IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 December 2022, accepted 30 December 2022, date of publication 10 January 2023, date of current version 13 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3235828

RESEARCH ARTICLE

LEAF: Let's Efficiently Make Adaptive Forwarding in Payment Channel Networks

XIAOFEI LUO AND PENG LI, (Senior Member, IEEE)
The University of Aizu, Aizuwakamatsu 965-8580, Japan
Corresponding author: Xiaofei Luo (d8202105@u-aizu.ac.jp)

ABSTRACT Blockchain technology is widely applicable to modern payment systems but has inherent throughput limitations. Off-chain networks are proposed to solve scalability issues, which allows parties to efficiently perform micropayments without committing all of the payments to the blockchain. Off-chain payments with security and privacy protection requirements use the smart contract to ensure security and reduce the risk of sensitive information leakage. Although off-chain payments avoid expensive on-chain operations, it raises many concerns, such as the capacity limitation of payment channels and highly dynamic channel status, lowering the throughput of payment channel networks (PCNs). In our work, we explore the path overlap issues in PCNs and propose a decentralized payment routing scheme to improve the network throughput and reduce the redundant traffic overhead of PCNs, thereby guaranteeing efficient payments. The simulation results indicate that the proposed routing algorithm can achieve higher throughput than other routing schemes while guaranteeing short payment times.

INDEX TERMS Blockchain, payment channel network, multi-paths routing, Markov approximate algorithm.

I. INTRODUCTION

Blockchain, a distributed ledger, becomes a promising proposal for building trust in decentralized environments [1]. It provides robust security and privacy protection for its participants. As one of the most popular blockchain applications, digital cryptocurrencies (such as Bitcoin [2] and Ether [3]) have emerged as an alternative payment way for modern payment systems. Cryptocurrencies maintain a fully decentralized ledger that updates by consensus protocols (such as proof-of-work [4], proof-of-stake [5]) of blockchains. The participants can make on-chain payments with other parties but it requires expensive on-chain payment fees. Additionally, on-chain payments are limited by the inherent throughput challenges, which are involved to block size and consensus of blockchain, e.g. in bitcoin consensus requires 10 minutes to confirm the payments in 1MB block [6]. To solve the scalability issues, payment channel networks (PCNs) are proposed as the scaling network for blockchain [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

In PCNs, payment channels are established between parties with a peer-to-peer connection and their deposits as the channel capacity. Payers use payment channels to transfer coins to their payees. Without a payment channel connected, off-chain payments between two parties require a routing path to forward. Payment channels on the routing path lock coins for payment occurring and are updated atomically to prevent the payer's funds from losing. Due to the limitation of the channel capacity, payment may fail when the payment amount exceeds the channel capacity. A failure message is then generated and sent back to the payer. Meanwhile, the failed payment gradually releases the funds already locked on payment channels among the routing path. It raises several problems like delayed payments and overhead on occupation and release of channel funds, which leads to lower throughput. Additionally, the payer needs to find another available routing path to resend the failed payment. The repeated path searching process and channel funds re-locking bring additional overhead, thereby decreasing the network performance.

To achieve higher throughput in PCNs, Sivaraman et al. [8] propose *Spider* which splits payments into payment units.

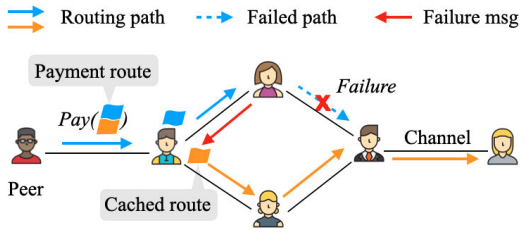


FIGURE 1. Illustrative example of our multi-branch routing system.

Spider routes payment units over multiple edge-disjoint paths and handles channel imbalance issues. However, this proposal delays payments since it needs to wait for the payments from the opposite direction. Wang et al. [9] split payments across multiple probed paths. The path selection depends on the probed information, which may be outdated before payments are issued. Furthermore, payment splitting imposes strict conditions on payment success, which requires all payment units to reach the destination resulting in long payment latency. In [10] and [11], the route construction also depends on the probe while the validity of time-sensitive probed information cannot be promised in highly dynamic PCNs. Bagaria et al. [12], [13] present redundant path schemes to route payments over multiple paths, but these schemes bring redundant resource occupation and require high collateral [14]. Existing works either delay payments or bring extra traffic loads. Moreover, they lack the reaction mechanism for payment failures.

In this paper, we explore the payment routing problem and propose a novel multi-path payment routing scheme, in which the payment route is constructed as a leaf-like structure. Payers prepare multiple available routing paths, including a primary path and several standby paths for their payments. Some special nodes on these paths, called fork nodes, cache the information of standby paths during the payment process. When a payment fails, the returned failure message can be prior handled by fork nodes to activate the cached standby path. The fork node forwards the payment across its cached path to the destination. Only a portion of locking funds requires to be released. Our scheme theoretically achieves a higher payment success ratio than the single-path routing, at the same time avoiding the redundancy of the multi-path routing. In highly dynamic PCNs, it's impractical for a payer to track the instantaneous capacity of each channel over the payment path. Hence, payers cannot either predict payment failure before their payments are issued, or track payment status during the payment process. The fork node cannot be specified immediately when the payment fails. Instead, a couple of nodes can be prior reserved as fork nodes, combined with corresponding standby paths to construct the pay packet.

To realize the proposed routing scheme, several challenges require to be overcome: 1) Privacy. The payment path is regarded as an important part of payment privacy to estimate the risk in [15] and [16]. The adversary can attack intermediate hops of a payment path to disturb the payment. The privacy protection issue should be considered to reduce the risk of privacy leakage. 2) Distribution. A PCN is a fully distributed network without centralized control. Every payer prefers to reserve standby paths as many as possible to prevent their payment from failing. But the capacity limitation of fork nodes leads to the overflow of standby path information. Therefore, the fork node selection is cortical to facilitate resource utilization. 3) Efficiency. As the number of standby paths increases, the probability of payment success theoretically increases. Payers prefer to select the path with a higher payment success probability as the primary path to reduce the opportunity of payment failure. But this path does not ensure a short payment time. There is a trade-off between the payment success ratio and payment time. We need to balance the two metrics to ensure payment efficiency.

Modern PCNs like Lightning Network (LN) [17] use an onion routing protocol to protect user privacy in payment routing. The Basis of Lightning Technology (BOLT) reveals the specification of LN [18]. We design a tailored onion routing to adapt our routing scheme. Standby path information is embedded into the onion packet as the branch of the primary path. Upon receiving a payment packet, the standby path information is cached on the fork node and waits for the payment failure message to activate. To implement an efficient system, we propose a distributed Markov chain-based path selection algorithm for our multi-path payment routing scheme. It requires payers to corporate with each other to achieve a relatively stable network state. Payers first collect a set of candidate paths to their destination. Each payer applies the proposed scheme to pick a set of paths consisting of the primary path and the standby paths as the current routing path configuration.

Our goal is efficient utilization of the channel capacity to increase the throughput of PCNs. We list our contributions below:

- We reveal the path overlapping phenomenon in the off-chain payment process and analyze the concerns of routing schemes in current research.
- We first propose a novel multi-path payment routing mechanism, which allows senders to prepare several standby paths for payment routing to achieve a higher payment success ratio and a stable latency.
- We implement a distributed Markov approximation algorithm for efficient routing and develop a simulator of LN to simulate the payment routing process in the network layer.

The organization of this paper is shown as follows. We first present path overlap issues and elaborate on our motivation in Section II. In Section III, we give an overview of the system design and describe our system model. Section IV details the Markov chain-based approximate algorithm for path selection. The system implementation is described in Section V. We conduct experiments in Section VI to evaluate the performance of the proposed scheme. Section VII concludes this paper.

II. MOTIVATION

A payment channel network is an off-chain distributed network consisting of peers and payment channels connecting them. Without the participation of on-chain miners, the implementation of off-chain payments depends on payment channels, thereby avoiding expensive on-chain operations. In general, payments between two peers indirectly connected by payment channels require multi-hop transmission. To guarantee the atomicity of payment, a contract called Hash Time-Lock Contract (HTLC) is proposed. Contracts are established on payment channels along the payment path, at the same time partial channel funds are locked for associate payments. In a highly dynamic PCN, concurrent payments will lead to payment failure due to channel funds being unavoidably competed during the shared payment channels. In this case, all of the established contracts will be canceled. Thus, the sender has to search a new available path to achieve successful payments. This brings overhead on pathfinding and contract re-establishment over overlap channels. Additionally, the prior released channel capacity can be preempted by other payments.

In current implementations of PCNs, almost systems use source routing mechanisms. To further increase the throughput and payment opportunity of PCNs, many researchers focus on the routing method and congestion control algorithm. Such proposals like Atomic Multi-Path Payments (AMP) design a routing mechanism to transfer payments through multiple paths [12], [13]. But it requires high collateral [14] that more coins would be locked in channels as 'in-flight' coins. The in-flight coins cannot be used by other payments leading to reduced channel capacity. A special multi-path routing scheme called spider was proposed in [8], which splits payments into several units to scatter the payload on a single path. However, the success of payment depends on the final completed unit which induces higher payment latency.

As one of the emerging PCNs, LN deploys a developed Dijkstra [19] algorithm for pathfinding in a distributed network. The Dijkstra algorithm employs beacon nodes as flags of path planning. Neighbors of each beacon node broadcast their shortest path to the beacon node to their neighbors. The shortest path information is gradually spread outward from beacon nodes, realizing the path discovery between any pair of nodes. Considering the case of payment failures, senders actually route their payments by constantly trying to use different available paths found by the pathfinding algorithm. These paths may overlap, resulting in repeated transfers of failed payments through overlapping channels. To verify the conjecture, we explore the current LN topology and sample a set of short paths between the specified payer-payee pair. In Figure 2, we show the overlap ratio of paths under different numbers of available paths within different network environments. The ratio is calculated by the count of overlapping channels and the total number of channels in the path set. LN core/edge denotes the cropped topology close to LN's core/edge network. Since each payer within the core network has numerous channels, the path overlap ratio is lower relatively than the payer at the edge. Insufficient channel balance causes payment failure, which leads to frequent locking and release of resources on overlapping channels. It brings communication costs on updating channel state and rapid changes in channel balance to affect subsequent payments using the channel.

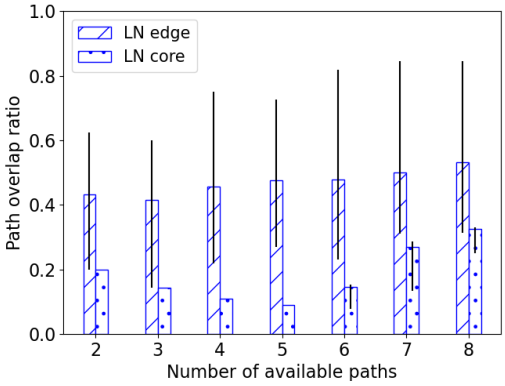


FIGURE 2. Change in the path overlap ratio under different numbers of available paths within different positions of LN.

We attempt to propose a novel routing mechanism to prevent the payment from failing as well as solve the path overlap issue. Inspired by the restoration approaches in [20] and [21], we try to apply path restoration to the payment process so that payments can react to the failure caused by insufficient channel capacity. When a payment is failed at an intermediate hop, 1) the original routing path is discarded, 2) a standby path is activated, and 3) the payment proceeds along the standby path. However, PCNs like LN apply source-routing in the payment process, which gives senders full control over their payment path within the network [18]. The payload needs to be packaged before payment is issued. In addition, the sender cannot track the status of their issued payments in real-time nor update the routing strategy for the issued payment. Therefore, the payment requires an adaptive adjustment strategy to cope with changes in the link state. A mitigation approach is to prepare a set of standby paths in advance. The information of standby paths cached in intermediate users waiting for downstream payment failure message to activate.

III. SYSTEM OVERVIEW AND MODEL

A. SYSTEM OVERVIEW

In this section, we make an overview of the design of the multi-path routing mechanism. As depicted in Figure 3, the payload of payment is initialized at sender S and sent to destination R. Different from the original payment routing mechanism to find the shortest path in LN, sender S collects a set of candidate paths to route payments. To eliminate the redundant communication cost of the overlapping channels, the selected paths should be pre-processed to build an integrated route. It can be represented figuratively as the construct of a leaf composed of a sender as the petiole, a receiver as the leaf apex, the primary path as the mid-vein, and standby

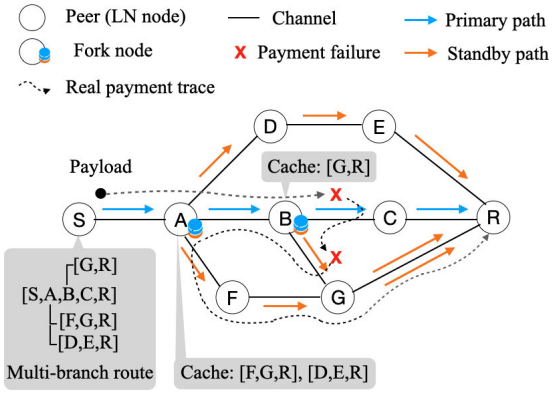


FIGURE 3. System overview. *S* and *R* are the payer and the payee, responsively. *S* generates the multi-branch onion route for its payment. The standby path information can be cached by fork nodes (like *A*, *B*) on the route.

paths as secondary veins leading to the leaf apex. The sender applies the multi-path routing mechanism to route payments to the destination, which can be treated as the transportation of nutrients from the petiole to the apex in leaves.

In our design, a fork node is a bifurcation point of two independent paths with overlapping channels. A payment route with multiple branches has multiple fork nodes. Senders first transfer their payments along the primary path. Fork nodes forward the payment along its current path and cache the corresponding information of its standby paths. For the example in Figure 3, sender *S* forwards the payload of the payment to the fork node *A*. *A* receives the payload and forwards it to the next peer *B* along the primary path. The information of the standby path is cached in fork node *A*. When payment failure occurs, instead of immediately aborting the payment, we take the failure message as a signal to activate the upstream nearest standby path. The fork node receives the signal and removes the primary path information of the corresponding payment. The cached standby path information is used to rebuild a new payment path. Then, the fork node forwards the payment to the destination via the new path. The details of implementation are described in Section V.

B. ASSUMPTIONS

We consider each channel in the PCN has a static capacity. And there are some concurrent payments with different senders, destinations, and payment amounts. Due to the privacy protection in PCNs, each sender has no knowledge about the route information of the payments launched by other senders. Additionally, we assume that the set of candidate paths is given in which each path has at least one other path overlapping with it. Perhaps a routing table can save the path-finding time for senders. However, it's unfeasible for a sender to build a large routing table to reach all nodes of the entire network. In this paper, we assume senders can efficiently collect a set of available short paths to use and leave the path-finding algorithm as an interesting direction but an orthogonal problem.

C. PROBLEM DEFINITION

As the participants of a PCN, senders prefer to reserve more standby paths to achieve a higher probability of successful payment. However, the decision of each sender mutually interacts with each other. Each sender needs to measure some factors to make their decisions. We characterize these factors in the following.

Path Ordering: The multi-branch routing mechanism allows senders to prepare several available paths to route their payments. The ordering of those paths is crucial to building the multi-branch route. Firstly, it directly determines the primary path. After that, the sender can identify the fork nodes on the primary path by intersecting the primary path with each standby path. Secondly, it can be used for fork nodes to activate the cached standby paths sequentially. We notice that the fork node may capture multiple standby paths for a single payment. For the example in Figure 3, fork node *A* captures two standby paths and caches them in the memory. The ordering of the standby paths is associated with the path index in the set, like $1 : [F, G, R], 2 : [D, E, R]$. The cache pops the standby path with the small index first. Hence, the payload first chooses the standby path $[F, G, R]$ to reach the destination *R*. In some cases, the bifurcation point of the two alternate paths may exist independently of the primary path. Furthermore, each path carries different probabilities of successful payments, which can be used for path ordering.

Latency: Payment latency can be regarded as a period of time from the moment that the sender sends out a payment to the moment that the payment result returns back to the sender. In practice, payers prefer their payments to be completed quickly to prevent the payment from expiring. The fast completion of payments frees up the resources locked in the contract to be used by other payments. It makes the resource utilization of the entire network more efficient. Payment latency can also be a factor to neutralize the impact caused by the probability of payment success. For example, when a payment is through across a set of payment channels with a large capacity, the probability of successful payment is higher, but it brings high latency. In contrast, a short path with low channel capacity leads to a low success ratio.

Fees: The Fees mainly represent the total forwarding fee charged by intermediate users. In our design, we take the forwarding fee corresponding to the longest path in the path set used to create the multi-branch route as the total forwarding fee. The explanation is described in Section V. It constraints the routing mechanism to choose the path with a shorter length and the user over the path with a lower forwarding fee requirement. We find that the impact of fees on path selection is similar to that of payment latency. Low latency often indicates fewer hops on a routing path. In addition, payment latency is a critical metric of the payment system, but fees can be negligible since off-chain payment fees are exceptionally low [17]. Hence, we merge the cost metrics and only use payment latency in our system model.

Amount: The payment amount determines whether the selected path is available at the beginning of path selection. Due to the distributed PCN having strong privacy protection, each sender has no knowledge about the payment amount of others to make a global routing path planning. Different from the communication networks, the growing number of payments in the forward direction leads to an increased channel deposit in the reversed direction. In general, payments with large amounts occupy more channel capacity, thereby decreasing the probability of successful payment. However, senders can only get a snapshot of the instantaneous state of a payment channel but cannot track the deposit status of the bidirectional channel along the path in real-time. Rapid changes in channel deposit result in a highly dynamic network that invalidates some available paths. The impact of the payment amount on the payment process is unpredictable but can be reflected in the probability of payment success.

D. SYSTEM MODEL

To simplify the exploration, we consider a PCN can be modeled as a graph $G(V, E)$, where there are N lightning nodes in the node-set V and a set of established channels E between them. There are a set of payment sessions S in G. Each payment session s ($s \in S$) has different pair of payer-payee. The network topology of our model is a subset of the real LN. Each sender of session s collects a set of candidate paths, denoted as P_s. The length of a single path p is the value of $|p|$. Senders choose a set of paths from P_s as reserved paths R_s ($R_s \subseteq P_s$) to route their payments. The reserved paths consists of one primary path and $|R_s| - 1$ standby paths, $|R_s| \geq 2$.

The target of the Multi-Branch Routing (MBR) problem is to find optimal paths set for each payment session in our system. We define a binary variable x_s^p to denote whether the path p ($p \in P_s$) is selected as a reserved path. It can be denoted as:

$$x_s^p = \begin{cases} 1, & \text{if the candidate path } p \text{ is selected by} \\ & \text{session } s \text{ as a reserved path.} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Then, we define an integer variable y_s^p to denote the ordering of the selected paths in R_s, $x_s^p \leq y_s^p \leq \sum_p x_s^p = |R_s|$. The following constraint can be obtained:

$$(x_s^p - 1)y_s^p = 0, \quad p \in R_s. \tag{2}$$

Furthermore, y_s^p affects the number of overlapping channels between two reserved paths.

The onion-routing protocol of LN specifies the maximum number of hops allowed in an onion packet. We use Δ to denote this upper bound number of hops. The function $H(|p|, y_s^p)$ is used to calculate the hops of a reversed path in a multi-branch route. It equals the difference between $|p|$ and the number of overlapping channels on the reversed path p. The condition of hop limitation can be expressed as:

$$\sum_{p \in R_s} x_s^p \cdot H(|p|, y_s^p) \leq \Delta. \tag{3}$$

Excessive cached path information on a fork node reduces the efficiency of path switching. Hence, each fork node i on path p maintains a buffer with a limited size to store information on standby paths:

$$\sum_{s \in S} \sum_{p \in R_s, i \in p} x_s^p \leq B_i. \tag{4}$$

Each sender in G enforces the multi-branch routing mechanism to improve the system performance, which is measured by the following objectives:

1). *The maximum probability of successful payments*. For a candidate path of payment session s, the probability is related to the payment amount z_s and the order of this path y_s^p, denoted as $\pi(z_s, y_s^p)$. The payment success probability of session s can be represented as:

$$\pi_s = \sum_{p \in R_s} \mathbb{E}(x_s^p \cdot \pi_s^p(z_s, y_s^p)). \tag{5}$$

2). *The minimal payment latency*. We use l_s^p to denote the payment latency of a payment session s on path p. Similarly, the payment latency of session s can be expressed as:

$$l_s = \sum_{p \in R_s} \mathbb{E}(x_s^p \cdot l_s^p(y_s^p)). \tag{6}$$

Consequently, the objective function of overall weighted system utility can be described as:

$$\phi = \max \sum_{s \in S} (\pi_s - \alpha ||l_s||), \tag{7}$$

where α is a positive weighted coefficient to balance the payment success ratio and the payment time, the $||\cdot||$ is a normalization function.

IV. DISTRIBUTED MULTI-BRANCH ROUTING ALGORITHM

The path selection problem with resource limitation is NP-complete [22], [23]. Since the PCN with privacy protection is fully distributed, it's impractical for users to share the sensitive path information to make a centralized optimization for routing path planning. In this section, we provide a decentralized algorithm to handle the path selection problem. Our proposal augments the success opportunity for payments, which reduces the risk of payment failure to achieve high throughput for PCNs.

A. LOG-SUM-EXP APPROXIMATION

Let $\mathcal{K}$ be a set of all feasible configurations for the MBR problem. A payer can obtain the local performance $\phi_s(k)$ of his payment session s under a given solution k. Then, the system objective function can be computed by aggregating the performance of each payment session: $\sum_s \phi_s(k)$, $(s \in S)$. We use π_k to denote the percentage of time that the available

solution k is in use. By adopting the approximation approach proposed in [24], our MBR problem can be approximated as:

$$\begin{aligned} \max \ & \sum_{k\in\mathcal{K}} \pi_k \sum_{s\in S} \phi_s(k) - \frac{1}{\beta} \sum_{k\in\mathcal{K}} \pi_k \log \pi_k \\ \text{s.t.} \ & \sum_{k\in\mathcal{K}} \pi_k = 1, \end{aligned} \tag{8}$$

where β is a positive constant. The approximation approach introduces an entropy term $-\frac{1}{\beta}\sum_{k\in\mathcal{K}} \pi_k \log \pi_k$ with an enhance approximation gap bounded by $\frac{1}{\beta}\log \mathcal{K}$. As the value of β increases, the approximated function of our MBR problem becomes more exact. We use $\pi_k^*, k \in \mathcal{K}$ to represent the optimal solution of the approximated function, which can be derived via solving the Karush-Kuhn-Tucker (KKT) conditions [25] and expressed as:

$$\pi_k^* = \frac{\exp\left(\beta \sum_{s\in S} \phi_s(k)\right)}{\sum_{k'\in\mathcal{K}} \exp\left(\beta \sum_{s\in S} \phi_s(k')\right)}, \quad \forall k \in \mathcal{K}. \tag{9}$$

Then, the MBR problem can be approximately solved through a time-sharing manner among different configurations based on π_k^*.

B. MARKOV CHAIN DESIGN

We construct a time-reversible Markov Chain (MC) on which a single state is an available configuration within the state space, and the stationary distribution is $\pi_k^*, k \in \mathcal{K}$. The transition between the two states is to replace a reserved path or reorder the reserved paths for any payment session. The best solution to achieve a near-optimal performance is to train the transitions to converge to the stationary distribution π_k^*. To describe the transition process intuitively, we use a non-negative value $q_{k,k'}$ to denote the transition rate between the two configurations k and k' and set it to zero, unless the two configurations satisfy the two conditions: 1). $|k \cup k'| - |k \cap k'| = 2$. 2). $k \cup k' - k \cap k' \in P_{\hat{s}}$. where $\hat{s}$ is the involving payment session to make the path swapping or ordering. Besides, the Markov chain has to guarantee that any two states can be reachable mutually and the detailed balance equation $\pi_k^* q_{k,k'} = \pi_{k'}^* q_{k',k}, \forall k, k' \in \mathcal{K}$ needs to be satisfied.

For the two direct-connect configurations (k, k'), we let the transition rate $q_{k,k'}$ and the difference in system performance be positively correlated. From [24] and [26], the transition rate can be expressed as:

$$\begin{cases} q_{k,k'} = \omega \exp\left(\frac{1}{2}\beta \sum_{s\in S} (\phi_s(k) - \phi_s(k'))\right) \\ q_{k',k} = \omega \exp\left(\frac{1}{2}\beta \sum_{s\in S} (\phi_s(k') - \phi_s(k))\right). \end{cases} \tag{10}$$

where ω is a positive constant. We can find that transition rates $q_{k,k'}$ and $q_{k',k}$ are symmetric. If the system performance is improved under the configuration k', the performance gap will be positive, increasing the probability of jumping to this configuration, and vice versa.

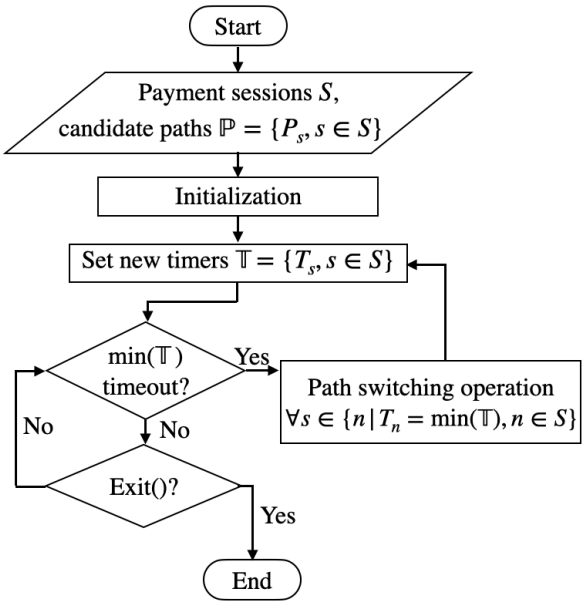


FIGURE 4. Flowchart of the proposed distributed Markov chain-based routing algorithm.

C. DISTRIBUTED MARKOV CHAIN BASED ROUTING SCHEME

We show the flowchart of the proposed distributed Markov chain-based routing algorithm in Figure 4. The detailed implementation of our algorithm is shown in Algorithm 1. Each payment session launches a processing thread on the corresponding end-host of its payer. To guarantee algorithm convergence in a distributed system, each payer needs to share the local performance with other payers in the system. Furthermore, the configuration of path selection is sensitive information involving the payer's privacy. Each payer cannot collect it from other payers to construct the current configuration of the system.

From the assumption of the Markov approximation in [24], the performance of the configuration requires to be prior computed by each payer. As aforementioned, an important metric to evaluate the system performance is the real payment time which is unpredictable for each payment with a different amount in fast time-varying PCN. An obverse condition for the payment time is no longer than the expiration of the time-lock of the first hop. The expectation of payment time becomes longer as the number of hops augments because of the extra transmission time. In our algorithm, we assume that the payment time is only correlated to the transmission time of each hop on the route. Another metric is the payment success ratio which reflects the payment success probability and increases theoretically with the growing number of payment paths. For example, if we get two available routing paths for payment and assume that the payment success probability of two paths is π_1 and π_2, $\pi_1, \pi_2 \in [0, 1]$, the success probability of multi-path payment can be expressed to $1 - \pi_1\pi_2$, which is bigger than either of the

Algorithm 1 Online Distributed MC-Based Routing Algorithm

for each $s \in S$ **do**
  execute `Initialization()`
  execute `Set-timer(s)`
end for
while system is still running **do**
  /*`Listen to State-Transit`*/
  if T_s expires **then**
    switch operation **do**
      case x
        $x_s^p \leftarrow 0$
        $x_s^{p'} \leftarrow 1$
      case y
        $y_s^p \leftrightarrow y_s^{p'}$
    execute `Set-timer(s)`
    broadcast a RESET($\phi_s(k')$) signal with local performance $\phi_s(k)$ to other payers
  end if
  /*`Listen to RESET Signals`*/
  if a payer receives the RESET($\phi_s(k')$) signal **then**
    $\phi_s(k) \leftarrow \phi_s(k')$
    refresh and reset the timer T_s
  end if
end while

two single paths. However, the payment success probability is corresponding to the payment amount and other network parameters. The current LN protocol provides an estimation method to estimate the success probability of payment on the payment path. In this manner, we can estimate the system's performance.

The algorithm is executed on each distributed payer. These payers cooperate with each other to obtain the current system performance of the entire network. A detailed description of our algorithm is shown as follows.

- `Initialization()`: The payer launches a processing thread for his payment sessions. Each payment session randomly chooses several independent paths as reserved paths and then shuffles the ordering of the selected paths.
- `Set-timer()`: For each payment session $s \in S$, the corresponding payer first randomly selects a reserved path p from R_s, and then selects another path $p' \neq p$ from P_s. If the path p' is in the reserved path set R_s, the operation of the payer is to update y by swapping the ordering of the two paths. If the selected path p' is not in the reserved paths, the operation of the payer is to update x by swapping the old reserved path to a new path. The system performance can be computed by local performance and pre-collected performance information from other payers. By estimating the performance under the new configuration k', the payer can trigger a timer T_s with an exponentially distribution for the corresponding payment session s with mean value: $\omega \exp(\frac{1}{2}\beta \sum_{i\in S}(\phi_i(k) - \phi_i(k'))) \cdot (|R_s| \cdot (|P_s| - 1))^{-1}$. The payer then broadcasts the RESET($\phi_s(k')$) signal carrying the local performance $\phi_s(k')$ of payment session s to other payers for further system performance computation.
- `State-Transit signal`: If a timer expires, the corresponding payer does the operation: (x). swapping the selected reserved path p with the unreserved path p'. (y). swapping the ordering of reserved path p with another reserved path p'.
- `RESET signal`: When a RESET signal is received by a payment session s, the corresponding payer refreshes the timers of his payment sessions invoking the expression (11).

The convergence of the proposed algorithm can be proved according to the algorithm analysis in [24] and [26].

Algorithm 2 `Initialization()`

Input: a payment session $s \in S$, candidate paths P_s
Output: R_s

launches a processing thread for s on the corresponding payer
$R_s \leftarrow$ randomly chooses several independent paths from P_s
shuffles the ordering of the reserved path R_s randomly

Algorithm 3 `Set-timer()`

Input: a payment session $s \in S$
Output: T_s,operation, p, p'

$p \leftarrow$ randomly chooses a reserved path from R_s
$p' \leftarrow$ randomly chooses a path from $P_s \setminus p$
if $p' \in R_s$ **then**
  operation $\leftarrow y$
else if $p' \in P_s \setminus R_s$ **then**
  operation $\leftarrow x$
end if
measures current system performance $\sum_{i\in S\setminus s} \phi_i(k) + \phi_s(k)$ with the collected performance information and local performance.
estimates the system performance $\sum_{i\in S} \phi_i(k')$ under the target configuration that swaps p with p'
generates a new exponentially distributed timer T_s for the payment session s with mean value as:

$$\frac{\omega \exp(\frac{1}{2}\beta \sum_{i\in S}(\phi_i(k) - \phi_i(k')))}{|R_s| \cdot (|P_s| - 1)} \tag{11}$$

V. SYSTEM IMPLEMENTATION

In this section, we first elaborate on the design of the multi-branch onion route in our system and then describe the details for the implementation of the payment routine.

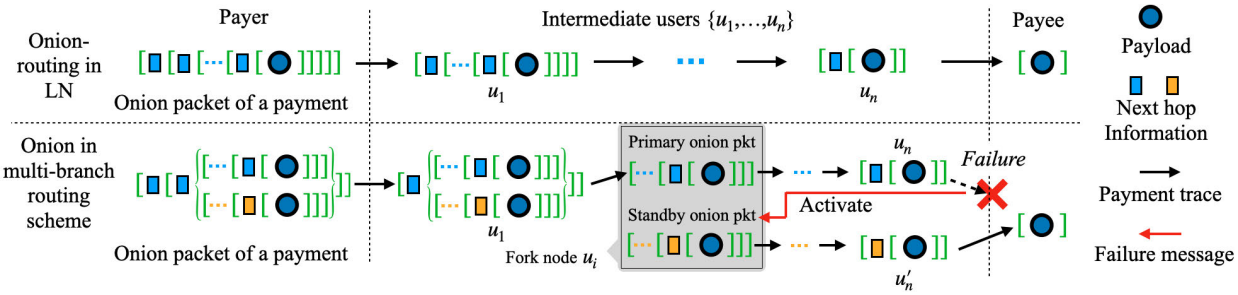


FIGURE 5. Difference of the onion-routing between our multi-branch routing mechanism and LN.

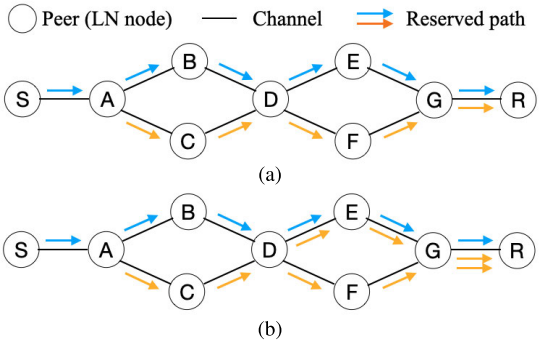


FIGURE 6. Two special cases in onion-routing.

A. ONION ROUTING

To protect the security and privacy of payments, the LN applies the onion routing protocol to payment execution. The source-routing mechanism allows payment senders to complete control of the payment path within LN. It highly adapts LN, that senders can customize some conditions (such as ignored peers, maximum fee, and total worst-case time-lock period) to query a satisfied routing path. The quarried path information can be decomposed into the instructions of each hop as per-hop payload encoding into the onion route, as illustrated in Figure 5. The payment is successful when the final payload reaches the payee. In our multi-branch routing scheme, we employ the onion routing protocol to guarantee the security and privacy features of LN. The onion packet is encapsulated with a multi-core structure, which only has a single core within the LN. The information on standby paths is stored in each sub-onion packet. For the example in Figure 5, the information of the standby path from fork node u_i to *payee*, is encapsulated into a sub-onion packet, combined with the rest information of the primary path as the payload of i-th hop. Each node opens up a buffer with a certain size for packet caching within the proposed system.

1) POLICY UPDATE

There are two pieces of information in the payload required to be updated against the change of the onion route: 1) *Forwarding fee*. In basic LN, peers use a gossip protocol to probe the existence of payment channels with the public charging fee [27]. The total forwarding fee is a cumulative fee that the sender pays to intermediate users for payment forwarding. Due to the length difference of each reserved path, the total forwarding fees over each path are uneven. In our scheme, we recommend the maximum forwarding fee among the reserved paths as the total forwarding fee of a multi-branch route. A payment carries its forwarding fee through a reserved path with a short length incurs overflow forwarding fees. The overflow fee is used to pay each fork node for caching the standby path information. 2) *Time lock*. The time lock is the expiry time for a payment to lock required coins on each channel over its path. The value of the time lock is gradually decreasing along the payment path [28]. For example, a payment sets a time lock on hop i, denoted as t_i. The time lock of previous hop $i-1$ is represented as $t_{i-1} = t_i + \Delta$, where Δ is a positive value. In our implementation, we assume a multi-branch route has several branches starting at hop i. The time lock of previous hop $i-1$ is $t_{i-1} = \sum_{i \in p, p \in R}(t_i^p) + \Delta$, where p are relevant paths gathering at hop $i-1$ in reserved path set R.

2) STABILITY ANALYSIS

Although the sender determines the ordering of reserved paths, the fork node has full control over the forwarding order of cached payloads in actual execution. If a fork node forwards a payment along its longest path, it can not receive the redundant fee for holding the standby payloads of this payment. From the perspective of a fork node, forwarding payments over a shorter path has the opportunity to get higher forwarding fees. On the other hand, the time lock limits the utilization of the channel funds. For a single payment, fork nodes prefer to forward the payload with a small time lock to efficiently utilize their funds on connected channels. These potential factors may reduce the utility of sender decisions, causing system instability. In general, senders can finalize the actual payment routing path when their payment is settled. A sender can detect whether the fork node is greedy based on the path statistics of successful payments, thereby implementing some countermeasures such as adding them to a block list [29].

3) MULTI-BRANCH ONION PACKET

To build the multi-branch onion packet, the sender first prepares reserved paths and prunes their overlapping hops.

Pay:

Initialize: a payment attempt as Tx with N candidate paths.

1: Select a set of n reserved paths $R = [R_0, R_1, ..., R_n]$ for Tx from candidate path set, $n \leq N$.
2: Get the set of m fork nodes $B = [b_1, ..., b_m]$ from $R, m \leq M$.
3: Clip the overlap channels to build a multi-branch route R' with H payment channels.
4: Specify a token v_i for each fork node $b_i, \forall b_i \in B$ and obtain a sequence of token set $V = [v_1, ..., v_m]$.
5: Assign the Time-Lock Delta (TLV) Δt_h to each channel $h, \forall h \in H$ and get $\Delta T = [\Delta t_1, ..., \Delta t_H]$. For a fork node b_i with an upstream channel h and downstream channels u, w on R', $\Delta t_h > \Delta t_u + \Delta t_w$. Otherwise, $\Delta t_h > \Delta t_{h+1}$.
6: Find the path with max fees as the total forwarding fee ($fees$) that requires to be prepared. $fees$ are precisely allocated to each hop. The overflow fees will be paid to fork nodes on the route
7: Compute cryptographic keys to generate the session key and construct the onion packet Tx$(R', B, V, \Delta T, fees)$.

Send: the packet to the first hop.

FIGURE 7. Payment initialization process in the pay routine of our multi-branch routing scheme.

Decision:

Receive Tx packet or downstream message Msg.

Decision(Tx)

1: **if** node i in B **then**
2: Hold standby onion packet O_i
3: **end if**
4: Forward Tx to $(i+1)^{th}$ node

Decision(Msg)

1: **if** node i in B **then**
2: **if** Msg is *fulfill* or *expiry* **then**
3: Del standby onion packet O_i
4: **else if** Msg is *fail* and token $v == v_i$ **then**
5: Activate standby onion packet O_i
6: Del Msg
7: **end if**
8: **else**
9: Forward Msg to $(i-1)^{th}$ node
10: **end if**

FIGURE 8. Decision making on the intermediate user when an upstream Tx packet or downstream Msg is received.

Reserved path preparation is done by the path-selecting algorithm. Hop information in the packet can be obtained by pruning overlapping hops and orderly merging the rest hops in reserved paths. Each standby path should be a complete and continuous path from the fork node to the destination. There are two special cases in overlapping hops handling: a) Multiple overlapping segments on the path. As demonstrated in Figure 6a, the primary path (blue arrow) overlaps with the standby path (orange arrow) at hop {[S, A], [G, R]} and node-D. In this case, the fork node is node-A, and the standby path is [C, D, F, G, R]. b) Forks on standby paths. The two standby paths overlap at hop {[S, A], [A, C], [C, D], [G, R]} as shown in Figure 6b. Here, the fork nodes are nodes [A, D], the standby paths are {[C, D, E, G, R], [F, G, R]}.

B. CONSTRUCTION DETAILS

In this part, we describe the details of the operations in our routing mechanism.

BOLT#04 describes the detailed onion routing protocol within the LN including onion packet construction and forwarding. We first update the payment initialization process as shown in Figure 7. Lines 1-3 describe the path pre-processing in onion-routed packet generation. The sender needs to prepare tokens for failed payment to extract the standby path on each fork node in line 4. Lines 5-6 elaborate on the time-lock and fee initialization. According to the protocol, an ephemeral cryptographic key should be computed for each hop and gathered by the sender to generate the payment session key. It is iteratively computed from the sender, independent of the entire path. Hence, it's possible for the sender to compute cryptographic keys for each standby path and encapsulate them into the multi-branch onion packet. The encapsulated multi-branch onion packet is then sent to the first hop on the primary path.

We modify the forwarding logic of intermediate users as shown in Figure 8. Upon receiving an onion packet, Intermediate users parse it to get the corresponding hop information inside the onion packet. Except for route verification and fee-charging, the intermediate users need to determine whether they are identified as fork nodes. As shown in *Decision* (Tx), if a user is a fork node, this user will hold relevant standby path information. Then, the inside onion packet will be forwarded along the current path. A message (Msg) carrying the payment result is initialized and sent back to the payment sender when the payment fails or is fulfilled. A fork node receives the message and verifies whether the payment is fulfilled or expired. If so, the node deletes the cached standby path information of this payment. Otherwise, the node activates the standby path if the token carried by the failure message is valid. Due to the message being transferred in the opposite direction of payment routing, the fork node closer to the message initialization node will be activated first.

VI. PERFORMANCE EVALUATION

To evaluate our multi-branch routing mechanism, we first develop a simulator for PCNs. The details of our simulation are described in VI-A. We then present the experimental

results under different settings and make a comparison with other routing schemes.

A. SETTINGS

1) SIMULATOR

We develop a Python-based simulator to model a lightning network. The simulator is constructed with basic payment modules that can accurately simulate the payment process. Each node (peer) maintains a forwarding queue in which queues received payment packets (payloads) to corresponding channels. The payment initialization module allows nodes to instantiate payment objects and build routed messages as payloads. Payers send payments along the payment path generated by the specified routing scheme. Each node can observe the topology of the entire network to enable the routing scheme to prepare suitable routes. We set up a buffer for each node to cache the standby path information, enabling the simulator to adapt to our multi-branch routing scheme.

The bi-direction channel carries certain funds deposited by the connected peers. It delivers payments and updates its balance by shifting the balance to the side of the downstream node along the payment path. Payments in a channel consume the channel funds and lock the funds as *in-flight* to avoid the occupation by others until their results (settlement/failure) are received. After the payee receives the payment packet, it registers for payment and sends a settlement message to the payer. Payment failure occurs when a channel over the payment path has insufficient balance. A failure message is then generated and sent along the reverse path, extending with a field that carries a computed token to activate the cached standby paths.

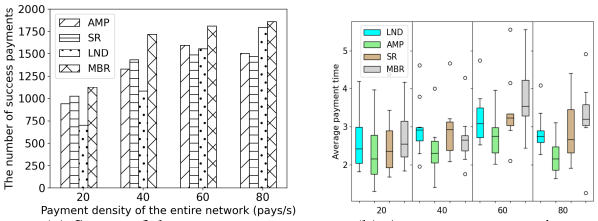
2) BENCHMARKS

We implement four routing schemes proposed within LN into our simulator for performance evaluation.

MBR: Multi-Branch Routing scheme picks up to k paths with overlapping channels as candidate paths. A payer can choose multiple paths from the candidate path set to route a payment to the payee. Except for the primary path, the information of standby paths will be cached on fork nodes waiting for the activation by the failure message.

LND: The routing scheme implemented in the current Lightning Network Daemon (LND) allows payers to find an available shortest path to route their payments. Once a payment fails at a channel with insufficient balance, the payer updates the local observation to ignore that channel during the pathfinding process. The ignored channels will be reconsidered in pathfinding after 5 seconds.

MPR: Multi-Path Routing scheme randomly chooses a set of n available paths for payers to route their payments to associate payees within the payment network. The payer first initializes a payment session containing n payment instances with a uniform payment hash and then assigns different paths to these payment instances. If any instance of this payment session reaches the payee, the payment is successful.



(a) Successful payments. (b) Average payment time.

FIGURE 9. Comparison of different scheduling methods under different payment density conditions.

The follow-up arriving instances with the same payment hash will be aborted as this hash is already recorded in the payment database.

SR: Split Routing [9] also employs multiple paths to route payments between each pair of payer-payee. Different from other multi-path routing schemes, *SR* splits a payment to payment units and sends them out across a set of candidate paths. Each payer collects candidate paths and records the bottleneck capacity of each candidate path to allocate the payment units. By dynamically adjusting the path selection, *SR* places payment units to relevant paths with lower payment fees. When all payment units arrive at the payee, the payer gets a successful payment.

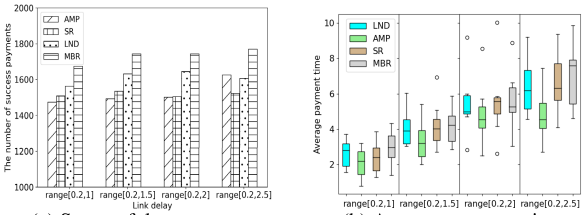
3) TOPOLOGY AND PAYMENTS

We clip a small-scale network topology from the edge of the LN main network with 25 LN nodes and 33 payment channels. We select 10 pairs of payer-payee to simulate the payment process. The payment workloads with Poisson distribution are randomly generated by a procedure in the simulator. The amount of each payment is normally distributed in the range [2, 7] (USD) according to the payment size distribution in [9]. The initial balance and delay of a channel are normally distributed in the range [80, 160] (USD) and range $[0.2s, 1s]$, respectively. Without the re-funding operation, the capacity of each channel is fixed. The expiry of each payment is set to 30s. The buffer size of each node is 100, which means each node can cache 100 standby path information within the network applying the proposed routing scheme. We sample available routing paths in the network to construct the candidate path set before starting our simulation. The candidate paths in our experiment have overlapping edges. The weighted coefficient is set to 0.5.

B. PERFORMANCE UNDER DIFFERENT NETWORK PARAMETER

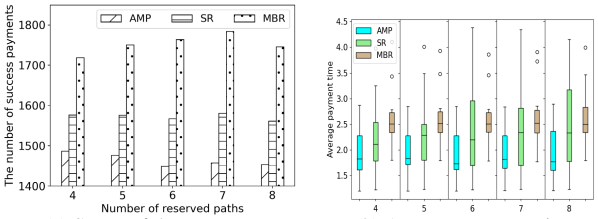
We deploy the five routing schemes above into our simulator and show their performance under different settings.

We use new payments initiated per second to denote the payment density (PD) of a PCN with 20 *pays/s* as default. The delay of each channel (link delay) to route payment packets is a constant, which samples from the range $[0.2s, 1s]$. We show the change in the number of successful payments and average payment time within 200 seconds under different payment



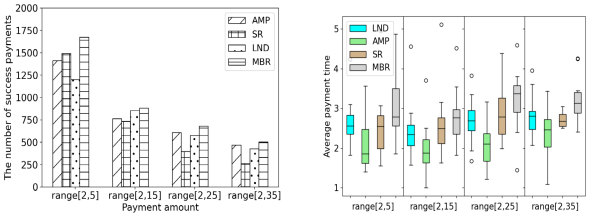
(a) Successful payments. (b) Average payment time.

FIGURE 10. Comparison of different scheduling methods under different link delay conditions.



(a) Successful payments. (b) Average payment time.

FIGURE 11. Comparison of different scheduling methods under different numbers of reserved paths.



(a) Successful payments. (b) Average payment time.

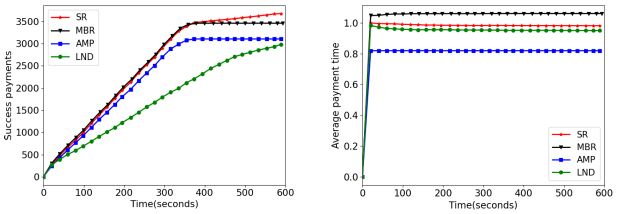
FIGURE 12. Comparison of different scheduling methods under the different payment amounts (USD).

densities in Figure 9. Notice that our method can achieve a higher success ratio compared with other methods as shown in Figure 9a. As payment density increases, interactions between concurrent payments on shared payment channels are more frequent, especially for multi-path routing schemes with many redundant payments. Hence, the payment success ratio of *AMP* and *SR* is lower than the other two methods when the payment density is higher. Without re-balancing and re-funding for channels, unidirectional capital flows skewing channel balance is unsustainable in PCNs. Channel balance limits the maximum number of successful payments. Therefore, under the current configuration, the total number of successful payments is limited to 2K. In Figure 9b, the average payment time of each method is relatively close when the payment density is lower than 40 *pays/s*. When we improve the payment density, the proposed method has a slightly higher payment time than other methods.

The link delay has an impact on the system's responsibility as well as affects the payment process. We show the performance of each routing scheme under different network connections with the link delay as per range {[0.2s, 1s], [0.2s, 1.5s], [0.2s, 2s], [0.2s, 2.5s]} in Figure 10. We find the number of successful payments is limited to a low level by applying *SR*. The results show that the proposed scheme can achieve a higher payment success ratio in the network with poor link connection (large delay). Additionally, the impact of link delay on the payment time by applying each scheme is similar. The number of available payment paths is critical for routing schemes that employ multiple paths. Figure 11 shows the performance of relevant routing schemes under different numbers of available paths. We first measure the metric on the number of successful payments as shown in Figure 11b. The number of successful payments increases as the number of reserved paths increases. *MBR* facilitates more payments to succeed under the same conditions. We then measure another important metric on averaged payment time (Figure 11a). Our multi-branch routing scheme outperforms other multi-path methods to achieve a relatively stable payment time.

The ability of large amount handling is also a critical metric to evaluate the efficiency of routing schemes. Due to the channel capacity limitation, the payment with large amounts has enhanced the challenge to be transferred across a PCN. However, off-chain network shows the advantage of lower payment fees to encourage users to pay in an off-chain manner. Hence, our experiment contains the analysis of routing schemes to route payments with a large payment amount. Figure 12 shows the comparison of different scheduling methods under different payment amounts. The payment amount is randomly sampled from each range set. We can obtain that the number of successful payments gradually decreases with the payment amount arguments. *MBR* can achieve a relatively higher success payment ratio than others. As the payment amount augments, the payment time of *AMP* and *SR* becomes unstable as shown in Figure 12b. Especially for the *SR*, the payment time is gradually increasing because of the strict payment success conditions.

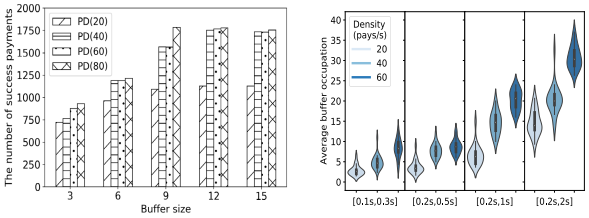
Figure 13a shows the change in the number of success payments with time increases. We reset the channel balance to range [100, 200] (USD) and link delay to range [0.1s, 0.3s], respectively. Notice that the total number of successful payments finally stabilized under different routing schemes. The reason is that payments in the bi-direction channel are imbalanced, thereby leading to a unidirectional channel [30] with no sufficient balance for further payments. Changes in funds at both ends of a payment channel will affect its payment forwarding capability, which is different from communication networks. The number of skewed channels gradually increases as the payment executes, and becomes a bottleneck restricting the overall payment success ratio [30]. The proposed routing scheme can also achieve a higher payment success ratio. It benefits from the lower collateral requirements. Besides, the final number of successful payments is slightly less than the *SR* method. Because the *SR* method splits payments to micro-payment units increases the liquidity [31] of funds in low-latency networks. For example, the payer cannot transfer 5 coins through two paths with a maximum capital of 4. But it can be done by applying *SR* method. Due to the



(a) The number of successful payments over time within the entire network.

(b) The average payment time comparison in steady-state.

FIGURE 13. Two metrics comparison with different scheduling methods.



(a) The number of successful payments under different payment densities with buffer size increases.

(b) Change in average buffer occupation of the entire network with different link delay.

FIGURE 14. Impact of buffer size and buffer occupation.

workloads exceeding the network processing capacity under the specified configuration, the slopes of the lines for these methods are close. Another metric is payment time as shown in Figure 13b. The *AMP* scheme achieves lower latency than other methods. The average payment latency of *MBR* is a little higher than others.

Through the above experiments, we found that when the network is crowded or the network delay is large, the performance of our algorithm in terms of payment success rate is better than other routing algorithms. However, it is worth noting that our algorithm has limitations in low latency and non-congested networks. In contrast, the *SR* algorithm outperforms other algorithms in terms of success rate, because it can call channel resources more finely. In addition, in order to ensure a high payment success ratio, the proposed algorithm has a slight increase in payment time compared with others. Because payment will constantly try each alternative routing path to reach its destination.

C. RESOURCE UTILIZATION IN PAYMENT PROCESS

We anticipate the buffer size of the LN node to affect the payment success ratio. A large buffer size allows LN nodes to cache more route information for payments. Each LN node can set up a buffer with a suitable size, which is close to the peak of routing information to be cached. It can be affected by the payment density and link delay. We first analyze the buffer occupation under different link delays. The buffer size of each node is set to a fixed value of 100. Figure 14b shows the average buffer occupation under different link delays and payment densities. The buffer occupation becomes higher as the link delay and payment density increase. We find that the network with high payment density and large transmission delay requires a large size buffer to cache route information for payments. In contrast, if a network with low payment density and high responsiveness, the issued payments can be quickly settled which cannot demonstrate the caching advantages of our routing scheme. Figure 14a shows the changes in the number of successful payments under different payment densities as the buffer size increases. The link delay is set to the range $[0.2s, 1s]$. A larger buffer size brings much more successful payments. But the channel capacity limits the network throughput reflected in the number of successful payments. Therefore, LN nodes need to find a tailored buffer size in crowded LN with poor link connections for payment routing.

VII. CONCLUSION

In this work, we study the routing issues in payment channel networks and reveal the path overlapping phenomenon in the payment process. We elaborate on the impact of path overlapping on payment routing. To offset the impact, we present a novel multi-branch routing scheme to build an efficient route for off-chain payments. The path selection and its ordering are both factors to affect payment efficiency. Hence, we further propose a Markov Chain-based routing algorithm to solve these concerns. Payers in PCNs can obtain near-optimal payment path planning by employing our algorithm. To verify the high performance of our algorithm, we develop a simulator of LN to simulate the payment routing process in the network layer. The simulation results indicate that the proposed routing algorithm can achieve a maximum of %15 improvement in payment success ratio compared with other routing schemes. Meanwhile, the collateral requirement of the proposed method is close to that of single-path routing methods but lower than most multi-path routing schemes.

REFERENCES

[1] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, p. 21260, 2008.

[3] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of ethereum smart contracts," in *Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain*, May 2018, pp. 9–16.

[4] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3416–3452, 4th Quart., 2018.

[5] *Proof-of-Stake Ethereum*. Accessed: Dec. 2022. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/

[6] J. Gobel and A. E. Krzesinski, "Increased block size and bitcoin blockchain dynamics," in *Proc. 27th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2017, pp. 1–6.

[7] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Jul. 2018, pp. 122–128.

[8] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2020, pp. 777–796.

[9] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2019, pp. 370–381.
[10] 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. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–9.
[11] H. Xue, Q. Huang, and Y. Bao, "EPA-route: Routing payment channel network with high success rate and low payment fees," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 227–237.
[12] V. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment-channel networks," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Malaysia: Springer, 2020, pp. 304–324.
[13] (2018). *AMP: Atomic Multi-Path Payments Over Lightning*. [Online]. Available: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html
[14] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 801–815.
[15] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," Cryptol. ePrint Arch. Tech. Rep., 2016.
[16] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy preserving payments in credit networks," in *Proc. Netw. Distrib. Secur. Symp.*, 2015, pp. 1–15.
[17] (2016). *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. [Online]. Available: https://lightning.network/lightning-network-paper.pdf
[18] *Basis of Lightning Technology Documents*. Accessed: Dec. 2022. [Online]. Available: https://github.com/lightningnetwork /lightning-rfc
[19] *The Current Path-Finding Implementation in LN*. Accessed: Dec. 2022. [Online]. Available: https://github.com/lightningnetwork/lnd/blob/master/routing/pathfind.go
[20] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band openflow networks," in *Proc. 9th Int. Conf. Design reliable Commun. Netw. (DRCN)*, Jun. 2013, pp. 52–59.
[21] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, 2013.
[22] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, Sep. 1996.
[23] X. Yuan, "Heuristic algorithms for multiconstrained quality-of-service routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 244–256, Apr. 2002.
[24] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6301–6327, Oct. 2013.
[25] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[26] H. Huang, S. Guo, W. Liang, K. Li, B. Ye, and W. Zhuang, "Near-optimal routing protection for in-band software-defined heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 11, pp. 2918–2934, Nov. 2016.
[27] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," Bitfury Group, The Netherlands,White Paper (Version 1.0), 2016.
[28] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 455–471.
[29] Y. Kano and T. Nakajima, "A novel approach to solve a mining work centralization problem in blockchain technologies," *Int. J. Pervasive Comput. Commun.*, vol. 14, no. 1, pp. 15–32, Apr. 2018.
[30] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 439–453.
[31] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Barcelona, Spain: Springer, 2018, pp. 67–75.

XIAOFEI LUO received the master's degree from the Department of Computer Science and Information System, The University of Aizu, Aizuwakamatsu, Japan, in 2018, where he is currently pursuing the Ph.D. degree with the Computer Organization Laboratory. His research interests include blockchain, reinforcement learning, computer networks, and payment channel networks.

PENG LI (Senior Member, IEEE) received the B.S. degree from the Huazhong University of Science and Technology, China, in 2007, and the M.S. and Ph.D. degrees from The University of Aizu, Japan, in 2009 and 2012, respectively. He is currently an Associate Professor with The University of Aizu. He has published over 100 technical papers on prestigious journals and conferences. His research interests include cloud/edge computing, the Internet of Things, machine learning systems, and related wired and wireless networking problems. He won the Young Author Award of IEEE Computer Society Japan Chapter, in 2014, and the Best Paper Award of the 2016 IEEE TrustCom. He supervised students to win the First Prize of IEEE ComSoc Student Competition, in 2016. He is an Editor of *IEICE Transactions on Communications* and IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY.