MPCN-RP

Abdul Razique
CSE Department
IIT Ropar
Rupnagar, Punjab [India]
2023csm1001@iitrpr.ac.in

Ashish Kohli
CSE Department
IIT Ropar
Rupnagar, Punjab [India]
2023csm1002@iitrpr.ac.in

Pritam Mahajan
CSE Department
IIT Ropar
Rupnagar, Punjab [India]
2023aim1012@iitrpr.ac.in

Vaibhav Barsaiyan
CSE Department
IIT Ropar
Rupnagar, Punjab [India]
2023csm1019@iitrpr.ac.in

Pradeep Dalal
CSE Department
IIT Ropar
Rupnagar, Punjab [India]
2023aim1008@iitrpr.ac.in

***Abstract*—As a result of the requirement to reach consensus among all network peers, blockchain-based cryptocurrencies have significantly constrained transaction throughput and latency. Payment channels offer an encouraging remedy for this problem, as they let two peers make an infinite number of atomic, trust-free payments without depleting the blockchain's resources. Paying between two peers without direct channels is possible using a linked payment channel network since it uses several intermediary nodes to forward and charge for the transactions. Nonetheless, distinct payment channel networks have distinct billing tactics for intermediary nodes. To give consumers the most cost-effective route in a multi-charge payment channel network, existing works do not yet have a comprehensive routing algorithm. The broad routing protocol that we propose in this study, called MPCN-RP, is a network of payment channels with various fees. Our comprehensive experimental results demonstrate that MPCN-RP performs much better in terms of time and cost than the baseline algorithms on both simulated and actual payment channel networks.**

***Index Terms*—Ethereum, Bitcoin, MPCN-RP, HTLC, Dijkstra,**

I. INTRODUCTION

The consensus process of blockchain networks, such as Ethereum and Bitcoin, has problems with transaction throughput and latency. This approach has restricted scalability and slower transaction processing times because it requires agreement from all peers in the network.

The multiple-channel network is linked to the payment channel. The Multi-Charge Payment Channel Network Routing Protocol, or MPCN-RP, is a suggested routing protocol meant to rectify the flaws in the current methods. In networks with several charging nodes, it seeks to determine the most economical route for transactions. The best path is probably chosen by the protocol taking into account variables like transaction requirements, network structure, and node fees.

State channels, another name for payment channel networks, allow off-chain transactions between two or more parties. Each operation allows these players to do multiple transactions on the Layer 1 blockchain. Benefits involve: Payment channels enable instantaneous, low-cost transactions, which lowers transaction costs and boosts throughput. As an illustration, a network of payment channels is called the Lightning Network. Off-chain expansion solutions can only be entirely implemented with payment channels since a user cannot establish a channel with every party with whom he wishes to transact. To solve this issue, Lightning Network created the Hashed [3] Time-Lock Contract (HTLC), a system of end-to-end payment [7] channels that links two dispersed nodes.

II. RELATED WORK

A. *Dijkstra-original*

MPCN-RP is built on the Dijkstra algorithm, but with a twist. Unlike the original, it factors in both path weight and hop count. This comparison with traditional Dijkstra helps us see how MPCN-RP affects results based on both factors.

B. *Dijkstra-LN*

A popular payment [7] network called Lightning Network determines middle node fees using an improved Dijkstra algorithm. The second-to-last node in the Lightning Network pays the recipient first, followed by the following node. Therefore, to identify the shortest path while utilizing Dijkstra-LN, they switch the sender and recipient, making charge calculation easier. Nevertheless, Dijkstra-LN eliminates fees associated with a hop count and just considers basic fees. We must compare the variations in fees and time to comprehend how it differs from MYPBT.

C. *MPCN-RP-cut*

We develop MPCN-RP-cut, an algorithm without the switch, to see if changing the sender and receiver in MPCN-RP functions. We perform the method from every node adjacent to the sender since there is no cost for the channel between the sender and the first IN. Next, we select the least expensive path that satisfies the capacity restriction.

D. *DFS: Depth First Search (DFS)*

By examining every possibility, DFS can determine the most affordable route. However, this may require a large amount of processing resources for large networks. Thus, it is necessary to compare the speeds of MYPBT and DFS. Once 30 pathways are found, we force DFS to halt.

III. BACKGROUND OF THE TOPIC

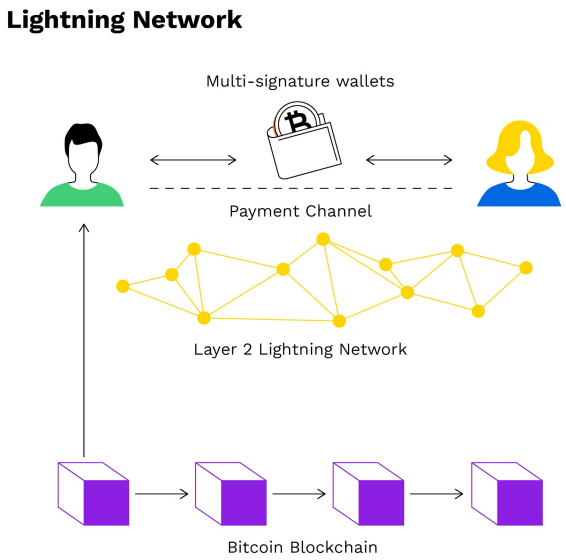

Fig. 1. Lightining Network

Source: https://www.google.com/imgres?q=lightning%20networks&imgurl=https%3A%2F%2Fbitpanda-academy.imgix.net

A. *Lightining Networks*

For blockchain networks such as Bitcoin, the Lightning Network provides a "layer 2" scalability option. By executing transactions off-chain—that is, without having to instantly record them on the blockchain. It is intended to speed up and reduce the cost of transactions. Rather, on the underlying blockchain, several transactions are aggregated and resolved on a regular basis.

This is how it functions:

1. Payment [7] Channels: Users establish direct channels of payment [7] with one another. They may transmit money back and forth through these routes without utilizing the blockchain for each transaction.

2. Routing Payments: Payments between two users can be routed through other network channels if they aren't directly connected to a payment [7] channel. This implies that payments may pass via a number of routes before arriving at their final destination.

3. Instant Transactions: As opposed to on-chain transactions, which could take some time to manage, off-chain transactions allow for quick confirmation. The goal of the Lightning Network is to provide a network of interconnected payment [7] channels that will increase scalability, lower transaction fees, and make it easier to microtransactions. It's an optimistic remedy for the scaling problems that blockchain networks like Bitcoin are now facing.

B. *Payment [7] Channel-Networks*

The payment channel network, often referred to as state channels, allows two or more parties to conduct off-chain transactions. The Layer 1 blockchain allows these players to perform numerous transactions for every activity. Pay channels offer instantaneous and low-fee transactions, which lowers transaction costs and boosts throughput. A payment [7] channel network is the Lightning Network, for instance. Although payment channels form the basis of off-chain growth options, they are insufficient in and of themselves since a user cannot establish a channel with every party with whom he wishes to transact. Lightning Network developed the Hashed [3] Time-Lock Contract (HTLC) to solve this issue. HTLC uses a number of end-to-end payment channels to establish a connection between two randomly disconnected nodes.

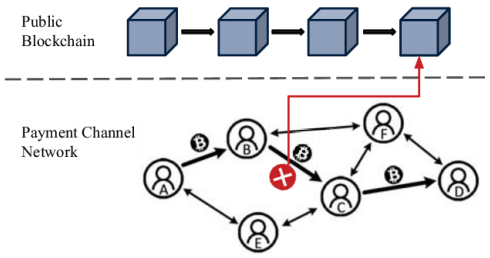

Fig. 2. PCN Illustration

Source: https://www.google.com/imgres?q=payment%20channel%20methods&imgurl=https%3A%2F%2Fslideplayer.com

C. *Hashed [3] Time-Locked Contracts*

These are a kind of smart contract that is utilized in payment channel networks such as the Lightning Network. HTLCs make it possible for parties who might not completely trust one another to trade securely and without risk. This is how it operates:

1. Hash Locking:

Using a distinct cryptographic hash that they create, the sender locks the cash. Only by disclosing the pre-image, the original data can the hash be created, and hence, the receiver can claim the money.

2. Time Locking:

The amount of time allotted to claim the monies is also limited. The sender may recover the payments if the receiver does not claim them within the allotted period.

HTLCs guarantee that the money is returned to the sender within a predetermined amount of time, or the transaction is properly completed. HTLCs also ensure that either the transaction is completed successfully, or the funds are returned

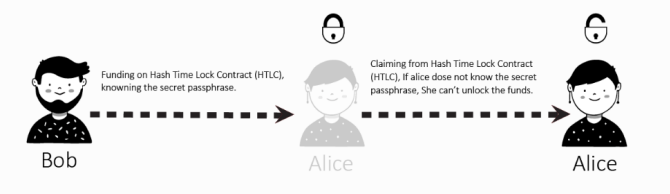

Fig. 3. Hash Locking
Source: https://shuttle.readthedocs.io/en/v0.3.3/htlc.html

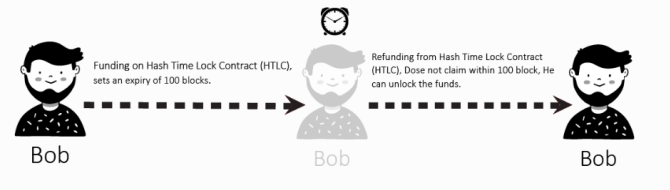

Fig. 4. Time Locking
Source: https://shuttle.readthedocs.io/en/v0.3.3/htlc.html

to the sender after a specified time, providing security and reliability in off-chain transactions.

IV. METHODOLOGY

A. *Cloth [9]-Simulator [10]*

We implemented and modified the paper using a cloth [9] simulator [10] for the payment channel network. Python scripting is used for output compilation, execution, writing, and printing. The majority of the writing is done in the C language. Simulator [10] files are organized into different folders for different tasks. For example, cloth.c is the main file that receives input and stores it in a parameter structure. It then calls the network.c file to build the network and the cloth.c file payment.c file to list all payments in the structure of the payments, sorted by starting time. Finally, it calls the event file to build events such as FINDPATH, SENDPAYMENT, FORWARDPAYMENT, RECEIVEPAYMENT, and so on. The simulation begins for the FINDPATH event, which uses Dijkstra to determine the path. The calculation of Dijkstra uses heap, and heap is maintained by heap.c.

B. *Run-Simulator [10]*

- Step 1:- make build
- Step 2:- /run-simulation.sh [path of Output directory]

```
(base) ashish@ashish-Lenovo-Ideapad-330-15IKB:~/Downloads/cloth-masteredit$ ./run-simulation.sh 123 /home/ashish/Downloads/cloth-masteredit/out/
GSL_RNG_SEED=123
NETWORK INITIALIZATION
PAYMENTS INITIALIZATION
EVENTS INITIALIZATION
INITIAL DIJKSTRA THREADS EXECUTION
Time consumed by initial dijkstra executions: 207 s
EXECUTION OF THE SIMULATION
Time consumed by simulation events: 284.785221 s   Avg hops: 3.000000
COMPUTE SIMULATION OUTPUT STATS
Batch length: 160422.0 ms
Total simulated time: 4812660.0 ms
SIMULATION OUTPUT STATS SAVED IN </home/ashish/Downloads/cloth-masteredit/out/cloth_output.json>
(base) ashish@ashish-Lenovo-Ideapad-330-15IKB:~/Downloads/cloth-masteredit$
```

Fig. 5. Run-Simulator [10] illustration

V. IMPLEMENTING PAPER

A. *Algorithm Design of MPCN-RP*

1) Algorithm-1 : Payment Request

Algorithm 1 The Process of Payment Request

Payment Request.
Input: The recipient's address A_r.
OutInput: The state of the request, *Reqstate*.
1. The sender requests the recipient's address.
2. If the recipient's address is invalid or the recipient does not accept payments, do not send the payment request and set the request state, $Reqstate = 2$.
3. If balances of all channels of the sender are less than the payment amount α then cancel this payment and set the request state, $Reqstate = 3$.
4. The sender encapsulates its own address, the recipient's address and the payment amount into a payment request $R(s, r, \alpha)$,then it sends it to the path-finding unit. Set the request state, $Reqstate = 4$
5 The sender wait for the path p from path-finding unit
6 If the sender receive a path p from path-finding unit, and set the request state, $Reqstate = 1$, turn to **Payment Execution**.
7 End.

Fig. 6. Payment Request Process

2) Algorithm-2 : Path Finding

Algorithm 2 The Process of Path-Finding

Path-finding.
Input: Addresses of the sender A_s and the recipient A_r.
OutInput: The path p.
1. Set the recipient as the starting node v_s and the sender as the destination node v_r.
2. Initialize the distance from v_s, and the hop number of all nodes. If $[v_i, v_s] \in E$, $dis_i = 0$, $hop_i = 0$. Else, $dis_i = +\infty$, $hop_i = 0$
3. Initialize the set *VIS* of nodes which the shortest path has been found and the set *UVIS* of nodes which the shortest path has not been found.
   $VIS = \phi$, $UVIS = V$
4. Move the node with minimum *dis* v_m from *UVIS* to *VIS*.
5. For every node $v_i \in VIS$, if $\exists v_u$, $[v_i, v_u] \in E$ and $dis_u > dis_i + f_u$, then
   update $dis_u = dis_i + f_u$.
   update the hop number and previous node of node v_u, hop_u++, $prep_u = v_i$.
7 If $VIS! = V$ and $v_s \notin VIS$, then return to step 4.
8 Initialize the path p and current node of the path.
   $p = [v_s]$, $v_{cur} = v_s$.
9 Set the previous node of the current node as the current node, $v_{cur} = pre_{cur}$, add pre_{cur} to p.
10 If $pre_i! = v_r$ and $pre_i! = null$, then send p to the sender. Else, return to Step 9.
11 End.

Fig. 7. Path Finding Process

3) Algorithm-3 : Payment Execution

Algorithm 3 The Process of Payment Execution

Payment Execution.
Input: Addresses of the sender A_s and the recipient A_r.
OutInput: The state of the payment *Paystate*.
1. Confirm the address of the recipient and set the current node $v_{cur} = v_r$. Set the state of the payment, $Paystate = 0$.
2. Calculate the fee f_{cur-1} that will be charged from the previous node to the current node.
3. If the current node publishes H(R) to the previous node before the end of the time lock.
   $b_{cur-1,cur} - \alpha - f_{cur-1,cur}$, $b_{cur,cur-1} + \alpha + f_{cur-1,cur}$. Set the previous node of the current node as the current node, $v_{cur} = pre_{cur}$. Return to Step 2.
4. Else, the payment fails. Set the state of the payment, $Paystate = 2$. Go to Step 6.
5. Set the state of the payment, $Paystate = 1$.
6. End.

Fig. 8. Payment Execution Process

4) Algorithm-4 : Update

Algorithm 4 The Process of Update

Update.
Input: The channel state in the network *Chnstate*.
OutInput: End time of the update t_{update}.
1. For all $v_i \in v, v_j \in V$, if the channel $[v_i, v_j]$ between them has been closed, v_i and v_j send $C_{i,j}$ and $C_{j,i}$to the path-finding unit.
2. After receiving $C_{i,j}$ and $C_{j,i}$ the path-finding unit delete $[v_i, v_j]$ from E and change the marks.
   $e_{i,j} = 0$, $e_{j,i} = 0$.
3. For all $v_i \in v, v_j \in V$, if there is a channel between $[v_i, v_j]$ has been built, v_i and v_j send $O_{i,j}$ and $O_{j,i}$to the path-finding unit.
4. After receiving $O_{i,j}$ and $O_{j,i}$ the path-finding unit add $[v_i, v_j]$ to E and change the marks.
   $e_{i,j} = 1$, $e_{j,i} = 1$, request $b_{i,j}, b_{j,i}$.
5. For all $v_i \in v, v_j \in V$, if the balances of the channel has been changed then v_i and v_j send $U_{i,j}$ and $U_{j,i}$to the path-finding unit.
6. After receiving $U_{i,j}$ and $U_{j,i}$ the path-finding unit request new balances from the two nodes.
   Request new $b_{i,j}, b_{j,i}$.
7. End.

Fig. 9. Updation Process

B. *System Model*

1) Payment Channel Network

2) Hashed [3] Time Lock Contract

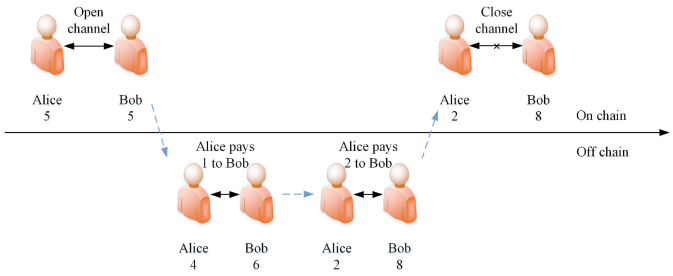

Fig. 10. Example of Payment Channel Network
Source: https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4275028/9792219/9663541/chen1-3139019-large.gif

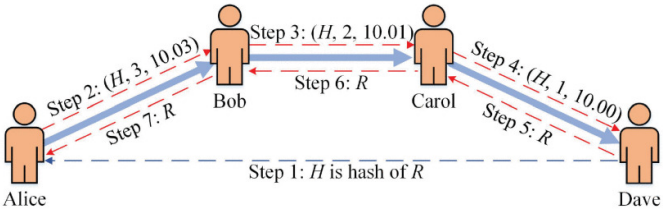

Fig. 11. Example of HTLC
Source: https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4275028/9792219/9663541/chen2-3139019-large.gif

C. *Network Model*

1) Fixed Fee :

$$F^{fix} = \sum_{i=1}^{n} f_i^{fix} \tag{1}$$

2) Time Value Fee :

$$F^{tv} = \sum_{i=1}^{n} f_i^{tv} \tag{2}$$

3) Proportional Fee :

$$F^{prop} = \sum_{i=1}^{n} f_i^{prop} \tag{3}$$

4) Imbalance Fee :

$$F^{im} = \sum_{i=1}^{n} f_i^{im} \tag{4}$$

5) Total Fee :

$$F = F^{fix} + F^{prop} + F^{tv} + F^{im} \tag{5}$$

D. *Problem Formulation*

$$b_{i,i+1} \geq \alpha + \sum_{j=i}^{n} \left[f_j^{fix} + f_j^{prop} + f_j^{tv} + f_j^{im} \right] \tag{6}$$

VI. Proposed Method

Our updated Dijkstra algorithm moves the emphasis from only maximizing distance to assessing the dependability of nodes by including defective probability as a crucial feature in queue ranking. We may efficiently measure each network node's reliability by computing the defective probability, which is defined as the ratio of faulty transactions to total transactions. As a result, there is a greater chance that the algorithm will choose a more dependable path by giving priority to examining pathways through nodes with a reduced chance of error.

```
/* compare the distance used in dijkstra */
int compare_distance(struct distance* a, struct distance* b) {
  uint64_t d1, d2;
  double p1, p2;
  int* faulty_nodes1 = getFaultyArray();
  int* total_passed = getPassesArray();
  double faulty_prob1 = a->node;
  double faulty_prob2 = b->node;
  d1=a->distance;
  d2=b->distance;
  p1=a->probability;
  p2=b->probability;
  /*if(d1==d2){
    if(a->node>=b->node)
      return 1;
    else
      return -1;
  }
  else if(d1<d2)
    return -1;
  else
    return 1;
  */
  if(faulty_prob1==faulty_prob2){
        if(d1==d2){
            if(a->node>=b->node)
              return 1;
            else
              return -1;
          }
          else if(d1<d2)
            return -1;
          else
            return 1;
  }
  else{
        if(d1==d2){
            if(a->node>=b->node)
              return 1;
            else
              return -1;
          }
          else if(faulty_prob1<faulty_prob2)
            return -1;
          else
            return 1;
  }
}
```

```
while(heap_len(distance_heap[p])!=0) {

  d = heap_pop(distance_heap[p], compare_distance);
  best_node_id = d->node;
  if(best_node_id==source) break;
  int* faulty_nodes1 = getFaultyArray();
  int* total_passed = getPassesArray();

  if(faulty_nodes1[best_node_id]/total_passed[best_node_id]>.8){
      continue;
  }
  if(faulty_nodes1[best_node_id]>id){
      continue;
  }
  to_node_dist = distance[p][best_node_id];
  amt_to_send = to_node_dist.amt_to_receive;
```

Fig. 12. Proposed Algorithm

Moreover, the algorithm's ability to discover optimum pathways is guaranteed by the use of distance as a tiebreaker in cases when nodes have equal erroneous probability. Prioritizing shorter distances helps preserve the algorithm's fundamental goal of determining the shortest path from the source node to every other node in the network in situations when nodes display comparable degrees of dependability.

All things considered, this hybrid method blends the concepts of efficiency and fault tolerance, which makes it well suited for applications where reliability and performance are crucial factors, such as distributed systems, communication networks, or blockchain technologies. This increases the possibility that more dependable routes will be chosen by enabling the algorithm to prioritize investigating pathways through nodes with lower possibilities of being incorrect.

VII. Evaluation

A. Analysis with Number of Nodes

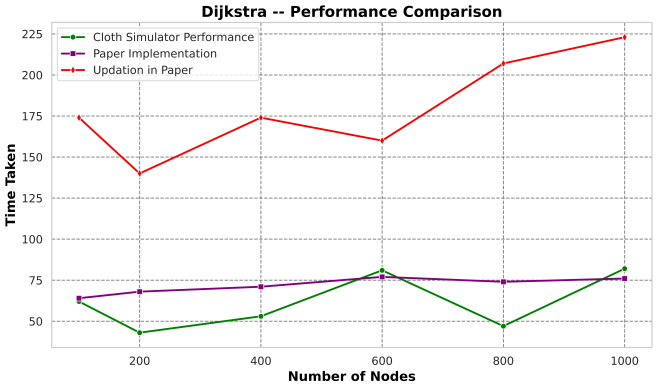

Fig. 13. Dijkstra on Nodes

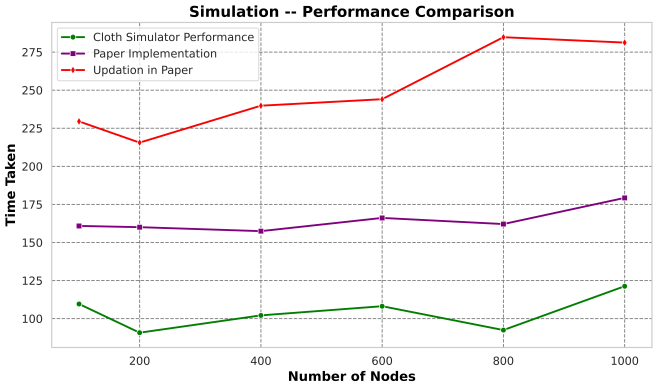

Fig. 14. Simulation on Nodes

B. Analysis with Number of Payments

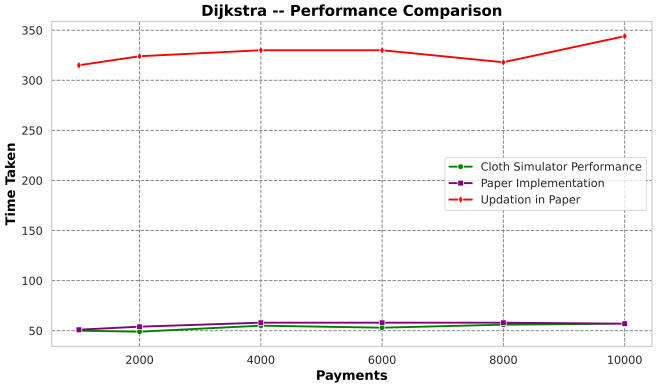

Fig. 15. Dijkstra on Payments

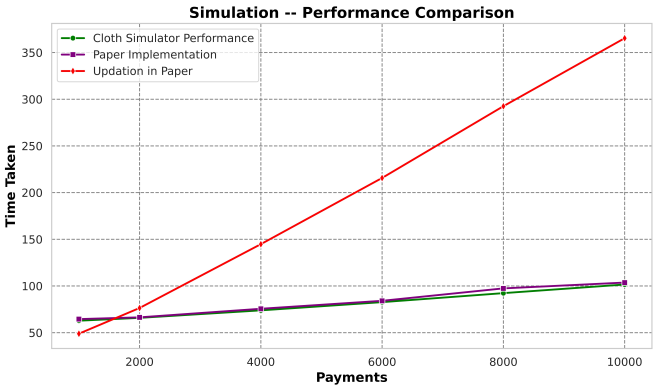

Fig. 16. Simulation on Payments

VIII. Conclusion

Throughout this study, we have meticulously maintained an array to track the ratio of defaulted payment attempts to the total number of forwarding opportunities for each node. This meticulous tracking aims to efficiently allocate nodes for forwarding payments, ultimately enhancing the payment delivery ratio.The graphical analyses presented above demonstrate that the implementation of MPCN, as outlined in the research paper, performs comparably to the existing method utilizing Dijkstra's algorithm. This similarity arises primarily due to the sole update in fee calculation, which does not significantly impact the simulation timing of Dijkstra's algorithm.

We have conducted a comprehensive analysis of both the paper implementation and the proposed method across various parameters, including the number of nodes and payments. The graphs depict that the proposed method exhibits slightly longer processing times compared to the other two approaches. This is attributed to the maintenance of an array storing default ratios for each node and the consequent frequent access to this array. In conclusion, while the proposed method may require slightly more processing time due to the additional array management, its potential to enhance the payment delivery ratio warrants further exploration and optimization.

REFERENCES

[1] 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.

[2] "Bitcoin Wiki: Bitcoin Scalability Faq." [Online]. Available: https://en.bitcoin.it/wiki/ScalabilityFAQ (accessed Dec. 29, 2021).

[3] "Bitcoin Wiki: Bitcoin Contract." [Online]. Available: https://en.bitcoin.it/wiki/Contract (accessed: Dec. 29, 2021).

[4] "Bitcoin Wiki: Block Size Limit Controversy." [Online]. Available: https://en.bitcoin.it/wiki/BlockSizeLimitControversy (accessed: Dec. 29, 2021).

[5] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, Flare: An Approach to Routing in Lightning Networks," Bitfury Group Limited, Amsterdam, The Netherlands, White Paper, 2016.

[6] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks," in Proc. IEEE Int. Conf. Comput. Commun. Netw., 2018, pp. 1–9.

[7] E. Rohrer, J.-F. Laß, and F. Tschorsch, "Towards a concurrent and distributed route selection for payment channel networks," in Data Privacy Management, Cryptocurrencies, and Blockchain Technology. Cham, Switzerland: Springer, 2017, pp. 411–419.

[8] Conoscenti, Marco. "Capabilities and Limitations of Payment Channel Networks for Blockchain Scalability." PhD diss., Polytechnic University of Turin, Italy, 2019.

[9] Conoscenti, Marco, Antonio Vetrò, and Juan Carlos De Martin. "Cloth: A lightning network simulator." SoftwareX 15 (2021): 100717.

[10] V. Sivaraman et al., "High throughput cryptocurrency routing in payment channel networks," in Proc. USENIX Symp. Netw. Syst. Design Implement., 2020, pp. 777–796.