A Proof-of-Majority Consensus Protocol for Blockchain-enabled Collaboration Infrastructure of 5G Network Slice Brokers

Wenmin Lin
Institute of VR and Intelligent Systems, Alibaba Business School, Hangzhou Normal University
Hangzhou, China
wenmin.lin@hznu.edu.cn

Xiaolong Xu
Nanjing University of Information and Technology
Nanjing, China
xxl.nju@gmail.com

Lianyong Qi
Qufu Normal University
Rizhao, China
lianyongqi@qf.edu.cn

Xuyun Zhang
Department of Computing, Macquarie University
Sydney, Australia
xuyun.zhang@mq.edu.au

Wanchun Dou*
State Key Laboratory of Novel Software, Department of Computer Science and Technology, Nanjing University
Nanjing, China
douwc@nju.edu.cn

Mohammad R. Khosravi
Department of Electrical and Electronic Engineering, Shiraz University of Technology
Shiraz, Department of Computer Engineering, Persian Gulf University
Bushehr, Iran
m.khosravi@sutech.ac.ir

ABSTRACT

With network slicing technology, 5G changes the classical mobile network business model from a network-operator-oriented business to a more open system with multiple stakeholders. In this context, a network slice broker plays the role of agency between network resource providers and vertical service providers, to lease and provide on-demand network slice services. In practice, a single network slice broker normally owns limited types of network slices. And a collaboration infrastructure could be built up to enable network slice sharing among multiple network slice brokers, so as to improve user experience, and enhance the resource utilization rate. However, a major challenge with such an infrastructure is how to support fair and secure slice trading process among multiple un-trusted network slice providers. Blockchain technology, on the other side, brings in high opportunities to implement a fair and secure trading system without a trusted third-party. In view of this observation, this paper proposes a design of blockchain-enabled collaboration infrastructure for network slice brokers, where a group of registered brokers can trade their network slices in a fair and secure manner. To enhance the accountability of slice trading records, transactions are verified and sorted by each broker node, which are further packed into blocks and appended to a shared ledger maintained by each member in the collaboration network. More specifically, we design a Proof-of-Majority (PoM) consensus protocol to sort slice trading transactions before packing transaction blocks of the shared ledger. Furthermore, an optimization for the basic PoM protocol is discussed, with dependency resolving strategy to improve system performance. Experiments are conducted to verify the feasibility of our proposal.

KEYWORDS

collaboration infrastructure, network slice broker, blockchain, consensus protocol, proof-of-majority

ACM Reference Format:
Wenmin Lin, Xiaolong Xu, Lianyong Qi, Xuyun Zhang, Wanchun Dou*, and Mohammad R. Khosravi. 2020. A Proof-of-Majority Consensus Protocol for Blockchain-enabled Collaboration Infrastructure of 5G Network Slice Brokers. In *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '20), October 6, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3384943.3409421

1 INTRODUCTION

The 5G technology provides a completely new vision of mobile networks, where vertical industries can join to unify and simplify the management of networks [13, 16]. It will revolutionize the way network services being provided and consumed. With 5G technology, various mobile network services can be deployed, not only traditional broadband services, but also services with their own network technology, such as automotive services, smart home services, and other Internet of Things (IoT) based services, to name a few [31] [33].

The enabler behind these different types of 5G services is *Network Slicing*, which is intended to enable network operators to slice a single physical network into several network substrates, known as *Virtual Networks* [3, 25]. A virtual network normally refers to a specific network resource, such as VNFs (Virtual Network Functions), PNFs (Physical Network Functions), CPU, storage, RAN resources, transport network, etc. And any specific network service request can be implemented by combining several available virtual networks as a network slice with certified service levels.

Network slice broker is introduced by Samnadis [14], to facilitate the operation of creating network slices in 5G network architecture. A network slice broker leases resources from network operators to build network slices, and sell them to vertical industry service providers. Those sub-slices involved in a network slice may be provisioned by different administrative domains with different QoS performance, so as to carry diversified 5G services [10, 11, 23].

In practice, the sub-slices of a network slice can be dynamically updated, added and released according to the dynamic change of service requirements [28]. And it is not possible for a network slice broker to lease all types of sub-slices on its own. In this context, a network slice broker collaboration infrastructure is necessarily required to facilitate network slice sharing and trading among multiple brokers [9].

However, it is a challenge to organize multiple brokers as a whole to guarantee slice trading is processed in a fair and secure manner, since brokers within a collaboration does not trust each other. Moreover, a broker collaboration system is vulnerable to threats such as data tampering and fake identity. Therefore, there is an urgent need for an infrastructure that guarantee the security for a network slice broker collaboration system. More specifically, the major challenge with a broker collaboration system is consensus issue, i.e., how to achieve consistent slice trading records among distributed brokers.

1.1 Motivation and Challenge

Here, we take a motivating example to highlight the consensus issue and flesh out the problem statements. In a network slice collaboration system, slice trading happens between two brokers, e.g., C_p and C_c. Once C_p and C_c complete the process of a slice trading negotiation, a transaction recording the trading details should be broadcast to all members. And the transaction will be verified by each member in the collaboration system. Furthermore, once the transaction record passes verification, it will be recorded by each member permanently for further accountability usage.

The fundamental problem in such a collaboration system is how to design the infrastructure and enable broker nodes to reach identical decision on the validity of slice trading transactions, and store identical trading records locally. Moreover, there are three typical scenarios should be taken into consideration:

- In the distributed broker collaboration system, some external entities may interfere the consensus progress by sending disturbing messages with faked broker identities;
- A slice trading message may be manipulated during its delivery. As a result, each trading record should be verified regarding the authority of the information contained in the message;
- Multiple slice trading records could be raised concurrently, and they should be recorded in identical sequence on each node. In other words, the transactions should be stored in identical form on each participant node. Therefore, any single failed broker node will not impact the reliability of the whole collaboration system.

Blockchain technology [18, 19] [24] combines benefits of both peer-to-peer network and cryptographic algorithms, making it as a possible solution to address aforementioned problems in a network slice broker collaboration system. Generally, each node in a blockchain network has an unique identity such as public address. Such information could be registered on each node in advance, to help other nodes to verify the identity of request initiator; also, the messages delivered over a blockchain network is signed with sender's signature, which could help to verify the authority of the information with a slice trading record; Moreover, consensus protocol in a blockchain network [5] could help each node to reach consistency regarding the sequence of each slice trading record, in form of packed and linked blocks.

In view of this observation, Blockchain provides a possible solution to build a practical decentralized network slice broker collaboration system. Moreover, typical blockchain application such as Bitcoin crypto-currency, can be used as a reference model to build such an infrastructure. In Bitcoin [21], the most critical technology is the consensus protocol, which enables the establishment of the source of truth with transaction records in the absence of a central authority. Bitcoin adopts "Proof-of-Work (PoW)" as the underlying consensus protocol. It takes about 10 minutes to generate a transaction block, and 1 hour to confirm the success of a transaction. The reason for such low throughput is due to the heavy power consumption of its participants, to compute a randomized nonce value. A report estimates that Bitcoin's electricity consumption of the Bitcoin network may draw over 14 Gigawatts of electricity by 2020, which is equivalent to the electricity consumpution of the whole Denmark [6]. In contrast, other main stream consensus protocols in blockchain applications such as Proof-of-Stake (PoS) consensus have weak security and poor fairness issues [4].

The weaknesses of existing popular blockchain consensus algorithms discussed above pose a significant challenge to implement a practical broker collaboration infrastructure with efficient consensus protocol. In order to build such a broker collaboration infrastructure, certain trade-offs need to be made, with the aim to achieving following goals.

- Security: the consensus protocol must be able to reach consistency decision even with failure of some single nodes;
- Performance: the consensus protocol should be able to provide low latency and high throughput, regarding small number of broker nodes within a network slice collaboration system.

1.2 Our contribution

In this paper, we aim to designing a practical collaboration infrastructure for network slice brokers, where the core challenge is the consensus protocol to guarantee the security and performance of the overall system. The main contributions are summarized as follows:

- We propose a permissioned blockchain-enabled network(i.e., consortium network) as the underlying network model for slice broker collaboration infrastructure, where each broker within a same collaboration network knows each other in terms of identity information(e.g., public address, MAC address, etc). Therefore, requests sent by external entities will be ignored by the collaboration network;
- A Proof-of-Majority(PoM) consensus protocol is designed to generate the shared ledger of slice trading transactions.

As long as majority of brokers are alive, the system will reach consensus regarding the sequence of each slice trading records, which will be further packed into linked blocks.
- To optimize the performance of basic PoM consensus protocol, we design an out-of-order transaction packing rule to improve the performance of the system regarding latency and throughput.

The reminder of this paper is organized as follows: In Section 2, we discuss the related work regarding network slice broker collaboration systems, as well as consensus protocol in blockchain applications. Section 3 presents a blockchain-enabled infrastructure for network slice broker collaboration system. Section 4 discusses the detailed Proof-of-Majority consensus protocol. In Section 5, we propose an optimization for the basic PoM consensus protocol from perspective of system performance. Section 6 analyses the security and performance of the PoM consensus protocol. And Section 7 evaluates the feasibility of our proposal with experimental evaluation and analysis. Section 8 concludes the paper and points out the future work.

2 RELATED WORK

2.1 Network slice broker collaboration systems

Network slicing is a key enabler technology for 5G applications, which is to satisfy the diversified requirements of different vertical industry services. In general, a network slice consists of a set of sub-slices, which represents different type of network resources. Based on different QoS of network resources, network slices can be roughly classified into three categories[2]: eMBB (enhanced Mobile Broadband) requiring very high bandwidth and data rate (e.g., Virtual Reality applications), uRLLC (ultra Reliable and Low Latency Communications) requiring very low latency access (e.g., automated driving services), and mMTC (massive Machine Type Communications) requiring support massive devices accessing simultaneously (e.g., IoT based services).

The sub-slices involved in a network slice could be provided by a single resource provider (i.e., all sub-slices are provided by the same operator), or from different resource providers. To facilitate the network slice creation process, network slice broker is proposed as a new component in 5G architecture [14]. The 5G network slice broker mainly in charge of (*i*) interfacing with external network tenants regarding their network slice request, (*ii*) managing the life-cycle of network slice regarding slice instantiation/resize/maintenance/deletion. (*iii*) monitoring slice traffic.

In practice, a single network slice broker may leases limited type of network resources, and it may be difficult to satisfy all its users' diversified requirements. Broker collaboration, on the other hand, is a promising solution solve this issue of resource limitation, by enabling network slice brokers share their slices as one network slice infrastructure. Collaboration is also known as federation, which gains wide popularity in both industry and academia. Typical collaboration example is cloud federation [17, 20] [32]. For example, in [17], a broker and federation architecture for resource allocation with end-to-end SLA is proposed in cloud networking environment. And in [20], a model of Virtual Cloud using "Rent Out the Rented Resources" is defined based on renting 3rd-party resources to provide its own services. In [32], the authors studies a blockchain-based cloudlet management framework to manage various multimedia workflows provided by different vendors.

The goal of network slice broker collaboration is to coordinate requests and supplies of network slices among a set of brokers, so as to overcome resource limitation issues of single entities. Moreover, broker collaboration could promote more efficient collaborative and networked business ecosystems [1, 7, 27]. However, the literature regarding network slice broker collaboration is still at its early stages, and existing works related to network slice broker collaboration is mainly focusing on business model, slice management as well as orchestration of cross-domain sub-slices [8, 9, 12, 14, 30]. In [8], an incentive and reputation driven collaboration model is developed to support cooperative network architecture and revenue generation, as well as innovative services enabled by brokerage. In [12], a business model supporting various levels of collaboration among resource provider, connectivity provider and service provider is discussed in details. In [14], a preliminary 5G network slice broker supporting network slices instantiation on the Radio Access Network has been fully studied and devised. In [9], a brokering architecture for network slicing is proposed, where a network operator can federate IT and network resources owned by third-party actors located at the edge of the network. In [30], a Reinforcement Learning-based 5G Network Slice Broker (RL-NSB) is discussed for solving the NP hard network slicing scheduling problem.

Different from existing related work on network slice broker collaboration, our work mainly focus on design a decentralized infrastructure based on blockchain technology, where network slice sharing is implemented among multiple slice brokers in a peer-to-peer manner. Several related works applied blockchain technology to implement decentralized network slice broker applications as well. For example, Boubakr et al. [2] explores a blockchain based network slice broker to build on-demand network slice via secure and private sub slices trading among network resource providers. Jere et al. [26] also explores blockchain network slice broker to reduce service creation time via smart contracts. Those works mainly adopts blockchain as a supporting tool with secure and decentralization features; while our work mainly focus on improve the system performance of blockchain enabled applications by designing a proper consensus protocol for the decentralized network slice broker collaboration system.

2.2 Blockchain consensus protocol

One important factor affecting the wide adoption of blockchain-enabled applications is system performance, known as efficiency to confirm a transaction in Bitcoin as well. Consensus protocol, on the other hand, plays the decisive role to determine the throughput and latency for blockchain applications [29] [22] [15]. Popular consensus protocol for blockchain applications are including PoW(Proof-of-Work) for Bitcoin, PoS(Proof-of-Stake) for Etherum, and PBFT(Practical Byzantine Fault Tolerance) for Hyperledger Fabric [5]. Among them, PoW requires quite a lot of computing power to compete the authority to record each block. While PoS is an subjective protocol where nodes with more Etherum coins have higher probability to own the authority to record each block. PBFT protocol, however, is a message based consensus protocol involving multiple rounds of participant voting, which is suitable

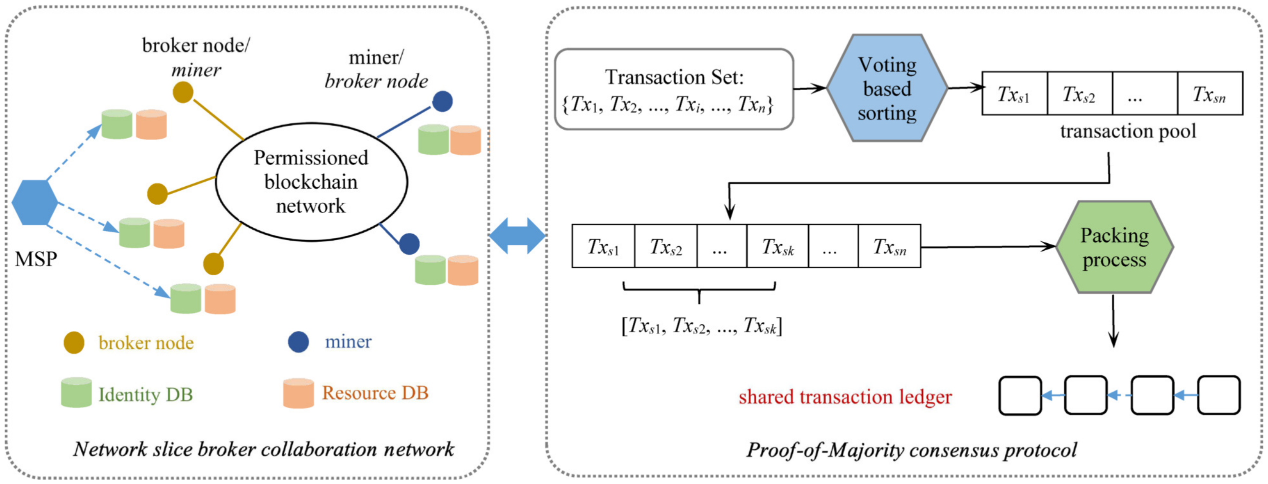

broker node/ miner
miner/ broker node
Permissioned blockchain network
MSP
broker node
miner
Identity DB
Resource DB
Network slice broker collaboration network
Transaction Set: $\{Tx_1, Tx_2, ..., Tx_i, ..., Tx_n\}$
Voting based sorting
Tx_{s1} Tx_{s2} ... Tx_{sn}
transaction pool
Tx_{s1} Tx_{s2} ... Tx_{sk} ... Tx_{sn}
$[Tx_{s1}, Tx_{s2}, ..., Tx_{sk}]$
Packing process
shared transaction ledger
Proof-of-Majority consensus protocol

Figure 1: Overview of the blockchain-enabled infrastructure model for broker collaboration systems

for applications with limited replica nodes. The similarity with those consensus protocols is they are all designed for applications to overcome Byzantine fault issues. But they suffer from issues such as waste of energy, non-objective, and heavy network overhead, to name a few.

To overcome issues with existing popular blockchain consensus protocols, researches on how to improve the block generation rate are studied. Jun et al. proposed a Proof-of-Trust consensus protocol [4], to enhance the accountability of transactions with a crowdsourcing platform. In their design, blocks are generated with two major phases. In the first phase, crowd workers with good trustworthiness value are elected to propose the transactions to be packed in a given block; and in the second phase, the crowdsourcing platform stakeholders will vote on the transactions obtained in first phase, to determine the content in the given block. In [5], Yu et al. proposed a new crypto-currency named Repucoin to overcome reputation issue caused by more than 1/3 evil validators within the system. Briefly, the consensus result is determined by majority of validators, whose accumulated reputation is not less than the total reputation value within the validator group as well.

Compared with existing works on blockchain consensus protocol, our Proof-of-Majority consensus protocol is different from two perspectives:

(1) The consensus protocols mentioned in related work are designed for Byzantine Fault applications, where nodes may be evil and send incorrect messages over the network. We simplify the application scenario by assuming each broker in the blockchain network are civil. That means a broker may fail or timeout regarding a request, but it will not send wrong messages or responses over the network. And to guarantee the security of PoM consensus protocol, all broker nodes must register their identity information and resource information in advance to all nodes within a blockchain network. As a result, messages sent from external entities will be ignored.
(2) Another issue with existing blockchain consensus protocols (e.g., PoW and PoS) is throughput and latency limitation. In practice, a valid block may be discarded when multiple miners raise a correct block simultaneously, and only the one wins the right to produce the block of a given sequence number. As a result, transactions in other blocks are invalid and will be put back to transaction pool, waiting for another block generation round. To address this limitation, we sorted the transactions on each node before generating the blocks for the shared ledger. And the sequence of the transactions are determined through majority voting process. Moreover, we design an out-of-order block generation algorithm to improve the system performance.

3 BLOCKCHAIN-ENABLED INFRASTRUCTURE FOR NETWORK SLICE BROKER COLLABORATION SYSTEM

3.1 System Model

As depicted in Fig. 1, a network slice broker collaboration system consists of two major components. The left-side component is a decentralized peer-to-peer network formed by multiple registered brokers. We adopt permissioned blockchain(i.e., consortium blockchain) as the overlay network, where each broker must get itself registered to the membership service provider(MSP) node. In other words, the MSP node maintains the identity information of each broker. Each broker can get a full list of registered brokers from MSP, so as to guarantee slice trading is conducted within the permissioned blockchain network. Moreover, each slice broker also need to get its slice resources registered to the MSP node as well. Such information will be referred by each broker to validate slice trading transactions. Over the network, brokers can initiate slice requests, validate slice trading transactions, and pack slice trading transactions into blocks by exchanging messages with each other. The right-side component is the Proof-of-Majority consensus protocol to produce the shared transaction ledger, which keeps a tamper-resist proof of slice trading transaction records among the broker nodes in left-side permissioned network.

Definition 3.1. Blockchain-enabled network slice broker collaboration system: A blockchain-enabled network slice broker collaboration system can be formalized as a tuple $< C, P, L >$, where C is the permissioned blockchain network consisting of n brokers, P is the Proof-of-Majority(PoM) consensus protocol applied to generate shared ledger of slice trading transactions, and L is the shared ledger maintained by each member in C.

For each broker node C_i in a permissioned blockchain network, it maintains two types of databases: identity information database $IR - DB$ and resource information database $RR - DB$. $IR - DB$ maintains identity information registered by broker nodes within the same collaboration network, i.e., $IR - DB = [IR_1, IR_2, ..., IR_n]$. $IR_i (1 \leq i \leq n)$ describes the identity information of peer broker node C_i, which could be formed as a key-value pair: $IR_i =< Key_i, Identity_i >$,where Key_i is the key data such as public address of C_i; $Identity_i$ is the detailed description of C_i, including the MAC address, port number, etc. The identity information in $IR - DB$ is fetched from the MSP node, which works as a white list to verify the identity of each message sender. While $RR - DB$ maintains the resource information owned by other peer brokers over the network, i.e., $RR - DB = [RR_1, RR_2, ..., RR_n]$. $RR_i = \{< S_1, QoS_1 >, < S_2, QoS_2 >, ... < S_m, QoS_m >\}$, and $RR_i (1 \leq i \leq n)$ describes the slices owned by C_i, which is fetched from MSP node as well. As mentioned in aforementioned section, each broker should get its resource information registered on MSP node, and each peer broker node will fetch its peer broker's resource information from MSP in advance, which will be referred to verify a slice trading transaction by checking whether the owner has sufficient slice resources.

In general, four main activities are supported in the broker collaboration network:

Identity registering: Each broker node gets itself registered on MSP node when it joins the permissioned blockchain network. In our design, identity registering is implemented via sending a registering message $M - IR_i$ containing C_i's identity information to MSP node. On receiving an identity registering message from C_i, MSP node will verify C_i's identity and record C_i's identity information for later verification usage. If C_i is a valid peer node, IR_i will be fetched by other peer brokers(e.g., C_j), which will be further recorded into C_j's $IR - DB$.

Resource registering: Similar to identity registering process, each broker also need to get its available slice resources registered to MSP node in advance , which will be recorded on each peer nodes as well. Resource registering is also achieved via sending a message $M - RR_i$ to the MSP node. MSP node will firstly validate the identity of C_i and record the resource information RR_i contained in $M - RR_i$ to its resource database. And RR_i will be fetched by other peer brokers and recorded in each broker's local $RR - DB$.

Request negotiation: Request negotiation is to help a broker to find a preferred slice provider to overcome its resource limitation issue. A broker C_i first initiates a slice request R_i describing its requirement, and broadcast the request to all peers. On receiving R_i, a peer broker C_j will verify the identity of C_i, and sends back a response if it has sufficient slice resource to fulfil R_i. Moreover, C_i will select a preferred slice provider among all responses to fulfil R_i. Finally, it will initiate a slice trading transaction Tx_i describing the negotiation details with its own signature and sends it to the slice provider C_p. After confirming the details, C_p will add its own signature and broadcast Tx_i to all peers.

Shared ledger generation: For each broker node, it also plays the role as a miner to pack slice trading transactions into blocks, which will be appended to the shared ledger for permanent storage. With Proof-of-Majority(PoM) consensus protocol, all broker nodes will reach consistency regarding the transactions contained in each block locally, and Tx_i will further be packed into blocks and appended the shared ledger for permanent storage. The details of PoM consensus protocol will be discussed in Section 4 and Section 5, respectively.

3.2 Data Model

Six type of data structures are involved in the network slice broker collaboration system: identity registering message, resource registering message, resource request message, slice trading transaction, the blocks in the blockchain, as well as the blockchain itself, i.e., the shared ledger maintained by each peer node.

An identity registering message $M - IR_i$ is formalized by (1):

$$M - IR_i =< Type = 0, Pk_i, P_i, \theta_i > \tag{1}$$

$Type = 0$ means the message refers to an identity registering notice. Pk_i is the public address representing C_i, which is used to verify C_i's identity. P_i is the identity proof information containing C_i's IP address, MAC address, port number, etc. θ_i is the signature of C_i to prove the message is sent by C_i. Please be noted that, since we adopt a consortium blockchain as the underlying overlay network, each broker is verified by MSP node. And the identity information should be known to each broker in advance as well. Therefore, the main goal of identity registering is to make C_i's Pk_i and P_i known to each peer node.

A resource registering message $M - RR_i$ is formalized by (2):

$$M - RR_i =< Type = 1, Pk_i, RR_i, \theta_i > \tag{2}$$

$Type = 1$ means the message refers to an resource registering notice. Pk_i proves the registering request is sent by C_i, $RR_i = \{< S_1, QoS_1 >, < S_2, QoS_2 >, ... < S_m, QoS_m >\}$ describes slice resources owned by C_i. And θ_i is the signature of C_i.

A resource request message $M - R_i$ is raised by a broker node C_i to imitate a slice trading with another peer broker. And a resource requesting message M_R can be described by (3):

$$M - R_i =< Type = 2, Pk_i, R_i, \theta_i > \tag{3}$$

$Type = 2$ identifies the message as a slice request, and $R_i =< S_i, QoS_i >$ describes the request regarding functional requirement and QoS requirement. Pk_i and θ_i is for identity verification similar to identity registering and resource registering cases.

A slice trading transaction Tx_i describes the detailed slice exchange records between two brokers C_p and C_c, after C_p successfully fulfill C_c's resource request. Formally, it could be formalized by (4):

$$Tx_i =< Pk_p, Pk_c, SLA, \theta_p, \theta_c > \tag{4}$$

Pk_p and Pk_c are public addresses of C_p and C_c, SLA records negotiation details of the slice trading transaction. θ_p and θ_c are signatures of C_p and C_c, respectively.

A transaction block B_i contains a set of k transaction records, i.e., $TS_i = \{Tx_1, Tx_2, ..., Tx_k\}$, a Merkle tree MT_i calculated with the k

transactions, as well as the hash value of the last block in current working ledger. Specially, a transaction block could be described as a fourth-tuple:

$$B_i =< hash(B_i), preHash(L.lastBlock), MT_i, TS_i > \quad (5)$$

The shared ledger noted as L includes all slice trading transactions. Those slice trading transactions are packed into linked blocks, and maintained by each member in the system. In fact, the shared ledger records the shared state of the broker collaboration system, and the PoM consensus protocol is to ensure all peers maintain an identical ledger containing the same sequence of slice trading transactions.

More specifically, two ledgers L, L' are the same if they represent the same ordered history of transactions starting from the unique genesis ledger. Each ledger also has sequence number $seq(L)$ that is one greater than its parent ledger's sequence number. The genesis ledger has $seq(L) = 1$. A ledger L is created by applying a sequence of transactions $Tx = [Tx_1, Tx_2, ...]$ to its parent $parent(L)$ according to the protocol rules. The Proof-of-Majority(PoM) protocol is designed to guarantee each peer broker node adds same set of transactions to its parent ledger, so as to maintain an identical shared ledger among each other.

4 THE BASIC PROOF-OF-MAJORITY(POM) CONSENSUS PROTOCOL

Regarding the basic Proof-of-Majority consensus protocol, we have following assumptions:

(1) $|C| = n$, i.e., there are n slice brokers in the permissioned blockchain network, and the size is fixed to n. All nodes should get registered and the identity information must be fetched to each peer broker before it raises a slice trading request;

(2) $n > 2f + 1$, where f is the maximum failure broker nodes in the system during consensus process. Please be noted that we assume there is no Byzantine node in the network, which is guaranteed by the identity registering process. A broker node may be disconnected or out of response, but it will not send wrong response or requests among the blockchain network;

(3) Communication is conducted asynchronously via delivering messages with a timeout bound t;

(4) A transaction is committed as long as it is confirmed by majority peer brokers, which will be further packed into blocks and recorded in ledger L. Moreover, different from popular PoW or PoS protocol, we avoid block forking in PoM, where all transactions are sorted in identical sequence on each broker. Therefore, the blocks will be generated by each peer broker locally after the transactions are confirmed by majority peers.

The PoM consensus process consists of 5 phases, as illustrated in Fig. 2.

Phase 1 - Identity registering Since we adopt a consortium blockchain as system overlay network, all peer brokers should know each other in advance. As mentioned in previous section, each broker node must get itself registered on the MSP node. And knowing each peer broker is achieved via fetching peer nodes' identity information from the MSP node by each peer broker. Identity registering is implemented through sending an identity registering message $M - IR$ to MSP node. The MSP node will verify and collect each broker's identity information, which will be further fetched by each peer node C_i and recorded in C_i's local identity information database $IR - DB_i$. For MSP node, when it receives $M - IR_i$ from C_i, it will match C_i's information against its white list firstly and record C_i's identity information, if C_i is a valid peer node.

Phase 2 - Resource registering Similar to identity registering, each broker should get its slice resources registered to MSP node as well, which will be recorded on each peer node. Resource registering is also achieved by sending a resource registering message $M - RR$ to the MSP node. Such information will be fetched by each peer broker, so as to facilitate the validation process of slice trading transactions in Phase 4(i.e, transaction sorting), to make sure the slice provider involved in a transaction really has the corresponding slice resource.

Phase 3 - Leader election Apart from the MSP node, there are two roles in a consortium network: leader and followers. A leader broker C_l is responsible for validating each slice trading transaction Tx_i, nominating the sequence of each Tx_i, and initiating voting process among the network to determine the sequence of Tx_i; while a follower C_i mainly validates and votes for the sequence of each slice trading transaction. The leader election process could be implemented with various methods, among which the simplest one is Round-Robbin method. Another typical well-proven leader election method is the one adopted in RAFT, where the peer with longest log length is elected as leader node [6]. In our implementation, we adopt the simplest Round-Robbin leader election method to select the leader broker, where each broker has equal probability to be the leader node.

Phase 4 - Transactions sorting Once leader C_l receives a slice trading transaction Tx_i, it will firstly validate both identity of message sender C_i and the transaction details contained in Tx_i, then determine the sequence of Tx_i by initiating a vote process among each node in C. Once $VoteRequst(Tx_i, Seq(Tx_i))$ receives votes from majority nodes, C_l will mark Tx_i as committed with determined sequence number $Seq(Tx_i)$. The transaction validation algorithm is listed in Algorithm 1.

Algorithm 1 Transaction validation algorithm

```
Input: $Tx_i$
Output: True or False
  if $\theta_p$ or $\theta_c$ in $Tx_i$ is invalid then
    return False
  end if
  if $C_p$ does not own the resource contained in $Tx_i.SLA$ according
  to $C_l$'s $RR - DB$, and according to the shared ledger $L$, there is
  no transaction record proving $C_p$ have sufficient resource within
  the time period before $Tx_i$ happens then
    return False
  end if
  return True
```

Once Tx_i passes transaction validation, C_l will firstly append it to the end of its local transaction pool $Tx - pool_l$. Then C_l will initiate a $VoteRequest(Tx_i, Seq(Tx_i))$ message and broadcast it over the network. Once C_l receives approval message from majority members within timeout t, it will notify all members about Tx_i's

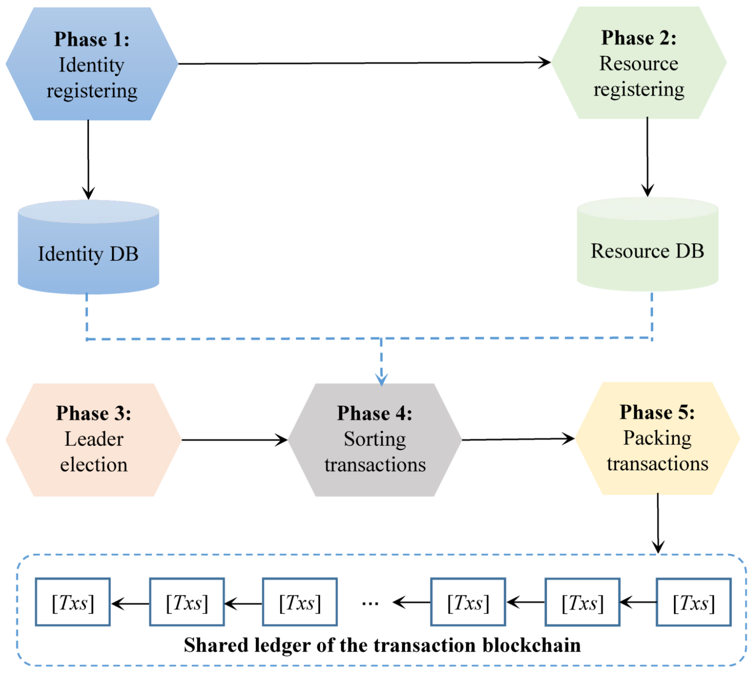

Phase 1: Identity registering
Phase 2: Resource registering
Identity DB
Resource DB
Phase 3: Leader election
Phase 4: Sorting transactions
Phase 5: Packing transactions
[Txs]
[Txs]
[Txs]
...
[Txs]
[Txs]
[Txs]
Shared ledger of the transaction blockchain

Figure 2: The PoM consensus process

sequence $Seq(Tx_i)$ and mark Tx_i as committed, which means Tx_i is ready to be packed into blocks. Otherwise, $Seq(Tx_i)$ will be marked as None, meaning $Seq(Tx_i)$ should be determined in another voting round. The whole process could be summarized by Algorithm 2.

Algorithm 2 Transaction sorting algorithm

```
Input: $Tx_i, C_l.Tx-Pool$
Output: $(Tx_i, Seq(Tx_i))$
1: $Seq(Tx_i) \Leftarrow C_l.Tx-pool.length+1$
2: $C_l.Tx-pool[Seq[Tx_i]] \Leftarrow Tx_i$
3: $Vote(Tx_i, Seq(Tx_i)) \Leftarrow 1$
4: broadcast "$VoteRequest(Tx_i, Seq(Tx_i))$" message to all peers in $C$
5: while $Vote(Tx_i, Seq(Tx_i)) < f+1$ and $timeout < t$ do
6:     if $C_l$ receives one $VoteRequest(Tx_i, Seq(Tx_i))\_OK$ message then
7:         $Vote(Tx_i, Seq(Tx_i))++$
8:     end if
9: end while
10: if $timeout >= t$ then
11:     $C_l.Tx-pool[Seq[Tx_i]] \Leftarrow None$
12:     return $(Tx_i, None)$
13: end if
14: broadcast "$C_l.Tx-pool[Seq[Tx_i]] = Tx_i committed$" message to all peers in $C$
15: $C_l.Tx-pool.length+=1$
16: mark $Tx_i$ as committed
17: return $(Tx_i, Seq(Tx_i))$
```

Phase 5 - Packing sorted transaction to generate blocks As long as the transaction pool of each broker nodeC_i is not empty, C_i could start to generate blocks and append it to the shared ledger L. Typically, a block has a determined size with several transactions. The intuitive method to generate blocks is packing transactions from the beginning of $Tx-pool$, which is described in Algorithm 3. In practice, the block size k can be set flexibly.

Algorithm 3 Block generation algorithm for the i-th block in shared ledger L

```
Input: $C_l.Tx-pool$
Output: $B_i$, A block of packed $k$ transactions
1: initiate an empty block data structure $B_i$
2: pick first $k$ transactions in $C_l.Tx-pool$, i.e., $TS_i = \{Tx_{s1}, Tx_{s2}, ..., Tx_{sk}\}$
3: $MT \Leftarrow merkely\ tree\ with\ the\ k\ transactions$
4: $B_i.MT_i \Leftarrow MT$
5: $B_i.TS_i \Leftarrow TS_i$
6: $B_i.hash \Leftarrow Hash(TS_i)$
7: $B_i.preHash \Leftarrow Hash(L.lastBlock)$
8: remove the picked $k$ transactions from $C_l.Tx-pool$
9: return $B_i$
```

5 OPTIMIZATION FOR BASIC PROOF-OF-MAJORITY PROTOCOL

To improve transaction processing efficiency and block generation rate, we propose an optimization for the basic Proof-of-Majority protocol in this section. Briefly, we allow transactions being handled concurrently in both Phase 4 and Phase 5. And regarding concurrency, the dependency among slice trading transactions should be taken into consideration.

Definition 5.1. Transaction dependency: A slice trading transaction Tx_i has a dependency on another slice trading transaction Tx_j, if the slice resource provider of Tx_i(i.e., $Tx_i.C_p$) is the consumer of Tx_j(i.e., $Tx_j.C_c$). And Tx_i's C_p does not own the slice resource referred in $Tx_i.SLA$.

In practice, the transaction dependency happens when Tx_i's C_p does not own the slice resource S_i referred in $Tx_i.SLA$ according to the $RR{-}DB$ on each peer node. But there exists a packed transaction Tx_j in L, where Tx_i's C_p is the consumer of S_i provided by Tx_j's C_p. At the same time, Tx_i's C_p still have valid access to S_i when Tx_i happens.

(1) Optimization for transaction sorting

Instead of handling each slice transactions one by one in serial manner, the leader node C_l could distribute m transactions to all peers in C for each transaction sorting round. More specifically, C_l will set the local sequence for each transaction according to the transaction dependency among the the m transactions. Then it will distribute the m transactions to each peer to get majority approvals, so as to determine the final sequence of each transaction. Different from Algorithm 2 to sort each transaction before sorting the slice trading requests, a dependency resolving algorithm should be applied to set local sequence of m transactions. We choose topological sorting as dependency resolving algorithm as depicted in Algorithm 4. With Algorithm 4, transactions in $Tx - Set$ will be sorted according to the dependency constraint.

Algorithm 4 Dependency resolving algorithm for slice trading transactions

```
Input: $Tx - Set = \{Tx_1, Tx_2, ...., Tx_m\}$
Output: Sorted slice trading transaction set $Tx - Set = [Tx_{s1}, Tx_{s2}, ...., Tx_{sm}]$
1: initiate an empty directed acyclic graph $Tx - Graph$
2: each transaction in $Tx - Set$ is represented as a node in $Tx - Graph$
3: for each node pair $(Tx_i, Tx_j)$ in $Tx - Set$ do
4:     if $Tx_j$ has dependency on $Tx_i$ then
5:         add a directed edge start from $Tx_i$ which points to $Tx_j$
6:     end if
7: end for
8: initiate $Tx - Vector$ as an empty container
9: while the node collection of $Tx - Graph$ is not empty do
10:    for each node $Tx_i$ in $Tx - Graph$ do
11:        if input degree of $Tx_i$ is 0 then
12:            add $Tx_i$ to $Tx - Vector$
13:            remove $Tx_i$ and all edges starting from $Tx_i$
14:        end if
15:    end for
16: end while
17: return $Tx - Vector$
```

(2) Optimization for block generation Since we support sorting slice trading transactions concurrently, transactions may get committed in an out-of-order manner due to issues like network delay, or temporary failure of single broker nodes. We define such scenario as transaction commit hole.

Definition 5.2. Transaction commit hole: For a broker node(e.g., C_i), in its local transaction pool $Tx - pool$, a transaction commit hole happens when a transaction Tx_i gets committed before Tx_j, even $Seq(Tx_i) \geq Seq(Tx_j)$. In this case, Tx_j looks like a hole for the block generation algorithm, since Tx_i is ready for block generation, while Tx_j is hold for the block generation process.

With this observation of transaction commit hole, we can not simply apply Algorithm 3 to generate blocks in serial manner. Instead, we improve Algorithm 3 with following principle: After a transaction Tx_i get committed, it can be packed to blocks as long as it has no dependency on other transactions, or all its dependency transactions are committed. For example, in the case mentioned in Definition 5.2, if Tx_i get committed before Tx_j, and Tx_i has no dependency on Tx_j, Tx_i is ready for block generation after it get committed. Otherwise, if Tx_i has dependency on Tx_j, it is hold for block generation process before Tx_j get committed.

More specifically, we introduce a dependency window associated with each slice trading transaction in Phase 4 to determine the sequence of transactions.

Definition 5.3. Dependency window: For a slice trading transaction Tx_i, its dependency window $Tx_i.Dep$ is a data structure containing m indexes of all dependency transactions of Tx_i in the $Tx - pool$, And Tx_i can only be packed into blocks after each Tx_k in $Tx_i.Dep$ get committed.

Dependency window helps a broker to judge whether a slice trading transaction can be packed after it get committed. When the leader node C_l initiates a voting request to determine the sequence of Tx_i, it attaches the dependency window value $Tx_i.Dep$ to the vote request, i.e., $VoteRequest(Tx_i, Seq(Tx_i), Tx_i.Dep)$. With $Tx_i.Dep$, each follower broker C_j can extract the indexes which Tx_i relies on. And after Tx_i get committed, it can be packed into blocks as long as $Tx_i.Dep$ is empty or all transactions associated with $Tx_i.Dep$ are committed already. Otherwise, Tx_i should be hold for packing until all its dependency transactions get committed.

Please be noted that in our optimized PoM protocol, committed transactions can be packed into blocks in an out-of-order manner. It is possible that transactions with larger index in $Tx - pool$ are packed before transactions with smaller sequence index. However, according to our block structure, each block maintains a hash pointer to its previous block to guarantee the tamper-resist feature of the shared ledger. Therefore, this will lead to empty hash pointer value for some early packed blocks. To address this issue, we modify the block data structure by adding a block index value for each block and keep the hash pointer to NULL for each early packed block. And a process is initiated to scan L every t seconds to flesh out the hash pointer value for each block based on its previous block. Algorithm 5 depicts the optimized block generation process.

Algorithm 5 Optimized block generation algorithm

```
Input: $C_l.Tx - pool$
Output: shared ledger $L$ produced from $C_i.Tx - pool$
 1: initiate an empty block data structure $B_i$
 2: $blockIndex, packIndex, packFlag \Leftarrow 0$
 3: while $C_i.Tx - pool$ is not empty do
 4:     if $packIndex \geq C_i.Tx - pool.length$ then
 5:         $packIndex \Leftarrow packFlag, packFlag \Leftarrow 0$
 6:         $blockIndex \Leftarrow blockFlag, blockFlag \Leftarrow 0$
 7:     end if
 8:     pick first $k$ transactions in $C_i.Tx - pool$ starting from
        $packIndex$, i.e., $TS_i = Tx_1, Tx_2, ..., Tx_k$
 9:     if each $Tx_i$ in $TS_i$ has no dependency transaction or all its
        dependency are resolved then
10:         call Algorithm 3 to generate the $blockIndex$+1 block
11:         $B_i.blockIndex \Leftarrow blockIndex + 1$
12:         if the $L.blockIndex - th$ block is not packed then
13:             $B_i.preHash \Leftarrow NULL$
14:             $B_i.Hash \Leftarrow NULL$
15:         else
16:             $B_i.preHash \Leftarrow B_i - 1.Hash$
17:             $B_i.Hash \Leftarrow Hash(B_i.preHash, B_i.MT, B_i.TS)$
18:         end if
19:         remove $Tx_i$ in $TS_i$ from $C_i.Txpool$
20:         append $B_i$ into $L$ as $blockIndex$-th block
21:     else
22:         $packFlag \Leftarrow 0?packIndex : packFlag$
23:         $blockFlag \Leftarrow 0?blockIndex : blockFlag$
24:         $packIndex+ = k$
25:     end if
26:     $blockIndex+ = 1$
27: end while
```

6 ANALYSIS

In this section, we analyse the PoM consensus protocol based on the properties of *security* and *performance*, respectively.

6.1 Security

The PoM consensus protocol guarantees the security of a broker collaboration system by forcing each broker to get itself registered by Phase 1. With identity registering, each broker nodes knows each other's identity such as public address, MAC address within the same consortium network in advance. And the public address is used to encrypt communication messages over the network. With identity registering preparation, we scope out Byzantine nodes for broker collaboration network. Therefore, messages sent from external entities will be ignored by the broker collaboration. Furthermore, the miners in the collaboration network are the broker nodes themselves, and the blocks are mined by broker nodes as well, which guarantee the authority and security of the shared ledger. Moreover, the PoM consensus protocol is a majority based protocol, where the sequence of each slice trading transaction is determined with approvals from majority members in the network. Therefore, as long as majority $f + 1$ members are alive among the n nodes, broker nodes can always reach consensus regarding the sequence of each Tx_i.

However, since PoM consensus protocol is based on majority voting, the size of the broker collaboration network should be fixed during the consensus process to sort slice trading transactions. In real applications, complex scenarios include: (1) a broker node may failed permanently and should be removed from the system; (2) new brokers join the collaboration system to enhance application's availability. As a result, the majority value f should be updated accordingly. However, in our discussion, we only focus on the normal working process of the PoM consensus protocol. How to enhance the scalability of PoM consensus protocol to adapt to membership changes will be studied in future work.

6.2 Performance

The essence of the PoM consensus protocol is to determine the sequence of each slice trading transaction before generating blocks to be appended to the blockchain ledger. Thus, intuitively, the performance of PoM is better than popular consensus protocols such as PoW and PoS. In PoW and PoS, transactions being packed into blocks could be possibly invalid and be put back to transaction pool, if the miner loses the recording right of a given block. Moreover, we have no block broadcast message cost in PoM consensus protocol, since all transactions are kept in same sequence on each local broker, and block packing is run by each local broker node as well.

From message cost perspective, assume that the size of a broker collaboration network is n and $n = 2 \times f + 1$, where f is the maximum number of failure brokers during consensus process.

Phase 1: Identity registering requires a broker register its identity information to the MSP node, by sending a $M - IR$ message to the MSP node. Thus the message produced in this phase is: n.

Phase 2: Resource registering is similar to the identity registering process, which will produce n messages as well.

Phase 3: Since the leader is elected in a Round-Robbin manner, the message cost will be $O(n)$.

Phase 4: For each slice trading transaction Tx_i, the leader node will determine its sequence and forward a voting request to all followers, and after $Seq(Tx_i)$ approved by majority peers, another commit message will be forward to each follower as well. As a result, the number of exchanged message will be: $3 \times (n - 1)$.

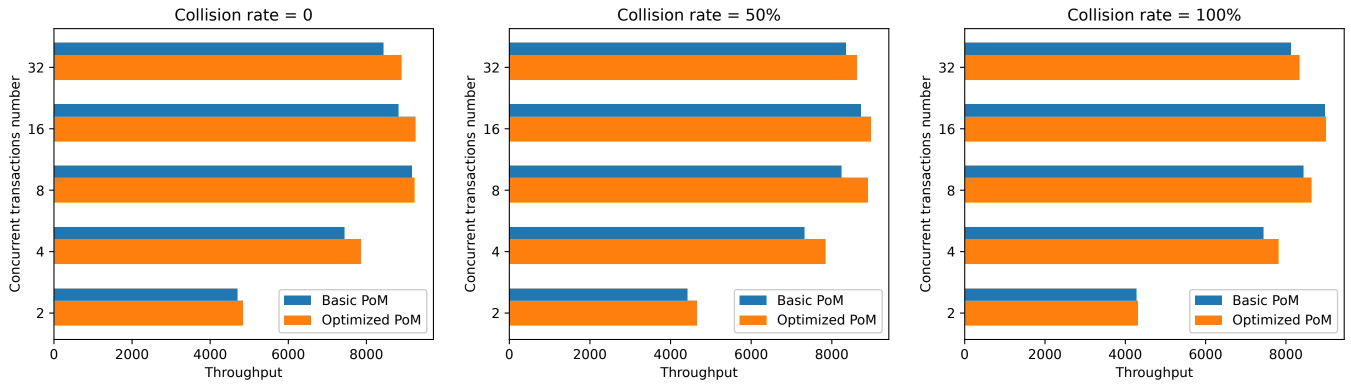
Phase 5: According to our design, there is no message cost in Phase 5, since all transactions are sorted already in Phase 4, and each broker miner packs its blocks locally.

In total, the number of messages exchange is $n+n+n+3\times(n-1)$. This shows the complexity of the process of message exchange is $O(n)$.

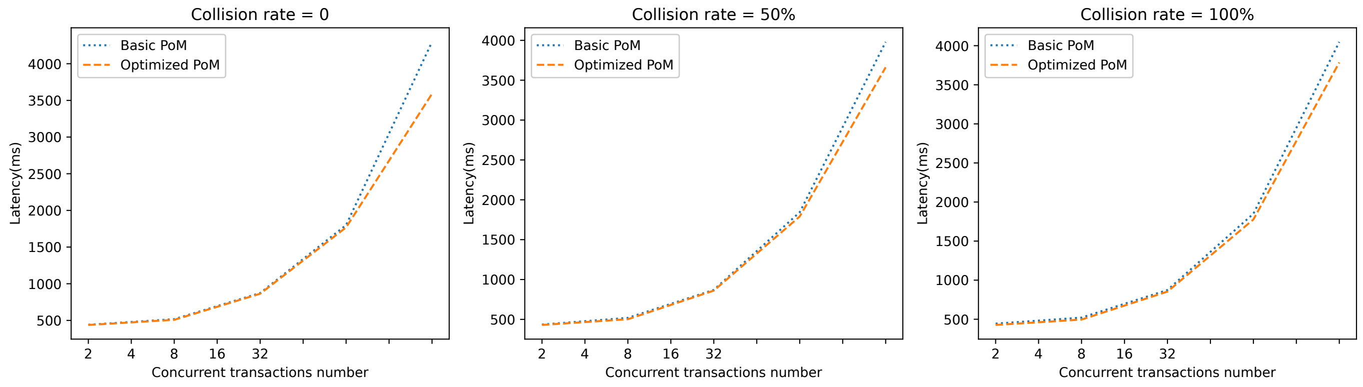
7 EXPERIMENTAL EVALUATION

To evaluate the performance of our PoM consensus protocol, we simulate the consensus process of PoM on a Mac OS system with an I7 Intel Core CPU and 128GB RAM. Two perspectives (i.e., system throughput and latency of block generation rate) are compared between the basic PoM protocol and the optimized PoM consensus protocol.

Based on the discussion in Section 4 and 5, there are two factors determining the performance difference between basic PoM protocol and the optimized PoM: the collision rate of slice trading



(a) Throughput comparison w.r.t collision rate



(b) Latency comparison w.r.t collision rate

Figure 3: Performance comparison w.r.t. collision rate

transactions, and real network environment, respectively. The collision rate means the percentage of slice trading transactions having dependency against each other within the whole transaction set. Since basic PoM protocol handles slice transactions one by one in sequence, the collision among slice trading transactions should not impact the performance of basic PoM protocol in theory. While the optimized PoM protocol handles transaction in an out-of-order manner, by taking into consideration of the dependency among each transaction. As a result, it is expected to observe better performance regarding smaller collision rate, since smaller collision rate value means higher concurrency for the optimized PoM protocol. On the other hand, the network delay introduced by network traffic will cause worse system throughput and latency in theory. Similarly, the optimized PoM protocol is also expected to work better than the basic PoM protocol.

7.1 Experimental setting

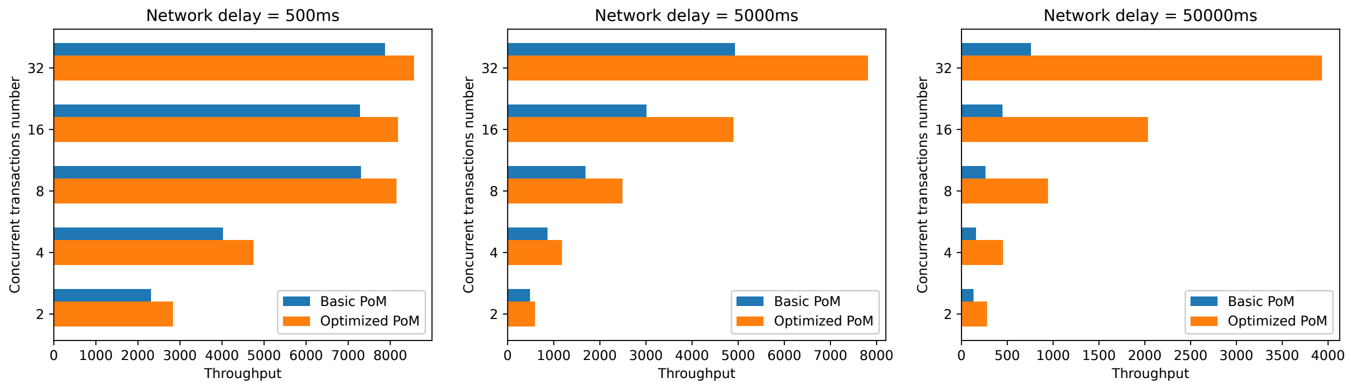
To reflect a real permissioned blockchain network as much as possible, we simulate two set of experiments with different collision rate value, and different network settings with different network delay values, respectively. Also, without loss of generalization, the size of our consortium network is fixed to 3, meaning the sequence of each transaction is determined once it gets approvals with at least 2 members in the network. Moreover, the number of concurrent transactions broadcasting within the network is in range of 2, 4, 8, 16 and 32, respectively. Without loss of generalization, we set the block size to 1 (i.e., each block contains only 1 transaction). We run the simulated experiments, and record the system throughput and latency of block generation rate for following analysis.

7.2 Experiment Results

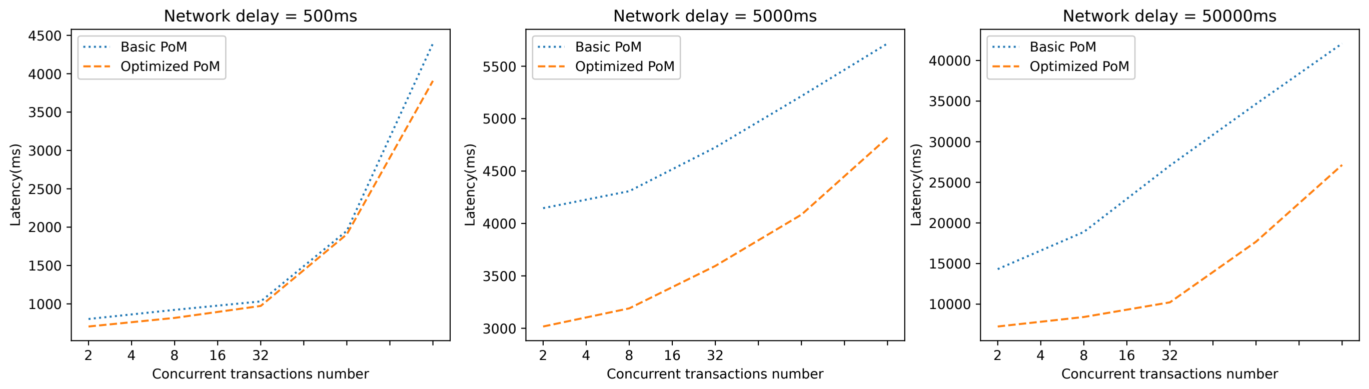
A. Performance comparison w.r.t collision rate

In this experiment, we focus on analysing the impact of collision rate among slice trading transactions on both protocols. Also, we simulate an ideal network environment by setting the network delay value to 0. By doing this, we can obtain the standard latency value to process a transaction to generate a block. Therefore, we can simulate the real network delay based on this value in following experiment *B*.

Fig. 3 demonstrates the comparison result between basic PoM protocol and the optimized PoM protocol with respective to different collision rate. As shown in Fig. 3, in overall, the optimized PoM protocol works better than the basic PoM protocol. However, the difference between the two protocols is quite small. The reason for this is analysed as follows: The time cost to process each transaction consists of two major parts: time cost to commit a transaction and time cost to pack the transaction. Since there is no network delay over the simulated consortium network, the waiting time for each



(a) Throughput comparison w.r.t network delay



(b) Latency comparison w.r.t network delay

Figure 4: Performance comparison w.r.t. network delay

transaction to be packed is quite small, compared with the time cost to commit the transaction. And the time cost difference between two protocols to commit a transaction is mainly determined by the number of concurrent transactions. As a result, the difference between the two protocols becomes distinct with the increase of the number of concurrent transactions.

Moreover, as depicted in Fig. 3(a), the throughput of both protocols get increased with the increase of the number of concurrent transactions at beginning. But when the number of concurrent transactions reach 8, the throughput becomes stable regardless the increase of the number of concurrent transactions. As a result, the maximum throughput in our simulated 3-brokers consortium network is 10000tps. Also, the smaller collision rate is, the better optimized PoM protocol works compared with the basic PoM protocol. The reason for this is the optimized PoM protocol handles transactions in out-of-sequence manner. The smaller collision rate means higher concurrency to process transactions for the optimized PoM Protocol. Similarly, in Fig. 3(b), the smaller collision rate, the bigger difference of system latency between the two protocols as well.

At the same time, with Fig. 3(b), we can obtain the value of standard latency to process each transaction is about 500ms in the simulated permissioned blockchain network. As a result, we will refer this value to determine the value of network delay settings in experiment *B*.

B. Performance comparison w.r.t network delay

In this experiment, we focus on analysing the impact of network delay on both protocols. According to experiment *A*, the standard latency to process each transaction is around 500ms with our simulated consortium network. In this setting, we simulate the real network environment by setting the network delay to 500ms, 5000ms and 50000ms on each broker node, and observe the performance of each protocols, respectively.

As shown in Fig. 4, in overall, the optimized PoM protocol works much better than the basic PoM protocol. And the difference between the two protocols becomes quite obvious, especially with the increase of network delay. Similar to the case in experiment *A*, the time cost to process each transaction is determined by time cost to commit a transaction and time cost to pack a transaction, respectively. And network delay introduces extra time cost to hold each transaction to get packed. As a result, the out-of-manner of handling transactions in optimized PoM will reduce the impact of network delay. And the bigger network delay, the much better performance of the optimized PoM protocol, compared with the basic PoM protocol.

Similar to experiment *A*, with the increase of concurrent transactions, the throughput get increased, and it will be fixed around

10000tps finally, as depicted in Fig. 4(a), which is determined by the simulated network itself. Also, as just mentioned, bigger network delay has bigger impact on the basic PoM protocol, compared with the optimized PoM protocol, regarding system latency as well, as shown in Fig. 4(b).

8 CONCLUSION

This paper proposes a blockchain-enabled network slice broker collaboration infrastructure, where multiple brokers can negotiate with each other to achieve slice resource sharing among each other. And we design a Proof-of-Majority (PoM) consensus protocol to enhance the accountability of collaboration records. Concretely, slice transactions are sorted on each broker node, and further packed into blocks to be appended to a shared ledger, which is maintained by each member in the consortium network. Different from popular consensus protocol such as Proof-of-Work (PoW) or Proof-of-Stake (PoS), our PoM consensus protocol determines a same sequence of slice trading transaction for each miner firstly, before generating each block, to guarantee block generation efficiency, as well as the robustness of the consensus process. The limitation of our work is the scalability issue when the members of the broker collaboration network changes, the consensus protocol should adjust accordingly, since majority members are changed accordingly. Improvements for such scenario are under consideration of our future work.

ACKNOWLEDGMENTS

This paper is partially supported by the National Science Foundation of China under Grant No. 61872219, No. 61672276, No. 61702277, the Natural Science Foundation of Shandong Province (ZR2019MF001), the Open Project funded by State Key Laboratory of Novel Software Technology (Nanjing University, KFKT2019B19).

REFERENCES

[1] Third Generation Partnership Project (3GPP). [n.d.]. System Architecture for the 5G System. document3GPPTS23.501,v.16.0.2

[2] B. Nour A. Ksentini, et al. 2019. A Blockchain-based Network Slice Broker for 5G services. *IEEE Networking Letters* 1 (2019), 99—-102.

[3] NGMN Alliance. [n.d.]. Description of Network Slicing Concept. Retrieved May 27, 2017 from ngmn.org/fileadmin/user_upload/160113_Network_Slicing_v1_0.pdf

[4] J. You B. Ye., et al. 2018. A Proof-of-Trust Consensus Protocol for Enhancing Accountability in Crowdsouring Services. *IEEE Transactions on Services Computing* 12 (2018), 429—-445.

[5] J. Yu D. Kozhaya, et al. 2019. RepuCoin: Your Reputation Is Your Power. *IEEE Trans. Comput.* 68, 8 (aug 2019), 1225-1237.

[6] S. Deetman. 2019. Bitcoin Could Consume as Much Electricity as Denmark by 2020. http://motherboard.vice.com/read/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020

[7] J. Salvat et al. 2018. Overbooking Network Slices through Yield-driven end-to-end Orchestration. 353—-365.

[8] T. Rasheed et al. 2015. Business Models for Cooperation. *Energy Efficient Smart Phones for 5G Networks: Signals and Communication Technology* Springer International Publishing Switzerland (2015).

[9] A. Boubendir F. Guillemin, et al. 2018. Federation of Cross-Domain Edge Resources: A Brokering Architecture for Network Slicing. 415–423.

[10] R. Wen G. Feng, et al. 2018. On Robustness of Network Slicing for Next-Generation Mobile Networks. *IEEE Transactions on Communications* 67, 1 (2018), 430–444.

[11] et al. H. Zhang, N. Liu. 2017. Network slicing based 5G and Future Mobile Networks: Mobility, Resource Management, and Challenges. *IEEE Communication Magazine* 55, 8 (2017), 138–145.

[12] R. E. Hattachi and J. Erfanian. 2015. 5G White Paper. *NGMN Alliance* (2015).

[13] Q. Wang J. A. Calero, et al. 2019. Enable Advanced QoS-Aware Network Slicing in 5G Networks for Slice-Based Media Use Cases. *IEEE Transactions on Broadcasting* 65, 2 (2019), 444–453.

[14] X. Costa-Perez K. Samdanis and V. Sciancalepore. 2016. From Network Sharing to Multi-tenancy: The 5G Network Slice Broker. *IEEE Communications Magazine* 54, 7 (2016), 32–39.

[15] D. Kraft. 2016. Difficulty Control for Blockchain-based Consensus Systems. *Peer-to-Peer Networking and Applications* 9 (2016), 397–413.

[16] D. A. Chekired M. A. Togou, et al. 2019. 5G-Slicing-Enabled Scalable SDN Core Network: Toward an Ultra-Low Latency of Autonomous Driving Service. *IEEE Journal on Selected Areas in Communications* 37, 8 (2019), 1769–1782.

[17] N. Mbarek M. Hamze and O. Togni. 2016. Broker and Federation based Cloud Networking Architecture for Iaas and Naas QoS Guarantee. 705—-710.

[18] A. Dorri M. Steger, et al. 2017. BlockChain: A Distributed Solution to Automotive Security and Privacy. *IEEE Communication Magazine* 55, 12 (2017), 119–125.

[19] T. Salman M. Zolanvari, et al. 2019. Security Services using Blockchains: A State of the Art Survey. *IEEE Communications Surveys Tutorials* 21, 1 (2019), 858–880.

[20] S. Malik and F. Huet. 2011. Virtual Cloud: Rent Out the Rented Resources. 536—-541.

[21] S. Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[22] D. Ongaro and J. K. Ousterhout. 2014. In Search of An Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference*. 305–319.

[23] A. Ksentini P. A. Frangoudis, et al. 2018. Providing Low Latency Guarantees for Slicing-ready 5G Systems via Two-level MAC Scheduling. *IEEE Network* 32, 6 (2018), 116–123.

[24] X. Xu Q. Liu, et al. 2019. A Blockchain-Powered Crowdsourcing Method with Privacy Preservation in Mobile Environment. *IEEE Transactions on Computational Social Systems* 6 (2019), 1407–1419.

[25] E. Coronado S. N. Khan, et al. 2019. 5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks. *IEEE Transactions on Network and Service Management* 16, 2 (2019), 715–728.

[26] J. Backman S. Yrjola, et al. 2017. Blockchain Network Slice Broker in 5G: Slice Leasing in Factory of the Future Use Case. 1—-8.

[27] Z. Kotulski T. Nowak, et al. 2017. On end-to-end Approach for Slice Isolation in 5G Networks. Fundamental Challenges. 783—-792.

[28] I. Afolabi T. Taleb, et al. 2018. Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions. *IEEE Communications Surveys Tutorials* 20, 3 (2018), 2429–2453.

[29] S. Underwood. 2016. Blockchain Beyond Bitcoin. *Commun. ACM* 59 (2016), 15–17.

[30] X. Costa-Perez V. Sciancalepore and A. Banchs. 2019. RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker. *IEEE/ACM Transactions on Networking* 27 (2019), 1543–1557.

[31] X. Xu X. Zhang, et al. 2019. BeCome: Blockchain-Enabled Computation Offloading for IoT in Mobile Edge Computing. *IEEE Transactions on Industrial Informatics* (2019). https://doi.org/10.1109/TII.2019.2936869

[32] X. Xu Y. Chen, et al. 2019. Blockchain-based Cloudlet Management for Multimedia Workflow in Mobile Cloud Computing. *Multimedia Tools and Applications* (2019). https://doi.org/10.1007/s11042-019-07900-x

[33] X. Xu Y. Chen, et al. 2019. A Blockchain-based Computation Offloading Method for Edge Computing in 5G Networks. *Software: Practice and Experience* (2019). https://doi.org/DOI:10.1002/spe.2749