Multi-Party Payment Channel Network Based on Smart Contract

Yanjiao Chen, *Member, IEEE*, Xuxian Li, Jian Zhang, and Hongliang Bi, *Member, IEEE*

***Abstract*—Blockchain-based cryptocurrencies are severely limited in transaction throughput and latency. A promising solution to this issue is a payment channel, which allows trust-free payments between two peers without exhausting the resources of the blockchain. A linked payment channel network (PCN) enables payments between two peers through a series of intermediate nodes that forward and charge for the payments. However, most of existing proposals only use the shortest path as the path of the transaction, which causes the frequently reused channels to be exhausted quickly. In addition, most of existing PCNs are almost only designed for payments between two parties, which leads to limited application scenarios. When multiple payments use the same intermediate channel, the two-party PCNs cannot achieve simultaneous payments. In this paper, we propose a multi-party payment channel (MPC) network, a payment channel proposal that supports multiple payments using the same intermediate channel simultaneously, thereby greatly expanding the application scenarios of payment channels. In addition, our channel selection and transaction conversion strategies can also increase the success rate of transactions. We implement MPC network in the simulated blockchain network and lightning network based on Truffle, and a large number of experiments verify the effectiveness of our solution.**

***Index Terms*—Blockchain, smart contract, multi-party, payment channel.**

I. Introduction

FOR THE past several years, with the development of blockchain and smart contract technologies, cryptocurrencies like Bitcoin and Ethereum have already been widely used for secure transactions [1], [2]. However, it still faces the problems of low throughput and high latency. For example, only 7 tps (transactions per second) can be settled and the time for one transaction confirmation is about 10 minutes in Bitcoin, which cannot satisfy people's daily needs [3], [4]. Many methods have been proposed to solve these problems in recent years, such as side chain, payment channels and sharding.

Among of them, payment channel has been considered as the most promising technology [5]. The transactions mainly include four stages: creating, updating on the chain, updating off the chain and closing payment channel. Except for the updating off the chain finished in the payment channel network, other three stages need to interact with the blockchain through smart contracts. Fortunately, many interaction processes can be carried out off the chain, which reduces transaction delay and improves blockchain transaction throughput. Moreover, numerous payment channels can form a payment channel network (PCN), so a peer can pay other users with a certain fee through some intermediate peers. There are many researches dedicated to payment network framework and routing algorithms in existing works. Lightning Network is the most popular project, which uses PCN to realize instant Bitcoin payments. The Lightning Network [6] is a well-known PCN project for Bitcoin using Revocable Sequence Maturity Contract (RSMC) and Hashed TimeLock Contract (HTLC) to complete off-chain payments while ensuring security. For blockchains that support smart contracts like Ethereum [7], some researchers have designed more efficient payment channels like Perun [8] and Counterfactual [9].

However, the current researches about payment channel networks still face some challenges. Firstly, most existing works only discuss two-party transaction and ignore the need for multi-party applications. There are some scenarios that multiple customers need to pay for the same supplier simultaneously [10], [11]. It will be difficult for the two-party payment channel (TPC) system to adapt to scenarios such as multi-party auctions. Secondly, many payment channel networks use the shortest path algorithm to calculate the route. The problems of channel reuse and imbalance may be occurred [12]. Finally, some works suggest establish a multi-party payment channel (MPC) by employing a payment hub [13]. It is a straightforward method as the hub needs to mortgage a large amount of money to establish payment channels with users, which is inefficient in terms of the success rate of transactions. As far as we know, a payment hub with 1000 channels and \$1000 per channel will require a \$1000000 deposit [14]. The atomicity of multi-party transactions is not guaranteed. In addition, the hub nodes will form a centralized structure, which is contrary to the basic idea of blockchain decentralization.

Manuscript received 26 October 2021; revised 1 February 2022; accepted 22 March 2022. Date of publication 30 March 2022; date of current version 31 January 2023. This work was supported in part by National Natural Science Foundation of China (Program No.61972296), in part by Fundamental Research Funds for the Central Universities (Program No.D5000210598), in part by Natural Science Basic Research Program of Shaanxi (Program No.2022JQ-618). The associate editor coordinating the review of this article and approving it for publication was H. Otrok. *(Corresponding author: Jian Zhang.)*

Yanjiao Chen is with the College of Electrical Engineering, Zhejiang University, Hangzhou 310027, Zhejiang, China (e-mail: chenyanjiao@zju.edu.cn).

Xuxian Li and Jian Zhang are with the School of Computer Science, Wuhan University, Wuhan 430072, Hubei, China (e-mail: jzhang@whu.edu.cn).

Hongliang Bi is with the National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology, School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China (e-mail: bihongliang@nwpu.edu.cn).

Digital Object Identifier 10.1109/TNSM.2022.3162592

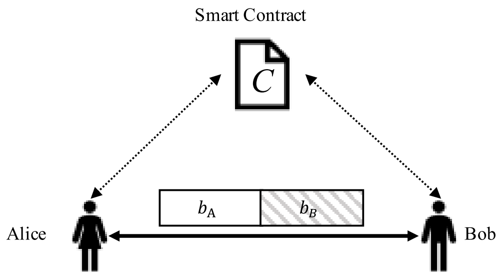


Fig. 1. Two-party payment channel.

In order to solve these challenges, in this paper, we propose a novel MPC network based on TPCs. In MPC network, a MPC allows any number of peers to join it, rather than limited to payments between two parties. MPC network converts multi-party payments into multiple directly connected two-party payments, which allows multiple payments using the same intermediate channel to be performed simultaneously. What's more, we employ a transaction optimization strategy to balance frequently used channels.

In summary, this paper makes the following contributions:

- We propose a novel MPC network by decomposing multi-party transactions and optimizing the transaction set, while executing payments through original TPCs.
- We have designed an improved transaction optimization strategy that increases the transaction success rate and reuse rate of payment channels and decreases the communication overhead.
- We implement MPC network in both simulated blockchain and Lightning networks based on Truffle, and the results of experiments prove the effectiveness of our solution.

II. Preliminaries

A. Two-Party Payment Channel

A TPC is the cornerstone of an MPC. The operations of creating and closing a TPC need to interact with the blockchain, which consumes more time than off-chain operations. The update operations are carried out off-chain, without waiting for a long time to confirm. We use the following example to explain the TPC transaction process in detail.

As shown in Fig. 1, at the creation step, Alice and Bob are willing to build a channel between them and agree to deposit b_A and b_B tokens to the smart contract respectively. Since the transaction is mutual, both Alice and Bob have their deposits. So Alice deposits b_A and Bob deposits b_B to the channel $t = [Alice : b_A, \ Bob : b_B]$, which denotes the original balance of the new channel. After creation, both Alice and Bob have the rights to request to propose a transaction and update the balance of the channel t. For instance, if Alice is going to pay Bob $p(p \leq b_A)$ tokens for a cup of coffee, then the balance of the channel will change to $t' = [Alice : b_A - p,$ $Bob : b_B + p]$, while the total amount of the channel $b_A + b_B$ keeps invariably. Then the updated information is sent to the blockchain. A transaction between two parties is authenticated by smart contract. The transaction version number is increased by one simultaneously. We can start the next round of the transaction and continue to repeat the above steps. We can start the next round of the transaction and continue to repeat the above steps. If the balance of any node is insufficient or 0, the transaction fails and will be rolled back to the previous updated state and submit it to the blockchain. Besides, if someone wants to defraud by providing an old version of the TPC, he/she will be punished by transferring all the balances to the other party. Finally, we need to be done is to register the latest state of the TPC to the smart contract for once before the TPC is closed. The party sends a close request to the smart contract when closing the channel, and the remained values will be withdrawn to Alice and Bob's accounts in the blockchain according to the latest balance distribution in the channel.

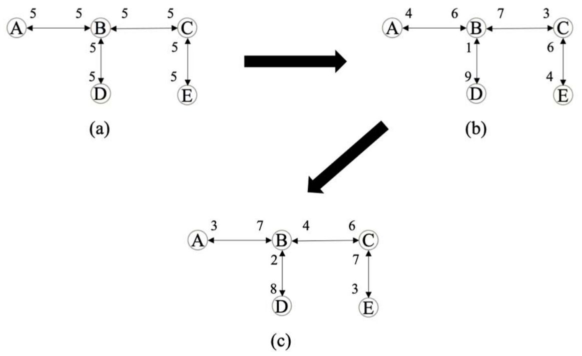


Fig. 2. Example of multi-party payment channel.

B. Multi-Party Payment Channel

The MPC is an extension of the above TPC. The TPCs constitute a connected topology and we can select some of the nodes and channels among them according to user's requirement to complete the construction of an MPC, which has the following features. Firstly, compared to TPC, the MPC supports multiple parties to take part in the same application. With multi-party channels, we can develop multi-player online games, auctions, crowdfunding, and other applications quickly and easily. Secondly, the MPC supports atomic multi-party transactions, which means the transaction between any two parties in channel either all happen, or none of them happen. Thirdly, there are no on-chain interactions when users conduct transactions on MPCs, which improves the efficiency of transactions and avoiding suffering from long latency.

We take a multiplayer online game as an example to illustrate the workflow of the MPC. As shown in Fig. 2, A, B, C, D, E are playing a multi-rounds online game. The rule is that there is one winner for each round, and other users should transfer one token to the winner. The prerequisite is that there are enough two-party channels between the five users to form a connected topology. Otherwise, they will not be able to conduct transactions.

First, the five parties reach a consensus and arbitrary party has the right to create a novel MPC by sending a request to the smart contract. The signatures can be collected by the party that initiates to establish MPC. Fig. 2(a) is the initial state of the MPC. There are four two-party channels, and each channel has five tokens at both ends. After a round of

TABLE I
SYMBOL DEFINITION

Symbol	Definition
id	the identifier of a TPC or an MPC
b_i^0	the initial balance deposited to the channel
b_i^{now}	the latest balance in the channel
$version$	the update version number of mpc_{id}
$expire$	the expire time of the channel
$busy$	an identifier to distinguish the status of the channel
$parties$	multiple blockchain peers that are willing to participate involved in the MPC
$tpcs$	the set of identifiers of the TPCs participating in the MPC contract
δ	the signatures of every peers in $parties$
$state$	a new state of the MPC after executing some multi-party transactions

the game, we suppose that party D wins the game and all other parties should transfer one token to D. A multi-party transaction is generated during the settlement phase, which can be denoted as $tx = \{(A, D, 1), (B, D, 1), (C, D, 1), (E, D, 1)\}$. Since two parties that are not directly connected need to transfer tokens through intermediate channels, some TPCs in the MPC will be used multiple times. To reduce the number of transactions and interactions between parties, we convert a multi-party transaction into transactions on multiple individual channels. In this example, we can convert tx to $tx_{new} = \{(A, B, 1), (B, D, 4), (C, B, 2), (E, C, 1)\}$. Once they reach a consensus of the update, they can register the latest state of the MPC to the smart contract. After the settlement phase of the first round, the new state of the MPC is depicted in Fig. 2(b). Similarly, the second round of the MPC will be updated to Fig. 2(c) when C wins the second round. It should be noted that the entire update process is atomic. For example, if the balance of node B is insufficient, node B will withdraw from the transaction process. The three payment channels associated with node B will also be closed. The multi-party online game will be over. When all users agree that the game is over, one of the parties in MPC will apply to close the channel. Or when the time of the channel has expired, the smart contract actively closes MPC.

III. PROBLEM STATEMENT

A. Assumption

We deploy the smart contract on the blockchain network. The smart contract allows users in the network to perform the functions for developing decentralized applications. Compared with Bitcoin, Ethereum can support Turing-complete smart contract script that can be developed by Truffle. Therefore, we only discuss Ethereum. We have some assumptions.

- We assume that there is already a blockchain communication network, which means that any user in the blockchain can send a message to any other and get a response.
- We assume that any peer participating in the multiple channel knows all counterparties participating in the same channel, and they can send authenticated and encrypted messages to each other.
- We assume that there are enough TPCs to form a connected topology and create viable multi-party channels.

B. Symbol Definition

An MPC network can be defined as an undirected graph $G = (V, E)$, where V denotes the set of blockchain addresses and E denotes the opened and active payment channels between two arbitrary peers. Each TPC in E can be defined as $tpc_{id} = (id, b_1^0, b_2^0, b_1^{now}, b_2^{now}, version, expire, busy, counterparty, add)$. An MPC is defined as $mpc_{id} = (id, parties, tpcs, version, expire, otherparties)$. The definition of symbols is shown in Table I.

In addition, the *counterparty* and *add* are two functions while $tpc_{id}.counterparty(v_i)$ returns the other party of v_i in the TPC, and $tpc_{id}.add(v_i, c)$ adds c tokens to v_i (c can be negative). Note that the equation $b_1^0 + b_2^0 = b_1^{now} + b_2^{now}$ always holds. The function $m_{id}.otherparties(partyid)$ returns a set of addresses of all the parties except the party with identifier *partyid*. If $busy = 1$ holds, the TPC has already joined in a multi-party channel and it is not allowed to be updated by transactions that are not in the specific channel. The version number will be used to prevent malicious users from fraud.

IV. SOLUTION OVERVIEW

In this section, we will briefly introduce the three stages: creation, update, and closure.

Creation: To create a MPC, a user needs to make a request and the consent of other users in the channel must be obtained. The multi-party payment channel service (MPCS) handles the messages from the users and contact with the smart contract. The smart contract will verify the signatures to ensure the security of the multi-party creation. Upon the MPC is created, it will return a unique identifier of the MPC. The involved TPCs will be locked until the MPC is closed.

Update: The MPC implements multi-party transactions through channel update. The update is executed off-chain instantly, which avoids huge latency in blockchain transactions. In the process of update, we design channel selection and multi-party transaction optimization strategies to reduce transaction overhead.

Closure: The MPC can be closed as long as all the parties agree with it. All the TPCs in the MPC should be released. If no party requests to close the channel until the time is run out, the channel mpc_{id} will be removed from the MPC network. The settlement is carried out according to the latest channel version.

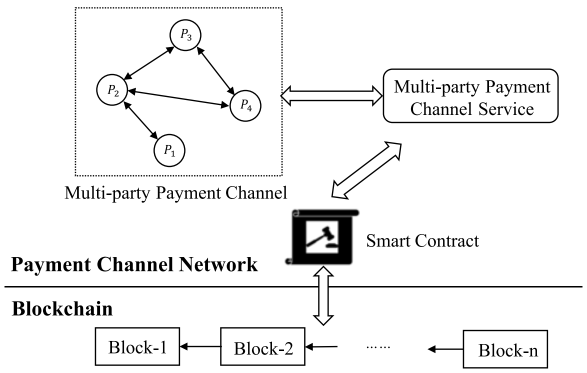


Fig. 3. Multi-party payment channel.

A simple example of the MPC network is shown in Fig. 3. P_1, P_2, P_3, P_4 are four parties in the MPC, and the four black double arrows represent four existing TPCs. A service node existing in the transaction chain can provide MPCS. The service node can initiate the transaction and participate in creation, updating and closure of MPC. Compared the peer-to-peer communication mode, it can facilitate the interaction among users and reduce the communication pressure. In future work, the reward mechanism can be set for the service node to promote the transaction.

V. Detail Construction

In this section, we describe our solution in detail and give completed definitions and procedures of MPC network. To manage the payment channels (both TPC and MPC), we assume that there are consistently *n* peers in the ledger of the blockchain $\mathcal{L}$, which are denoted as a set $\mathcal{P} = \{P_i \mid i \in (1, 2, 3, \ldots, n)\}$. We can deposit tokens to a smart contract and withdraw the balance if P_i is going to join or exit a payment channel. The detailed procedures of the creation, update, and closure of MPCs are illustrated in Algorithm 1.

A. Create a Multi-Party Payment Channel

First, one of the parties *i* that is going to create a new MPC send a request message $Req = (mpc-create,\ \delta_i)$ to the MPCS, where

$$mpc-create = \text{``}create\ an\ MPC\ between\ P_1, P_2, P_3, P_4\text{''}$$

and δ_i is the signature of the party. Once the other parties receive the request message forwarded by the service, they check the signature of the sender and reply to a message if the signature is correct, which is denoted as $Reply = (mpc-create-agree, \delta_j)$, where δ_j is the signature of the replier. Otherwise, they will send a rejecting message *Reply* = (*mpc–create–reject*). After collecting all the signatures of the other parties, the service creates a new MPC. Then the involved TPCs in the MPC will keep locked until the MPC is closed. The flag *busy* of each TPC will be set to 1, which means that the TPC has joined in an MPC. If the collection is not completed within a certain period, the process of creating a channel will be stopped. If it's successfully created, the function will return a unique identifier of the new MPC to the service and then the service will notify all the parties.

Algorithm 1 Detail Construction of Multi-Party Payment Channel

```
Input: The set of the parties $\mathcal{P}$.
Output: The state of transaction.
1: Multi-party payment channel creation.
2: for $P_i, P_j \in \mathcal{P}$ & $i \neq j$ do
3:     $P_i$ sends a request $(mpc-create, \delta_i)$ to other parties $P_j$.
4:     if $P_j$ agrees & $\delta_i$ verifies then
5:         $P_j$ sends $(mpc-create-agree, \delta_j)$
6:     else
7:         $P_j$ sends $(mpc-create-reject, \delta_j)$
8:     end if
9: end for
10: Generate a new MPC instance m in channel space $\mathcal{C}$.
11: busy=1. version=0.
12: Send (mpc–create, m) to parties.
13: Multi-party payment channel update.
14: for $P_i, P_j \in parties$ & $i \neq j$ do
15:     $P_i$ sends $(mpc-update, \delta_i)$ to the MPCS.
16:     if MPC with identifier id checks & $\delta_i$ verifies then
17:         Return (mpc–update–agree).
18:     else
19:         Return (mpc–update–failed).
20:     end if
21:     Determine which TPCs should be updated according to a
        multi-party transaction consisting of several diverse two-party
        transactions mptx.
22:     Optimize transaction $mptx_{converted}$ according to the
        selected paths and channels.
23:     Notify other parties with the message
        $(mpc-update_{converted}, \delta_i)$.
24:     if $P_j$ from parties agrees then
25:         $P_j$ sends $(mpc-update_{converted}-agree, \delta_j)$.
26:     end if
27:     Update the balances of the TPCs influenced according to
        $mptx_{converted}$.
28:     $P_i$ replies with a message $(mpc-updated-tpcs, \delta_i)$.
29: end for
30: if All the messages receive successfully then.
31:     The updated MPC $version' = version + 1$.
32:     The (mpc–updated) is sent to parties.
33: else
34:     A message (mpc–update–failed) is sent to parties.
35: end if
36: Multi-party payment channel closure.
37: for $P_i, P_j \in \mathcal{P}$ & $i \neq j$ do
38:     $P_i$ sends a request $(mpc-close, \delta_i)$.
39:     if $P_j$ agrees & $\delta_i$ verifies then
40:         $P_j$ sends $(mpc-close-agree, \delta_j)$
41:     else
42:         $P_j$ sends $(mpc-close-reject, \delta_j)$
43:     end if
44: end for
45: Close the MPC. busy=0.
46: Send all the parties with a message (mpc–closed, id).
```

B. Update a Multi-Party Payment Channel Off-Chain

Transaction Generation: The update of a multi-party channel means a new generation of the multi-party transaction, which occurs at the upper application layer. For example, when four people play a game of poker cards, each round ends with a multi-party transaction. We can settle the transaction through our MPCs.

Update Request: To update an MPC, a party in the MPC must send a request message to the MPCS, which is

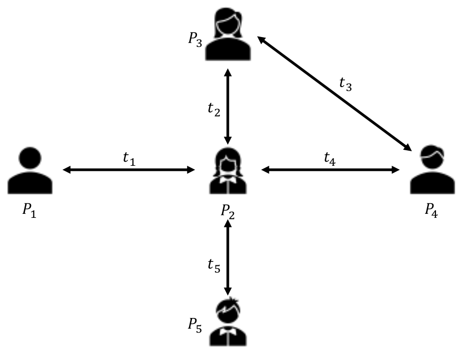


Fig. 4. Multi-party payment channel example.

designed to support interactions between parties and provide multiple services such as channel selection and transaction optimization. The message can be defined as $Req = (mpc{-}update,\ \delta_i)$, where

$$mpc{-}update = \text{“}update\ mpc_{id}\ by\ mptx,\ with\ version = n\text{”}$$

where *mptx* is the generated multi-party transaction, δ_i is the signatures of the message sender. When the request message is delivered, the transaction *mptx* and version number will be verified. If it is a legitimate request, we continue to the following steps. Otherwise, it is terminated.

Channel Selection: The purpose of channel selection is to determine which TPCs should be updated in the topology, and verify whether the balances of the TPCs are enough to accomplish *mptx*. Some of the TPCs in the entire topology are redundant and not all TPCs need to be updated. As depicted in Fig. 4, there are five TPCs in the MPC topology marked as $t_i (i \in \{1, 2, 3, 4, 5\})$. It is not necessary to update all of the TPCs. Suppose that P_5 is the winner after a round of the game, P_1, P_2, P_3, P_4 can pay to P_5 through t_1, t_2, t_4 and t_5. t_3 becomes a redundant TPC in this round. However, it may be updated in the next round if there are not enough tokens in t_2 when P_3 needs to pay to P_2.

The example above exposes that selecting the appropriate channels is necessary to update an MPC. The balance of the channel frequently changes with each update. Choosing the appropriate channels helps to maintain the balances of the channels, thereby supporting more transactions and improving the success rate of transactions. We design a channel selection algorithm to avoid updating the channels' weights for each transaction. We calculate a minimum spanning tree of the topology according to the initial weight and use the channels in the tree several times until it does not work to ensure that all parties are connected and support as many transactions as possible while reducing calculations. In order to measure the quality of the channels to select robust ones, we define some metrics as shown below.

$$Dist_{ij} = \frac{b_{ij} + b_{ji}}{b_{ij} + b_{ji} + \sum_{k \in N_{ij}} \left(b_{ik} + b_{ki} + b_{jk} + b_{kj}\right)} \quad (1)$$

$$Bal_{ij} = \frac{\left|b_{ij} - b_{ji}\right|}{b_{ij} + b_{ji}} \quad (2)$$

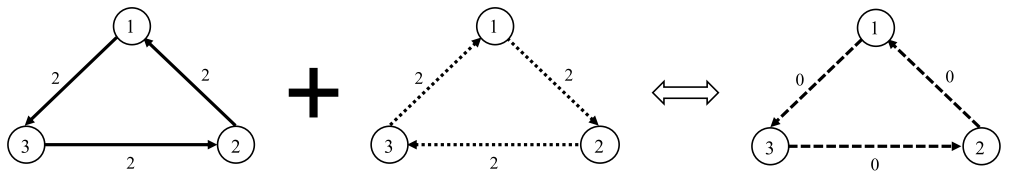
$$W_{ij} = k_1 * \left(1 - Dist_{ij}\right) + k_2 * Bal_{ij} \quad (3)$$

Algorithm 2 Channel Selection

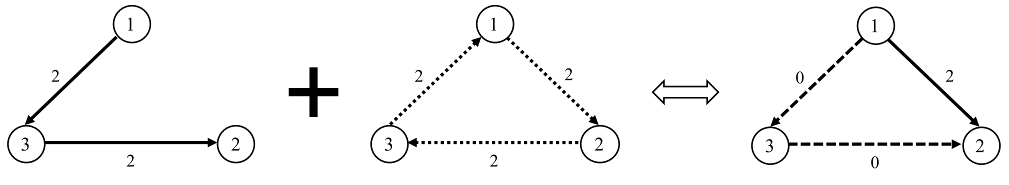
```
Input: The undirected graph topology, $G = (\mathcal{P}, \mathcal{T})$, where $\mathcal{P}$ is
    the set of the parties involved in the MPC and $\mathcal{T}$ the set of the
    existing TPCs. The set of the TPCs' neighbor, $\mathcal{N}$. Two adjustable
    parameters $k_1$ and $k_2$. The set of pending transactions, TX.
Output: The set of paths of the channels which are selected, $T_s$.
1: for each channel $t \in \mathcal{T}$ do
2:     get the two counterparties $P_i$ and $P_j$ of t.
3:     calculate the neighbor parties $N_{ij}$ of t.
4:     calculate $Dist_{ij}$ according to Equation (1).
5:     calculate $Bal_{ij}$ according to Equation (2).
6:     calculate the value of $W_{ij} = k_1 * (1 - Dist_{ij}) + k_2 * Bal_{ij}$.
7:     set $W_{ij}$ to be the weight of the edge in G.
8: end for
9: calculate the minimum spanning tree MST of G according to the
   weights, using the Prim algorithm or Kruskal algorithm.
10: for each transaction $tx \in TX$ do
11:     get the path p in MST for tx.
12:     if the TPCs in p have enough tokens then
13:         add p to $T_s$.
14:     else
15:         get another path $p'$ in G for tx.
16:     end if
17:     if the TPCs in $p'$ have enough tokens then
18:         add $p'$ to $T_s$.
19:     else
20:         No suitable channel to complete transaction tx.
21:     end if
22: end for
23: return the set $T_s$.
```

In Equation (1), the value of $Dist_{ij}$ is defined as the proportion of the total amount of the channel *ij* in the local area. The larger the value, the higher the proportion, and the better the quality of the channel. i, j, k are identifiers of parties in the MPC, b_{ij} denotes how many tokens that *i* can pay to *j* in channel $ij : i \leftrightarrow j$. N_{ij} denotes the identifiers of the neighbor parties of channel *ij*. A neighbor *k* of channel *ij* implies that *k* has two channels connected to *i* and *j*. Take the channel $t_4 : P_2 \leftrightarrow P_4$ in Fig. 4 as an example. P_3 is a neighbor node of channel t_4 because P_3 has TPCs with both P_2 and P_4. In Equation (2), the numerator represents the absolute value of the difference between the balances at both ends of the channel, and the denominator represents the total balance on channel *ij*. It can estimate the imbalance of a channel because we prefer to choose the channel with a smaller Bal_{ij} value, which is in an excellent state and is more able to undertake the transfer. Equation (3) is a combined weight consisting of $Dist_{ij}$ and Bal_{ij}. It takes into account the local proportion and balance extent of the channel with two adjustable parameters k_1 and k_2. During channel selection, we calculate W_{ij} for every TPC as the weight of the edge in the MPC topology.

The implementation of channel selection is shown in Algorithm 2. We first calculate a minimum spanning tree *MST* according to the weights, which have a good balance state and can cover more transactions than the other TPCs not in *MST*. With increasing transactions, the selected channels will finally be exhausted, but we have a backup solution to select channels from the whole topology *G* if the path in *MST* is not available.



(a) example 1



(b) example 2

Fig. 5. Examples of multi-party transaction optimization.

Transaction Optimization:

Transaction Conversion. If we handle a multi-party transaction as several ordinary two-party transactions, some TPCs will be used repeatedly. It will cause unnecessary communication overhead and the atomicity of multi-party transactions cannot be guaranteed. Therefore, before processing a multi-party transaction, we need to make a conversion. Transaction conversion involves two steps, split and merge. For example, The former step splits every two-party transaction to several transactions in directly connected TPCs. A two-party transaction $mptx_1 = (P_1 \rightarrow P_2 \rightarrow P_3, 1)$ can be converted to $(P_1 \rightarrow P_2, 1)$ and $(P_2 \rightarrow P_3, 1)$. Another transaction $mptx_2 = (P_2 \rightarrow P_3 \rightarrow P_4, 2)$ can be split to $(P_2 \rightarrow P_3, 2)$ and $(P_3 \rightarrow P_4, 2)$. The latter step will merge the split transactions in the former step, resulting in a new transaction set $mptx_{converted} = \{(P_1 \rightarrow P_2, 1), (P_2 \rightarrow P_3, 3), (P_3 \rightarrow P_4, 2)\}$. In this way, the number of update interactions of TPCs will be reduced from 4 to 3. We define an addition between two-party transactions $t_1 = (P_i, P_j, token_1)$ and $t_2 = (P_i, P_j, token_2)$, $t_1 \bigoplus t_2 = (P_i, P_j, token_1 + token_2)$, which means we can merge two two-party transactions to one transaction if the payer and the payee are the same. The details to implement transaction conversion are illustrated in Algorithm 3.

Algorithm 3 Transaction Conversion

Input: The multi-party transaction containing multiple two-party transactions, *mptx* and the set of paths of the channels which are selected, T_s.
Output: Converted transaction, $mptx_{converted}$.

```
1:  for each two-party transaction, $tx \in mptx$ do
2:      get the path $p = (p_1, p_2, \ldots p_n)$ in $T_s$ for $tx$.
3:      split $tx$ to a set $tx_{splited}$ which includes $n - 1$ transactions on directly connected TPCs, $tx_{splited} = (p_i, p_{i+1}, token_i)|i \in [1, n]$
4:      for each transaction $tx_i = (p_i, p_{i+1}, token_i) \in tx_{splited}$ do
5:          if there is a transaction $tx_m \in mptx_{converted}$ and $t_m.payer = tx_i.p_i$, $t_m.payee = tx_i.p_{i+1}$ then
6:              $tx_m = tx_m \bigoplus tx_i$.
7:          else
8:              if there is a transaction $tx_m \in mptx_{converted}$ and $t_m.payer = tx_i.p_{i+1}$, $t_m.payee = tx_i.tx_i.p_i$ then
9:                  let $tx_i{}' = (p_{i+1}, p_i, -token_i)$.
10:                 $tx_m = tx_m \bigoplus tx_i{}'$.
11:             else
12:                 add $tx_i$ to $mptx_{converted}$.
13:             end if
14:         end if
15:     end for
16: end for
17: return $mptx_{converted}$.
```

After the transaction conversion is completed, we have converted the multi-party transaction to transactions on directly connected TPCs with length 1. There are no repeated transactions on the same TPC, which avoids unnecessary interactions between two same parties and reduces communication overhead.

Reverse Transaction: The topology of the converted multi-party transaction can be considered as a directed graph, and the transaction in the graph can be further optimized especially when there are cycles. As shown in Fig. 5, there are three transactions in example 1, but everyone's total balance will not be changed. Similarly, we get the optimized result of example 2 by adding a reverse ring transaction which does not affect the correctness of the original multi-party transaction, so the two transactions in example 2 are converted to one transaction. It can be seen that we can optimize the existing transaction topology by adding reverse ring transactions to achieve the purpose of reducing off-chain transactions.

We define any two users on a TPC as *a* and *b*, and their balances are $bal_{a,b}$ and $bal_{b,a}$, respectively. The original transaction in the channel is denoted as $t_{a,b}$, which means *a* should pay $t_{a,b}$ tokens to *b*. The value of the added reverse ring transaction is $\phi_{b,a}$, this indicates that *b* pay back $\phi_{b,a}$ tokens to *a* from the opposite direction. We define a function *f*(*a*, *b*) as Equation. (4), which expresses that if the values of the original transaction and the reverse transaction are equal, then the original transaction is eliminated. Otherwise, there is still a transaction in the TPC. So we can define this optimization problem as the form of Equation. (5), to eliminate as many transactions as possible in the graph.

$$f(a, b) = \begin{cases} 1, & if \quad t_{a,b} = \phi_{b,a} \\ 0, & others \end{cases} \tag{4}$$

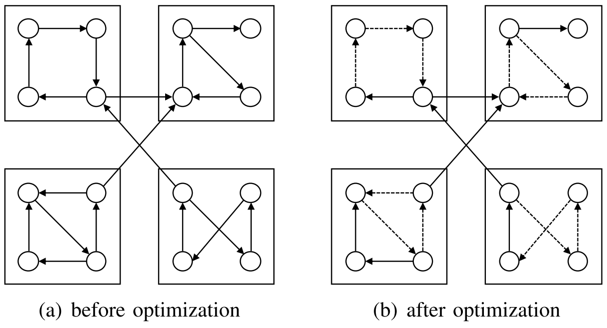


Fig. 6. Example of ring transactions optimization.

$$\begin{cases} maximize & \Sigma_{a,b} f(a,b) \\ s.t. & \forall a\colon \Sigma_a \phi_{a,b} = \Sigma_a \phi_{b,a} \\ & \forall a,b\colon t_{a,b} \geq 0 \Leftrightarrow \phi_{b,a} \geq 0 \\ & \forall a,b\colon \phi_{b,a} - t_{a,b} \leq bal_{b,a} \end{cases} \quad (5)$$

In Equation. (5), the constraint $\forall a : \Sigma_a \phi_{a,b} = \Sigma_a \phi_{b,a}$ means for every party in the graph. The added reverse transaction ought to ensure that the total amounts of its in-degree and out-degree are the same, so as not to affect the correctness of the original transaction. And $\phi_{b,a} - t_{a,b} \leq bal_{b,a}$ means that the user must have sufficient balance to complete the converted transaction. Since it is rather difficult to get the global optimal solution for this problem, especially when there are many parties in the MPC and multiple ring transactions overlap, we propose a local optimization strategy. First, we find all ring transactions in the multi-party transaction topology and then optimize each ring transaction individually. To find all the circuits in the topology, we can use the algorithm presented in [15]. The time complexity of this algorithm is $O((n + e)(c + 1))$, where n denotes the number of nodes, e denotes the number of edges and c is the number of circuits in the topology. For each circuit, we choose the value with the most occurrences on the edges as the payment amount of the reverse ring transaction, because this will eliminate the most edges. There is an example of the optimization of ring transactions shown in Fig. 6. There is a circular trading chain in each square, so we find and optimize the transactions by adding reverse ring transactions. The result is shown in Fig. 6(b), in which the dotted arrows represent the transactions that are eliminated. Through this strategy, we can minimize the number of transactions in the MPC, thereby reducing the number of interactions with the blockchain and improving the efficiency of transactions. By the way, when implementing the functionalities of channel selection and transaction optimization in an integrated project, they can be included in the MPCS, which is the same as that of the Lightning Network and the Raiden Network.

Update MPC: After completing the above steps, there is a converted multi-party transaction $mptx_{converted}$, and the parties in the MPC are ready to update the state. An update request is sent by the MPCS to the parties. The request message is:

$$mpc{-}update_{converted} = \text{``}update\ mpc_{id}\ by\ mptx_{converted},\ with\ version = n\text{''}$$

If all the parties agree with the update, the update procedure will be performed. However, if the balance of any party is insufficient, the party will withdraw from the transaction and close the payment channels associated with his/her node. The current round of multi-party transactions fails. The channel state is rolled back to the state of the previous round.

C. Update a Multi-Party Payment Channel On-Chain

At any time, a party is allowed to send a request to the smart contract to register a new state for the MPC. This is achieved by calling the *updateMPCState* function exposed by the smart contract. The caller must submit the *id* of the MPC and a new *state* of it to register. The *version* number and the signature set are used to verify the correctness.

Malicious parties who submit an old version state of an MPC to the smart contract can be easily spotted and will be punished by deducting his/her deposit. Usually, only one registration is needed before the channel is closed.

D. Close a Multi-Party Payment Channel

To close an MPC, a party of it is required to perform similar procedures as when creating an MPC. First, one of the parties i that is going to close a new MPC sends a request message $Req = (mpc{-}close,\ \delta_i)$ to the MPCS, where

$$mpc{-}close = \text{``}closeaMPCmpc_{id}\text{''}$$

and δ_i is the signatures of the message which is signed by the party who requests to close mpc_{id}. Upon receiving the request, the MPCS will inform the other parties in the MPC.

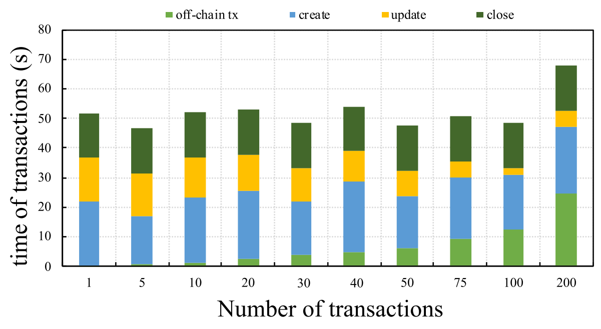
After collecting all the agreements of the other parties, the TPCs in mpc_{id} are settled according to the latest version of mpc_{id}. Then every TPC involved in the MPC will be changed to be unlocked, *busy* of each TPC will be set to 0, which means it is free to participate in other MPCs now. If the collection is not completed before the time is out, the process of closing an MPC will be stopped. If it is closed successfully, the smart contract will notify the parties and remove the unique MPC in the context.

VI. Evaluation

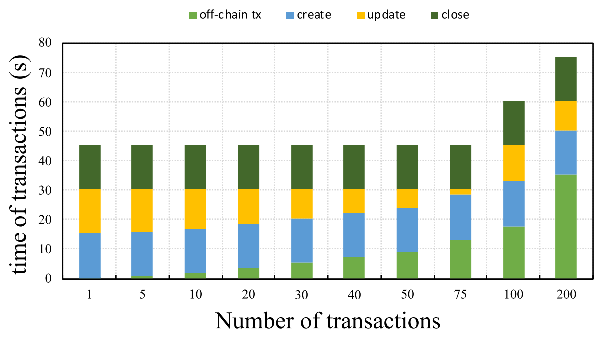
We deployed a smart contract of MPC network to the development blockchain generated locally by Truffle. In addition, we used the Ethereum JavaScript API web3.js to develop the client to interact with smart contracts. We mainly evaluated three performances of MPC network: delay, success rate and natural gas cost of transactions.

A. Transaction Latency

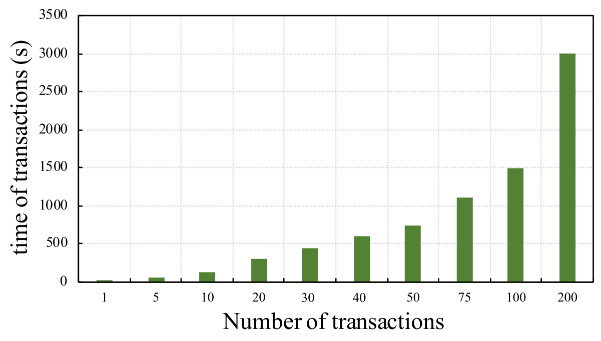
We implement both TPC and MPC on Ethereum, and compare their transaction delays with the delays of Ethereum transactions. Please note that we set the network delay to 50ms indicating a lower latency network, while the blocking time in Truffle is set to 15 seconds [16], [17]. We first create payment channels (TPC or MPC), then execute off-chain transactions multiple times, and finally update the latest state of the channels to the blockchain and close them. As shown in Fig. 7,



(a) Two-party payment channel transactions



(b) Multi-party payment channel transactions



(c) Ethereum transactions

Fig. 7. Transaction latency of different payment methods.

we can see that the waiting time of Ethereum transactions increases almost linearly with the increase in the number of transactions. Every time a transaction is initiated, we must wait approximately 15 seconds to confirm the transaction. At the same time, the waiting time in Fig. 7(a) and Fig. 7(b) consists of four parts: the time to create the channels, update off-chain, update on-chain and close the channels. The results show that with the increase in the number of transactions, the overall delay of the payment channel is much smaller than the delay of the Ethereum transaction.

As shown in Fig. 7(b), the time of transactions is approximately constant when the number of transactions is less than 100. Except for the necessary transactions on the chain, many interaction processes off the chain do not need to interact with the blockchain, which has a very low delay. Therefore, the time is almost constant. As shown in Fig. 7(a) and Fig. 7(b), we find that when the number of transactions is bigger than or equal to 100, the time of transactions is beneficial to TPC. Because the channel resource depletion and imbalance will be occurred with the number of transactions increasing. Compared with TPC, the MPC has more significant delays due to complex channel paths. Nevertheless, our scheme has less transaction delay than Ethereum.

B. Transaction Success Rate

The transaction success rate is the key performance of the payment channel network. A payment channel network with a higher success rate is more robust and can handle more transactions on the network without worrying about channel imbalance in a short time. We have studied the influence of amount of payment, the number of nodes, and topology structure on the success rate of transactions.

Topology structure: In the case of the same number of nodes and ether nodes, we compared the success rate of transactions in random topology and hub-and-spoke topology. The random topology in our simulation is generated by NetworkX. In the initial state, the balances of the two parties of the payment channel are the same. As shown in Fig. 8, we conducted three experiments on different numbers of nodes with the same batch of transactions in two types of topologies. Obviously, the success rate of random topology is higher. The fact is that in addition to central nodes, hub-and-spoke networks can easily lead to channel imbalances after multiple transactions. However, in a payment network with random topology, other channels can replace exhausted channels. Further observation shows that the success rate decreases with the number of transactions increasing under the same number of nodes. The channel resource will exhaust with more transactions, resulting in channel imbalance and reducing the transaction success rate.

Numbers of nodes: In order to analyze the impact of the number of nodes on the transaction success rate, we set three groups of experiments in random topologies with different transaction amounts including less than 5, 10 and 50 respectively. Fig. 9 shows that under the same number of transactions, the more nodes the network has, the higher the transaction success rate is. When some channels are exhausted, the standby channel can be selected for the transaction. Therefore, the transaction success rate can be improved.

The distribution of payments: The success rate of transactions is also affected by the distribution of payments. A uniform distribution means that every node in the network has the same probability of being a payer or payee. Moreover, with a normal distribution, the payment will be focused on certain nodes, just as a lot of buyers pay to several sellers. We have did three groups of experiments, including one group of payments following a uniform distribution and two groups of payments that follow two normal distributions with diverse parameters. As shown in Fig. 10(a), because unbalanced channels have a high probability of being charged in the opposite direction the success rate is always close to 1.0. It changes the unbalanced channels to balanced channels again. But in Fig. 10(b) and Fig. 10(c), the success rate decreases as the number of transactions increases. At this time, many users need to make payment transactions with a few users. There will be more transactions in one direction and fewer reverse transactions. With the number of transactions increasing, most

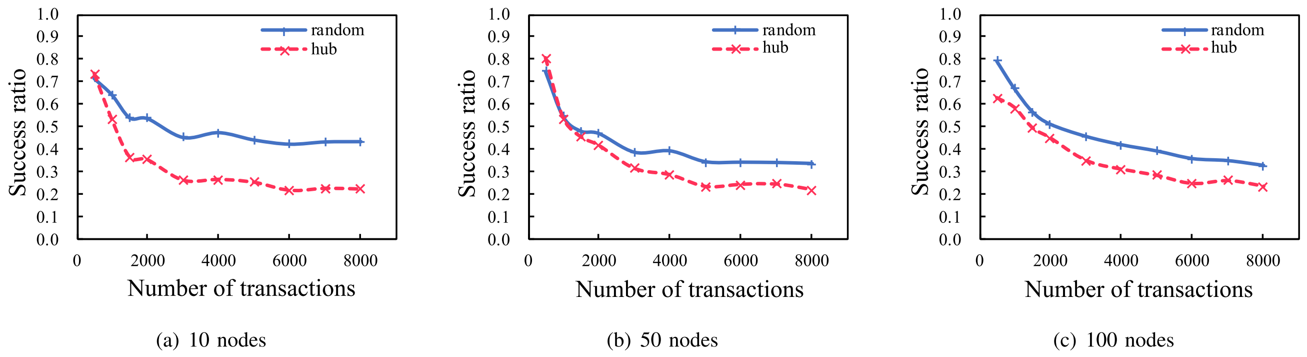


Fig. 8. Impact of topology on the transaction success rate.

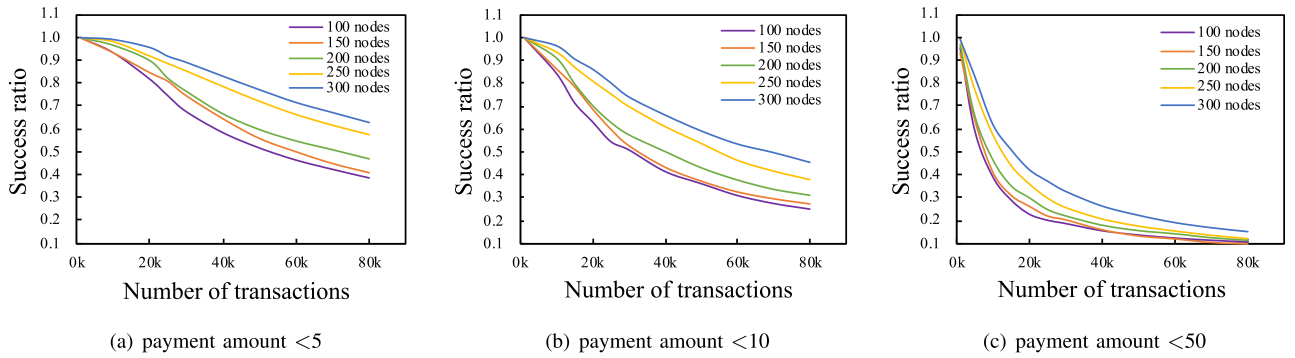


Fig. 9. Impact of node number on the transaction success rate.

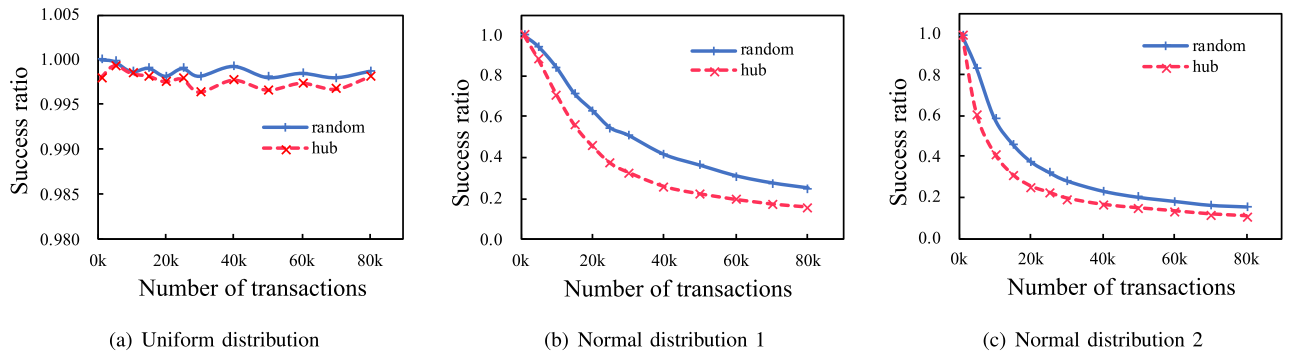


Fig. 10. Impact of the distribution of payment on the transaction success rate.

payment channels may reach an unbalanced state, reducing the transaction success rate.

Amount of payment: In order to research the impact of transaction amount on success rate, we set four kinds of experiments with 5, 10, 50 and 100 of payment amounts respectively. As shown in Fig. 11, we can see that if the less the payment amount, the higher the transaction success rate is. Since the balance of each channel is limited, the payment channel network is more suitable for small payments amounts. Since the balance of each channel is limited, the payment channel network is more suitable for small payment amount. The larger the transaction amount is, the faster the channel will be exhausted. Many channels will quickly reach a state of imbalance, which reduces the transaction success rate.

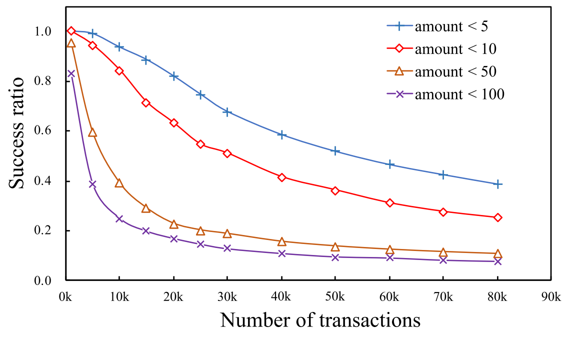
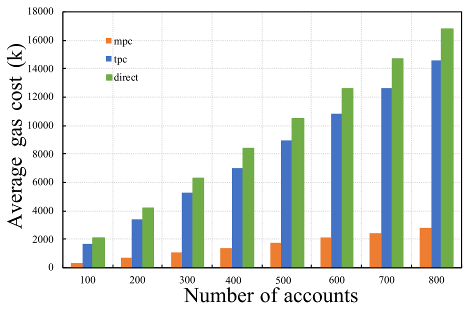


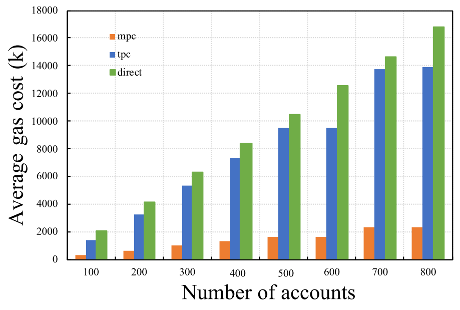
Fig. 11. Impact of the amount of payment on the transaction success rate.

C. Gas Cost

Gas cost may be the most concerned performance for users because the payment fee is calculated based on gas when trading on Ethereum. We set up eight groups of different scales experiments and carry out the topologies cut out from the Lightning Network and random topologies generated by NetworkX respectively. The premise is that the balances at both ends of the initial channel are the same and sufficient and the entire topology is connected. Note that in our simulation,



(a) LN topologies



(b) Random topologies

Fig. 12. Gas cost of different methods in different topologies.

we only request an update once after every 10 transactions. In practical applications, a channel can be updated only once before it is closed, which is beneficial to reduce gas consumption. As shown in Fig. 12, MPC network costs the least gas compared with other baselines.

VII. Related Work

Due to the step and PoW (Proof of Work) mechanism, long delay and low throughput are the biggest challenges of blockchain. Most of the existing work focuses on sharding, side chains and payment channel networks. The basis of sharding is to divide the blockchain network into several parts [5], [18]. Each part only stores a part of the blockchain instead of the entire blockchain. This idea is like a distributed database. It is expensive to increase throughput by executing transactions in parallel. Side chains are also called fixed side chains. Users can transfer their assets from one blockchain to another securely, so the source blockchain (such as the Bitcoin blockchain) can be scaled. The key to the side chain is how to ensure security when conducting transactions between different blockchains. Therefore, the payment channel network is the most promising technology proposed to solve the above problems.

Most of the current works use a payment hub to create MPC. Spankchain can establish the payment channel with multiple users through the hub [19]. In [13], to achieve efficiency, flexibility in the absence of a trusted third party, a secure N-party payment hub is proposed, which allows multiple participants within the payment hub to make concurrent and direct off-chain coin transfers. In [12], it proposes an efficient and lightweight off-chain system, which can shorten the routing path for the payment network and perform secure off-chain cross-channel transaction with hub. However, to realize the hub connection, it needs to mortgage a large amount of money to establish payment channels with many users. In addition, hub nodes will form a centralized structure, which is contrary to the basic idea of decentralization of blockchain.

The two-party micropayment channel is proposed by Christian Decker and Roger Wattenhofer [20], which allows instant off-chain transactions between two parties while ensuring the security of the Bitcoin network. Duplex micropayment channels use Bitcoin contracts to ensure that both parties will never violate the agreement. HTLC is used for payment between two parties who do not have a directly connected channel. The life cycle of a payment channel includes setup, reset, disassembly and refresh stages. In Bitcoin, the Lightning Network is the most popular payment channel. It uses RSMC and HTLC to support two-way payments, and payments can be made across multiple channels. Like Raiden Network, Raiden Network is an off-chain scaling solution for Ethereum. However, most of the work does not consider the establishment of MPC, which limits the application scenarios.

The performance of the payment channel network is affected by factors such as network topology, routing algorithms and channel rebalancing strategies to avoid imbalances [21]. There are some researchers dedicated to studying these aspects to improve the performance of payment networks [22], [23], [24], [25]. Osuntokun designs a protocol for atomic multi-path payment through the Lightning Network in [26], which helps improve the payment success rate by eliminating the capacity limitation of a single path. However, it does not fully consider the rebalancing of payment channels. Perun [8] provides a new method called virtual payment channel, which can interact between two parties through multiple channels. However, Perrin is not designed for routing transactions. It only applies to two parties connected by the same payment center and does not support long-distance transactions.

The state channel is originally proposed by Jeff Coleman in 2015. This is the general form of payment channels, applying the same idea to any kind of state change operations usually performed on the blockchain [27]. The basic life cycle of a state channel is to lock the state of participants, update the state by constructing and signing transactions off-chain, and finally submit the state to the blockchain when the channel is about to close. Dziembowski *et al.* [28] gave the first complete specification of the general state channel network, which is an extension of the payment channel network. The author defines some ideal functions, including state channel creation, contract instance update, contract instance execution and state channel closure. The state channel network model allows virtual channels to be constructed for any two parties through any number of intermediaries. Reference [29] extends the model of the multi-party state channel. Multi-party state channels include multiple parties and execute arbitrarily complex smart contracts involving two or more parties. Some projects apply state channels, such as Celer Network [30] and Counterfactual [9].

VIII. Conclusion

In this paper, we have presented MPC network, a universal multi-party payment network model, which allows any number of users to participate in the same multi-party channel and performs atomic multi-party transactions based on the original TPC. In order to support as many transactions as possible and avoid repeated channel updates through the blockchain network, we have proposed a channel selection strategy to select high-quality transaction channels, thereby increasing the success rate of transactions. In addition, the novel multi-party transaction method in MPC network reduces routing and communication overhead. We have provided detailed experiments to examine the influence of various factors on transaction success rate and gas cost, which can be used as a reference when designing payment channels.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Business Review, White Paper, Oct. 2008.

[2] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the bitcoin network," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1598–1615, Jun. 2021.

[3] W. Hao *et al.*, "Towards a trust-enhanced blockchain P2P topology for enabling fast and reliable broadcast," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 904–917, Jun. 2020.

[4] I. Stoepker, R. Gundlach, and S. Kapodistria, "Robustness analysis of bitcoin confirmation times," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 4, pp. 20–23, 2021.

[5] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 17–30.

[6] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," in *Proc. Lightning Labs,* 2016.

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum, Zug, Switzerland, Yellow Paper, 2014.

[8] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *Proc. IEEE Symp. Security Privacy*, 2019, pp. 106–123.

[9] J. Coleman, L. Horne, and L. Xuanji, "Counterfactual: Generalized state channels," 2018.

[10] M. Carbonell, J. M. Sierra, and J. Lopez, "Secure multiparty payment with an intermediary entity," *Comput. Security*, vol. 28, no. 5, pp. 289–300, 2009.

[11] K. Jin, J. Zhu, and G. Fan, "MET: Multi-party e-commerce transaction model," in *Proc. IEEE 3rd Int. Conf. Privacy*, 2011, pp. 587–590.

[12] J. Zhang, Y. Ye, W. Wu, and X. Luo, "Boros: Secure and efficient off-blockchain transactions via payment channel hub," *IEEE Trans. Dependable Secure Comput.*, early access, Dec. 14, 2021, doi: 10.1109/TDSC.2021.3135076.

[13] Y. Ye, Z. Ren, X. Luo, J. Zhang, and W. Wu, "Garou: An efficient and secure off-blockchain multi-party payment hub," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4450–4461, Dec. 2021.

[14] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-two blockchain protocols," in *Proc. Int. Conf. Financ. Cryptogr. Data Security*, 2020, pp. 201–226.

[15] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM J. Comput.*, vol. 4, no. 1, pp. 77–84, 1975.

[16] "Definition Lower Latency." [Online]. Available: https://www.suse.com/suse-defines/definition/lower-latency/ (Accessed: 2022).

[17] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2019, pp. 625–656.

[18] A. E. Gencer, R. van Renesse, and E. G. Sirer, "Service-oriented sharding with Aspen," 2016, *arXiv:1611.06816*.

[19] A. Soleimani and D. Vogelaere. "SpankChain—A Cryptoeconomic Powered Adult Entertainment Ecosystem Built on the Ethereum Network." [Online]. Available: https://spankchain.com/static/SpankChain (Accessed: Oct. 23, 2017).

[20] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Proc. Symp. Self-Stabilizing Syst.*, 2015, pp. 3–18.

[21] M. Conoscenti, A. Vetrò, and J. C. De Martin, "Hubs, rebalancing and service providers in the lightning network," *IEEE Access*, vol. 7, pp. 132828–132840, 2019.

[22] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. Int. Conf. Financ. Cryptogr. Data Security*, 2019, pp. 508–526.

[23] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 439–453.

[24] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Soc. Open Sci.*, vol. 5, no. 8, 2018, Art. no. 180089.

[25] F. Engelmann, H. Kopp, F. Kargl, F. Glaser, and C. Weinhardt, "Towards an economic analysis of routing in payment channel networks," in *Proc. 1st Workshop Scalable Resilient Infrastruct. Distrib. Ledgers*, 2017, p. 2.

[26] O. Osuntokun. "AMP: Atomic Multi-Path Payments Over Lightning." 2018. [Online]. Available: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html

[27] "State Channels." [Online]. Available: https://www.jeffcoleman.ca/state-channels/ (Accessed: Nov. 6, 2015).

[28] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 949–966.

[29] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2019, pp. 625–656.

[30] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring Internet scale to every blockchain," 2018, *arXiv:1810.00037*.

Yanjiao Chen (Member, IEEE) received the B.E. degree in electronic engineering from Tsinghua University in 2010 and the Ph.D. degree in computer science and engineering from the Hong Kong University of Science and Technology in 2015. She is currently a Bairen Researcher with the College of Electrical Engineering, Zhejiang University, China. Her research interests include network security, mobile sensing, and spectrum management.

Xuxian Li received the B.S. and M.S. degrees in engineering from Wuhan University in 2017 and 2020, respectively. His research interests include cryptocurrency, blockchain network, and machine learning.

Jian Zhang received the B.S., M.S., and Ph.D. degrees in computer architecture from Wuhan University in 1998, 2001, and 2006, respectively. He is currently a Professor of Computer School with Wuhan University. His research interests include cloud computing, Internet of Things, and big data.

Hongliang Bi (Member, IEEE) received the B.S. degree in engineering from the Xi'an University of Technology in 2013, the M.S. degree in engineering from Soochow University in 2015, and the Ph.D. degree in engineering from Wuhan University in 2020. He is currently an Assistant Professor with the School of Cybersecurity, Northwestern Polytechnical University. His research interests include machine learning, IntelliSense, privacy security, and Internet of Things.