We use a simulation program to simulate users submitting transactions.

The simulation program tests the performance of the system from two perspectives: single blockchain and multiple blockchains. When the master blockchain is the only blockchain, all the transactions are local transactions. The average transaction processing speed is measured as about 1684 transactions per second (TPS) when transaction timeout begins to appear, which means the system has reached a bottleneck. In the case of multiple blockchains, the simulation program submits cross-blockchain transactions with a probability of 20%, and local transactions with a probability of 80%. The results are shown in Table 1.

Table 1: Average transaction processing speed of multi-blockchain systems

| Number of slave blockchains | Cross-blockchain transaction speed (TPS) | Cross-blockchain transaction success rate | Total speed (TPS) |
|---|---|---|---|
| 2 | 656 | 99% | 2946 |
| 3 | 984 | 98% | 4457 |
| 4 | 1347 | 98% | 5725 |
| 5 | 1507 | 96% | 7198 |

In the case of too many transactions, due to the upper limit of the size of the transaction pool, some transactions may be removed from the transaction pool before being processed by the blockchain node, which explains why the success rate of cross-blockchain transactions is not 100%. Those failed cross-blockchain transactions are correctly rolled back according to the cross-blockchain protocol.

In addition, within the system of the master blockchain and 5 slave blockchains, the simulation program is set up to submit cross-blockchain transactions with a probability of 100%. The average transaction processing speed of the system is about 3328 TPS. Since the master blockchain uses blocks as a unit when packaging cross-blockchain transactions and only verifies signatures, the speed of the master blockchain for packaging cross-blockchain transactions is higher than that of directly processing transactions (the aforementioned 1684 TPS). Compared with the results from Table 1, the speed indicates that in this case, the transaction processing speed of the multi-blockchain system is directly limited by the processing capacity of the master blockchain.

Figure 6 displays the relationship between the average transaction processing speed of the system and the number of slave blockchains when cross-blockchain transactions are submitted with a probability of 20%.

We can conclude that when most transactions of the system are local transactions and the master blockchain has not become a bottleneck, as the number of slave blockchains increases, the overall transaction processing performance of the system can be improved almost linearly. By fitting the polyline in Figure 6, the result is y = 1417.3x + 111.4, that is, each additional slave blockchain can make the system process 1417 more transactions per second.

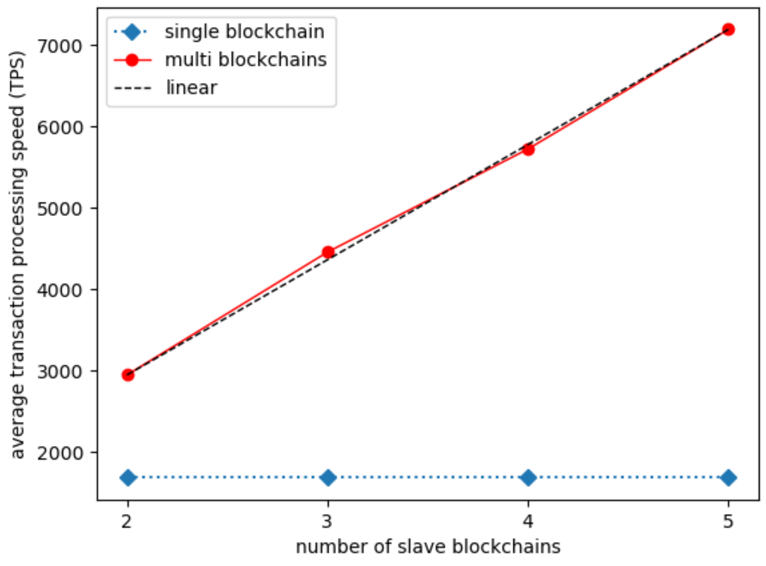

Figure 6: Transaction processing speed of different blockchain systems.

## 6 CONCLUSION

In this paper, we propose the master-slave chain model for multiple blockchains. In the model, the master blockchain coordinates the slave blockchains and new slave blockchains can be created to undertake partial work or perform specific tasks. For communications among those blockchains, we introduce the cross-blockchain protocol, along with the blockchain router algorithm. Judging from the experimental results, when most transactions are local transactions, MSCM can be regarded as a scalable way to improve the performance of blockchain systems. However, the master blockchain is prone to be a bottleneck as cross-blockchain transactions increase significantly. We plan to improve our model by introducing a mechanism for scheduling accounts between different slave blockchains based on transaction statistics to ensure that there are as few cross-blockchain transactions as possible. Furthermore, Byzantine fault tolerance has not been taken into account in our cross-blockchain protocol, which will be discussed in the future work.

## ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under Grant 61672074 and 61672075, the Project of National Key Research and Development of China under Grant 2018YFB1402702 and the Funding of Ministry of Education and China Mobile MCM20180104.

## REFERENCES

[1] Nakamoto, S. 2008. Bitcoin: A peer-to-peer electronic cash system.

[2] McGinn, D., McIlwraith, D., and Guo, Y. Towards open data blockchain analytics: a Bitcoin perspective. *Royal Society open science* 5, 8 (2018), 180298.

[3] Deery, B., Snow, P., and PAOLINI-SUBRAMANYA, M. Validating documents via blockchain, Sept. 2019. Patent No. 10419225, Filed Jan. 30th., 2017, Issued Sept. 17th., 2019.

[4] Wood, G., et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[5] Buterin, V., et al. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).

[6] Larimer, D. Delegated proof-of-stake consensus, 2014. Retrieved Sept. 23, 2020 from https://bitshareshub.io/delegated-proof-of-stake-consensus/.

[7] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Tim´on, J., and Wuille, P. Enabling blockchain innovations with pegged sidechains. *URL: http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* 72(2014).

[8] De Kruijff, J., and Weigand, H. Understanding the blockchain using enterprise ontology. In *International Conference on Advanced Information Systems Engineering* (2017), Springer, pp. 29–43.

[9] Chow, J. Btc relay, 2016. Retrieved Sept. 23, 2020 from https://buildmedia.readthedocs.org/media/pdf/btc-relay/latest/btc-relay.pdf.

[10] Clack, C. D., Bakshi, V. A., and Braine, L. Smart contract templates: foundations, design landscape and research directions. *arXiv preprint* arXiv:1608.00771(2016).

[11] Kwon, J., and Buchman, E. Cosmos: A network of distributed ledgers. *URL https://cosmos. network/whitepaper* (2016).

[12] Buchman, E., Kwon, J., and Milosevic, Z. The latest gossip on BFT consensus. *arXiv preprint* arXiv:1807.04938(2018).

[13] Robinson, P. Consensus for crosschain communications. *arXiv preprint* arXiv:2004.09494(2020).

[14] Wood, G. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper* (2016).

[15] De Angelis, S., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., and Sassone, V. PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain.

[16] Castro, M., Liskov, B., et al. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.