Intelligent Scheduling for Off-Chain Transactions in Payment Channel Networks

Xiaofei Luo

A DISSERTATION

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems

The University of Aizu

2022

The thesis titled

Intelligent Scheduling for Off-Chain Transactions in Payment Channel Networks

by

Xiaofei Luo

is reviewed and approved by:

Main referee

Senior Associate Professor

Peng Li

Professor

Anh T.Pham

Senior Associate Professor

Chunhua Su

Senior Associate Professor

Yuichi Yaguchi

THE UNIVERSITY OF AIZU

Oct. 2022

Contents

List of Figures

List of Abbreviations

PCNs	Payment Channel Networks
LN	Lightning Network
LND	Lightning Network Daemon
RL	Reinforcement Learning
DQN	Deep Q-Network
HTLC	Hash Time-Lock Contract
PAPCN	Priority-Aware Payment Channel Network
FIFO	First-in First-out
LIFO	Last-in First-out
TPS	Transaction per second
MDP	Markov Decision Process
BOLT	Basis of Lightning Technology

Acknowledgment

In the period of my doctoral study, I have got a great range of assistance and support. I would like to thank all the people who helped me during this period.

First of all, I would like to show my deepest gratitude to my supervisor, Prof. Peng Li, who has provided me with professional guidance throughout my doctoral research. Without his instructive suggestions and valuable comments, this dissertation would not have been accomplished.

His passionate attitude toward research always inspired me. He convincingly inspires us to make more improvements in paper writing and presentation.

My sincere appreciation also goes to Prof. Anh T.Pham, Prof. Chunhua Su and Prof.Yuichi Yaguchi for their reviews and constructive comments for this dissertation. Their advice remind me to improve my work and inspire me to solve problems with divergent thinking. Many thanks for the time and the patience of all referees.

I really appreciate it so much that my friends and lab members Qinglin Yang, Tao Liu, Changhong Zou, Fahao Chen, Boqian Fu, Chong Li. They provided me much help in my doctoral period and gave me much advice on thinking. we usually discuss problems together.

Also, many thanks to these guys who take care of each other in life, Jianbo Xu, Zhaoyang Han, Kungan Zeng, ever lived in the same mansion. I did not think it was possible to have a Ph.D. and so much fun at the same time!

Finally, I must express my gratitude to my family for their encouragement and sup- port. With their love, I could study abroad and living in another country.

Thanks for all your encouragement! Best wishes to you!

Abstract

A payment channel network (PCN) is one of the promising solutions for scalable blockchains since it shows great potential in improving blockchain network throughput. However, the growing number of transactions and the payment-channel sharing of concurrent transactions can lead to channel congestion. Although many studies have proposed different solutions to solve this problem, they ignore the fact that applications may have different transaction rates at different times. We first propose a priority-aware PCN to meet the requirements of those transactions. Senders in priority-aware PCNs can specify the priority of their transactions by paying a corresponding forwarding fee on each hop along the transaction path. However, capacity competition occurs on the shared hops. Moreover, we propose a multi-agent DQN-based priority assignment algorithm to address the competition issue and design a PCN simulator for performance evaluation. Simulation results show that our solution can guarantee a high throughput of transactions, and assign priorities appropriately to balance the transaction rate and forwarding fee cost. The experimental results demonstrate that the priority scheduling scheme can achieve higher transaction throughput and success ratio than other scheduling methods in a congested PCN environment.

In highly dynamic PCN, concurrent payments compete for channel capacity on shared payment channels leading to payment failure. All of the established contracts shall be canceled. To complete the payment, the sender needs to research a now available path for resending. It brings overhead on pathfinding and contract reestablishment over overlap channels. Additionally, the prior released channel capacity can be preempted by other payments. 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 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 Lightning Network (LN) simulator to simulate the payment routing process in the network layer. The simulation results indicate that the proposed routing algorithm can reach a higher payment success ratio compared with other routing schemes. Meanwhile, it requires the same collateral as most routing methods, lower than the Atomic Multi-Path (AMP) routing.

概要

ペイメントチャネルネットワーク（PCN）は、ブロックチェーンネットワークのスループットを向上させる大きな可能性を示しているため、スケーラブルなブロックチェーンの有望なソリューションの 1 つです。ただし、トランザクション数の増加と同時トランザクションの支払いチャネル共有は、チャネルの輻輳につながる可能性があります。多くの研究がこの問題を解決するためにさまざまな解決策を提案していますが、アプリケーションがさまざまな時間にさまざまなトランザクションレートを持つ可能性があるという事実を無視しています。まず、これらのトランザクションの要件を満たすために、優先度を意識した PCN を提案します。優先度を意識した PCN の送信者は、トランザクションパスに沿った各ホップで対応する転送料金を支払うことにより、トランザクションの優先度を指定できます。ただし、容量の競合は共有ホップで発生します。さらに、競合の問題に対処し、パフォーマンス評価用の PCN シミュレータを設計するために、マルチエージェント DQN ベースの優先順位割り当てアルゴリズムを提案します。シミュレーション結果は、当社のソリューションがトランザクションの高スループットを保証し、トランザクションレートと転送料金コストのバランスをとるために適切に優先順位を割り当てることができることを示しています。実験結果は、優先スケジューリング方式が、混雑した PCN 環境で他のスケジューリング方法よりも高いトランザクションスループットと成功率を達成できることを示しています。

高度に動的な PCN では、同時に支払いが共有支払いチャネル上でチャネル容量を競合し、支払いに失敗しました。締結したすべての契約はキャンセルしなければならない。支払を完了するには、支払送信者は現在使用可能な再送信パスを検討する必要があります。オーバーラップチャネル上の経路探索と契約再構築にオーバーヘッドをもたらします。さらに、先に解放されたチャネル容量は、他の支払いによって優先されてもよい。支払チャネルネットワークにおけるルーティング問題を研究し、支払過程におけるパスオーバーラップ現象を明らかにした。経路重複が支払経路に与える影響について詳細に述べた。影響を差し引くには、効率なチェーン外支払経路を構築するための新しいマルチブランチルーティングスキームを提案します。パス選択とそのソートは、支払い効率に影響を与える要因です。そこで、これらの問題を解決するためにマルコフチェーンに基づくルーティングアルゴリズムをさらに提案した。PCN 中の支払人は、提案されたアルゴリズムを使用することにより、最適に近い支払経路計画を得ることができる。私たちのアルゴリズムの高性能を検証するために、ネットワーク層の支払いルーティングプロセスをシミュレートするために、稲妻ネットワーク（LN）シミュレータを開発しました。シミュレーションの結果、提案されたルーティングアルゴリズムは、他のルーティングスキームに比べて高い支払い成功率を達成できることが明らかになった。同時に、ほとんどのルーティング方法と同じ補助装置が必要であり、原子マルチパス（AMP）ルーティングよりも低い。

Chapter 1

Introduction

1.1 Introduction

The digital cryptocurrencies(such as Bitcoin [1] and Ether) become an alternative payment way for modern payment network. Instead of the traditional centralized ledger maintained by a trust third party, the cryptocurrencies have a fully decentralized ledger which updates by consensus protocols(such as proof-of-work, proof-of-stake [2]) of blockchain and maintained by all of the blockchain participates. The participates can transfer their money to other parties by on-chain transaction. Public blockchian allows parties(peers and organizations) to freely join and leave. The parties requires miners to transport their transactions to a block. The block is then send to all of the parties for consensus. The consensus process is time-consuming and expensive. Hence, micropayments [3, 4] with small value may require higher fee for the on-chain processing by miner than its actually worth in Bitcoin, furthermore, these on-chain transactions also lead to the performance issue due to the occupancy of block space.

On-chain transactions are limited by the inherent challenges, transactions per second (TPS), which is involved to block size and consensus mechanism of blockchain, e.g. bitcoin consensus requires 10 minutes to confirm transactions in 1MB block. The throughput of bitcoin is limited around 7 TPS. Blockchain scalability solutions are proposed to improve the transaction throughputs, which categorised as layer-1 scaling solutions and layer-2 solutions. The layer-1 scaling solutions are on-chain scaling schemes that upgrades the existing architecture of public blockchain, such as sharding mechanism [5], directed acyclic graph structure in avalanche [6]. Blockchain layer-2 solutions are off-chain networks that bring frequent micropayments to under chain layer, such as payment channel networks (PCNs) and optimistic rollups [7]. Off-chain transactions avoid costly expensive on-chain transaction fees and slow consensus process.

In this dissertation, we mainly focus on off-chain transactions in payment channel networks. Although blockchain layer-1 scaling solutions increase the TPS of blockchain, it requires to interact with the blockchain ledger and expensive transaction fees. Public blockchains provide strong security, where on-chain transactions are serialized by miner and verified through consensus algorithm. In contrast, the payment channel network is proposed as a blockchain layer-2 network, which realizes instant transactions without consensus process and high transaction fees. However, it is difficult to guarantee the security of off-chain transactions without the endorsement of online miners. The off-chain protocol first formulates a punishment policy to punish dishonest users who attempt to steal funds from the payment channel. Additionally, it applies smart contracts to implement atomic transactions to protect user's coins from loss.

In PCNs, payment channels are established between parties with a direct peer-to-peer connection and their deposits as the channel capacity. Both two parties commit to maintaining the balance of their deposit as a two-party ledger on the bidirectional channel. The channel capacity is the sum of the channel balance. Payment channels and parties have jointly constructed a distributed network called payment channel networks. A well known off-chain implementation in Bitcoin is Lightning Network (LN) which allows the parties to transfer their money between each other under Bitcoin blockchain [8]. Off-chain transactions between two parties without a direct channel connection are forwarded through a tailored routing path, composed of payment channels and intermediate parties. Concurrent transactions can share payment channels on their routing path. Due to the limitation of the channel capacity, transaction failure incurs when its amount surpasses the channel capacity on its routing path.

Channel congestion induces frequent transaction failures decreasing the transaction throughput of PCN. The congestion issues may be caused by a combination of limited channel capacity, high volume of transactions and poor scheduling mechanism. The channel capacity is the maximum amount of funds that can be used by transactions on a channel, and it is fixed unless user deposits. High volume of transactions bring high traffic loads on payment channels. Except for the channel capacity, transactions consume the smart contract processing resource of payment channels. The maximum number of concurrent contracts on a channel is a network parameter with different upper bound in off-chain implementations. Due to the various time requirements of transactions, transaction senders customize expiry times for their transactions. The scheduling mechanism of traditional PCNs ignores the user requirements and just queues their transactions in a first come first out order. In a congested network, a large number of blocked transactions will fail due to expiration, resulting in a decline in network throughput.

Transaction path selection will also affect the TPS of PCN. A suitable path can guarantee high transaction success rate and low transaction time. In general, short paths have higher opportunities to get lower transaction time. An available short path must ensure that payment channels over the path have sufficient capacity to route the transaction. However, payment channels can be shared by different transactions. Path becomes unavailable when concurrent transactions consume up the channel capacity, which is common in highly dynamic networks. If each transaction sender or a centralized server can obtain the path information of others, the path programming can be optimized. Unfortunately, PCNs protect user privacy that senders cannot track the channel utilization information. Moreover, centralized control brings security issues. Multi-path routing is a promising solution to solve these concerns that is normally used in communication networks like multi-path TCP protocol [9]. It increases the transaction success probability, but brings redundant loads.

The main idea of this dissertation is to improve the network throughput in payment channel networks and provide novel off-chain scaling solutions. Due to the privacy protection of fully distributed PCNs, it's hard to obtain the status and the utilization information of payment channels to make a centralized control for off-chain transaction flows. Furthermore, Highly dynamic network brings fast varying channel status that leads to congestion issues and inefficient routing. We present two different proposals to solve the scheduling and routing issues, as illustrated in Figure 1.1. We first focus on the transaction scheduling issues in PCN and propose *proposal* 1, a priority-aware PCN. We then explore the transaction routing issues and propose *proposal* 2, a novel multi-path routing scheme. Our contributions are listed as follows:

- We use real transaction data of the Ripple network to test the transaction performance of LN and analysis the characteristics of off-chain transactions.
- We develop in terms of two simulators to simulate the transaction scheduling and routing processes in a PCN.
- We present a multi-agent reinforcement learning-based transaction scheduling method to achieve an efficient scheduling in prioritized PCNs.
- We propose a novel multi-path routing mechanism and implement a distributed Markov approximation algorithm for efficient routing in PCNs.

1.2 Organization

The rest of the thesis is organized in the following pattern.

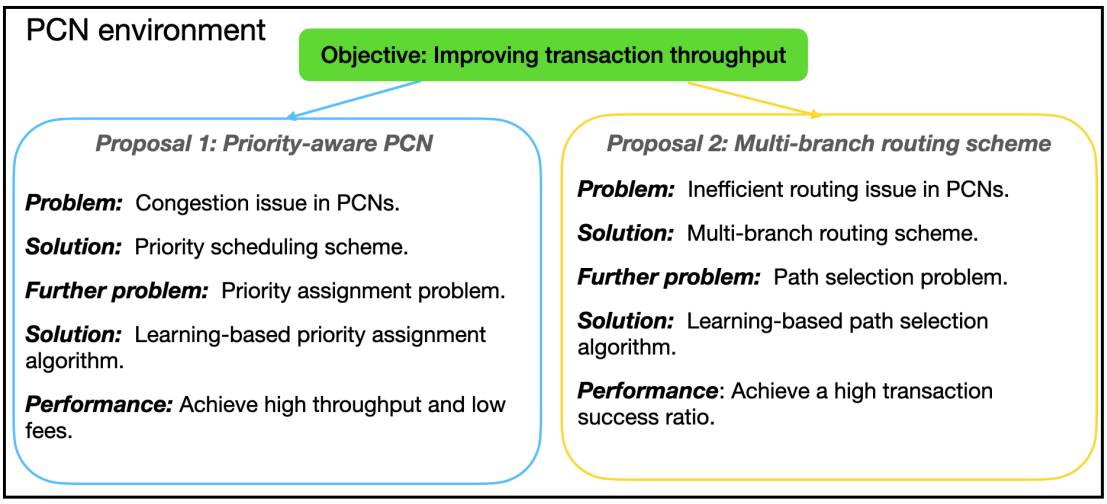


Figure 1.1: Organization of thesis structure

In **Chapter 2**, the research background is given, which mainly includes the fundamental concepts of blockchain and PCNs.

Next, in **Chapter 3**, we propose a priority-aware PCN to meet the requirements of those transactions. Senders in priority-aware PCNs can specify the priority of their transactions by paying a corresponding forwarding fee on each hop along the transaction path. However, capacity competition occurs on the shared hops. Moreover, we propose a multi-agent DQN-based priority assignment algorithm to address the competition issue and design a PCN simulator for performance evaluation. Simulation results show that our solution can guarantee a high throughput of transaction, and assign priorities appropriately to balance the transaction rate and forwarding fee cost. The proposed priority scheduling scheme can achieve higher transaction throughput and success ratio than other scheduling methods in a congested PCN environment.

Then, in **Chapter 4**, we explore the payment routing problem and propose a multi-branch payment routing scheme. Senders need to prepare multiple routing paths that contains a primary path and several branch paths as the standby path for their payments in our scheme. When payment failure occurs, instead of directly aborting the payment, the failed payment can be prior handled by fork nodes over its path as they receives the failure message. The fork node forwards the payment through across a branch routing path to the destination. Only a part of established contracts requires to be canceled. Our scheme theoretically achieves a higher payment success ratio than the single-path routing, as the same time avoid the harsh completion conditions in multi-path routing.

Finally, **Chapter 5** concludes this dissertation.

1.3 Publications

The following papers have been published or accepted by journals or conferences.

1. Major journal paper

    - **Xiaofei Luo**, Peng Li, *et al.* Learning-Based Off-Chain Transaction Scheduling in Prioritized Payment Channel Networks [J]. IEEE JSAC SI-Intelligent Blockchain special issue. (published)
    - **Xiaofei Luo**, Peng Li, *et al.* LEAF: Let' s Efficiently Make Adaptive Forwarding in Payment Channel Networks [J]. IEEE Access. (Under review)
    - Qinglin Yang, **Xiaofei Luo**, Peng Li,et al. Cooperation of MobileDevices for Fast Inference of Deep Learning Applications [J]. MobileNetworks and Applications. (published)

2. Major conference paper

    - **Xiaofei Luo**, Qinglin Yang, Peng Li, et al. Grouping Strategy for RFID-Based Activity Recognition in Smart Home (*IEEE* Conference). 2019 International Conference on Internet of Things (iThings)
    - Peng Li **Xiaofei Luo**, Toshiaki Miyazaki. Privacy-preserving Payment Channel Networks using Trusted Execution Environment (*IEEE* Conference) ICC 2020
    - Qinglin Yang, **Xiaofei Luo**, Peng Li, *et al.* Collaborative Inference for Mobile Deep Learning Applications (Conference) 2nd International Conference on 5G for Ubiquitous Connectivity.
    - Qinglin Yang, **Xiaofei Luo**, Peng Li,*et al.* Computation offloading for fast CNN inference in edge computing (*ACM* Conference) International Conference on Research in Adaptive and Convergent Systems, Chongqing China, September 2019.

Chapter 2

Background

2.1 Blockchain

Blockchain, a fully decentralized ledger, is emerged and integrated into people's daily life in recent years. It widely adapts to the modern transaction system and the field of economics, education, and politics. According to the different openness of blockchain, the blockchain network can be separated into two classes: permissionless blockchain and permissioned blockchain. Permissionless blockchains mainly represent public blockchain which allows participants to freely join and leave. Bitcoin and Ethereum are typical public blockchains. Permissioned blockchain can be private or a hybrid of private and public where participants are required to provide their token to join the system. The network throughput is an important metric to measure the performance of the transaction system. General networks such as data-center networks, and VISA support more than 100K and 46000 transactions per second respectively. Permissioned blockchains such as Hyperledger Fabric, ripple supports 1K - 4K transactions per second [10, 11]. However, the public blockchains can only reach 7 - 15 TPS which cannot satisfy the demand of modern transaction systems [12].

2.1.1 Blockchain trilemma

On-chain transactions need to reach a consensus across the network to ensure the consistency of the blockchain ledger. These transactions are first packaged into blocks by blockchain nodes called miners, which are required to solve a mathematical puzzle as proof of their work. Blocks are then broadcast to the network to complete the consensus process. Information communication and block broadcasting consume network bandwidth resources. Especially for decentralized systems, redundant traffic will exhaust network bandwidth and aggravate network transmission pressure. In addition, highly decentralized systems will have more internal

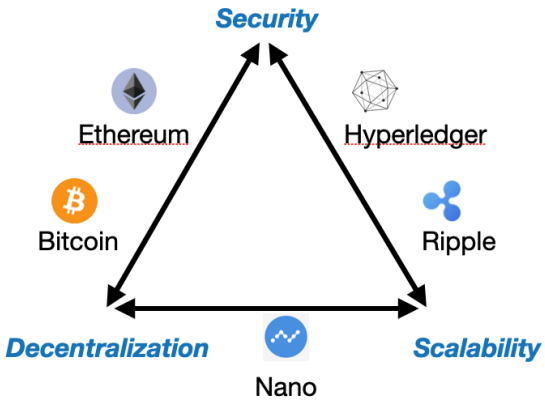


Figure 2.1: The blockchain trilemma

nodes. Redundant and decentralized storage on these nodes ensures the security of the system, but reduces the overall transaction verification rate. In a nutshell, the consensus mechanism of blockchain maintains a secure and trusted distributed ledger environment, but it also leads to poor scalability.

The blockchain trilemma is finding a balance between decentralization, security, and scalability [11] as shown in Fig. 2.1. Public blockchains like Bitcoin, Ethereum ensure a secure and decentralized environment but poor scalability. Permissioned blockchains like Hyperledger and Ripple improve the scalability but abandons the characteristics of full decentralization. Nano chain takes into account decentralization and scalability, but poor security. Centralized systems are more scalable but less secure. Centralized computing capacity speeds up transaction verification rate, but reduces its ability to resist attacks. For example, the sharding mechanism [13], as a promising layer-1 scaling solution, divides the entire network into several shards that guarantee partial decentralization. Nodes in a shard take less time to reach consensus than those in a public blockchain. However, it will be cheaper for attackers to attack these shards.

2.1.2 Layer-2 solutions of blockchain

The main network of public blockchain or called the blockchain layer-1 network can be regarded as a congested highway. In order to improve the traffic throughput, a new road (layer-2 network) can be laid to connect with the highway to divert the traffic from the main road. Blockchain layer-2 networks are proposed to solve the scalability issue of the public blockchain. It is established over the layer-1 network and offloads the on-chain transactions to off-chain networks. There are four blockchain layer-2 solutions:

- Payment channel is a bidirectional channel that maintains the deposit of

connected users. Transactions can be routed through payment channels that update their balances to transfer coins. It avoids the slow on-chain consensus process and expensive transaction fees. Only the final state of deposits on the channel will be released onto the blockchain.

- Rollups, bundle multiple transactions into a batch. These transactions are performed in an off-chain network and then submitted to *Rollup* servers for validity verification. The final state of transactions will be sent back to the layer-1 network and recorded.

- Sidechains is an independent blockchain network bridged to the public blockchain. They have a customized consensus mechanism for transaction verification that reduces the burden of the public blockchain.

- Nested blockchain, is an internal blockchain embedded within the public blockchain. Public blockchains can set the network parameters and operating rules for their nested blockchain but do not participate in transaction execution.

In this dissertation, we mainly focus on the payment channel network which is a promising scaling solution for public blockchains.

2.2 Payment channel network

2.2.1 Payment channel

Payment channel creates a logical channel between two users over the peer-to-peer(P2P) connection, which allows off-chain users to perform multiple payments and does not require them to commit to the public blockchain [14]. A direct physical connection between the two peers on the channel is unnecessary. The two peers are connected with each other via logical links. The payment channel between the two peers is different from a general network connection, which opens up a special channel for peers to transfer their money. The general network is used for information exchange between peers and payment channels, including message and data packet transmission. The payment channel can use the received packets and messages for transaction commitment and channel state (ledger) updating.

Hash Time-Lock Contract (HTLC) is proposed to handle the atomicity problem for a multi-hop payment [8]. To establish and cancel a contract on the payment channel, the two connected peers requires to update their signature and synchronize the channel status. There are only two necessary on-chain operations that

upload the initial and final state of channel balance to the blockchain for consensus while the channel is initialized and closed. The initialization of a channel attempt to deposit the balance from one or both party, which require an open channel commitment on the public blockchain. Each off-chain payment on the channel just shifts the channel balance along the payment direction, in a nutshell, it atomically increases the balance for the payee side, at the same time reduces the balance for the payer side. After the closing channel commitment is completed, the deposits in the released channel are sent back to associate parties. According to the bidirectional characteristic of the payment channel, a simple PCN can be considered as a directed graph with a set of blockchain users as the vertices and payment channels as weighted edges. At least one payment channel is needed for an independent user to connect the entire network.

To support channel and node discovery in LN, the gossip protocol is applied to exchange messages between all peers in the network [15]. Each peer broadcasts three gossip messages to their neighbors: node announcements (peer ID, host, and port), channel announcements, and channel update messages (announces the fees and minimum time lock). Once the gossip message has been processed, it's added to a list of outgoing messages, destined for the processing node's peers, replacing any older updates from the origin peer. This list of gossip messages will be flushed at regular intervals, such a store-and-delayed-forward broadcast is called a staggered broadcast. Each peer can positively gather information on the network topology and forwarding fees from the blockchain itself or be disseminated by a gossip protocol [12]. A node should flush outgoing gossip messages once every 60 seconds, independently of the arrival times of the messages [15]. In this manner, a peer can observe all of the nodes in PCNs, because the network topology is known to every peer.

The instantaneous information of a payment channel can be only known to the two peers on the channel, which includes [16]:

- *Channel ID*, is the unique channel ID for the channel.
- *Capacity*, is the total amount of funds held in this channel.
- *Local/remote balance*, is the current balance of this node/ the counterparty in this channel.
- *Commit (forwarding) fee*, is the amount calculated to be paid in fees for the current set of commitment transactions.
- *Unsettled balance*, is the unsettled balance (balance locked by HTLCs) in this channel.

- *Pending HTLCs*, is the list of active, uncleared HTLCs currently pending within the channel.

Due to the privacy protection of PCNs, each peer can only obtain partial information (e.g. the capacity, the forwarding fee) of any non-directly connected payment channels. Additionally, each peer can only observe a snapshot of non-directly connected payment channels within the network. Network delay reduces the availability of the observed channel information. Hence, it's difficult to ensure high payment throughput in a highly dynamic network.

2.2.2 Payment routing in PCNs

PCNs apply source routing to payment routing so that senders are able to fully specify the intermediate nodes in their routes [17] or employ a path-finding algorithm to find an available payment path. Current LN implementation uses a revised version of Dijkstra's algorithm to find a single shortest path between the source node and the destination [18]. In a decentralized network at intervals, PCN nodes first run a pseudo-random process to decide on beacon nodes at the beginning of the path-finding process. Neighbors of each beacon node broadcast their shortest route to the beacon node. The neighbors of the neighbor of each beacon node get the shortest path information and in turn, broadcasts their shortest route. When every reachable node receives the shortest path from the neighbor to the beacon node, they update its shortest path to the beacon node. Finally, a new round of beacon node selection is started.

The payment path is sensitive information associated with the sender's privacy. Each sender has no knowledge about the payment path information of other senders. Hence, the sender cannot directly observe the congestion that shares payment channels on the payment path with others. Due to the information restriction, senders cannot get such information (like how many payments are on the intermediate hops, and how many coins are carried by other payments) on the payment path. But the sender can learn the congestion information from the payment results. When frequent payment failure occurs, the routing path must be congested. As aforementioned, channel congestion causes payment failure.

There are two terms of overhead when a failed payment changes its path for re-routing. The first overhead induces by hash time-locked contract (HTLC) establishment and revoking. The HTLC is used by transactions to lock coins on channels for their use within a specified timeout. A successful transaction requires establishing HTLC on each hop along its path. When the transaction is failed on an intermediate hop, all of the prior established HTLC will be revoked. Then, after path re-finding, the failed transaction establishes new HTLCs along its new

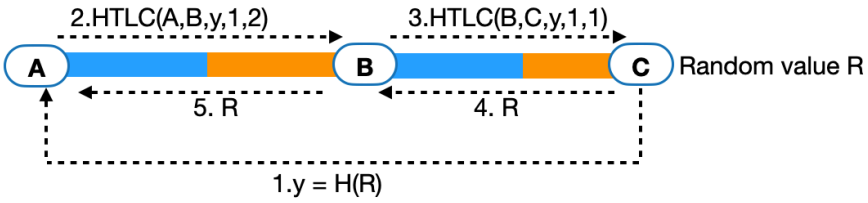


Figure 2.2: The HTLC in a transaction process

path. The second overhead induces by path re-finding. In a large-scale PCN, the pathfinding algorithm will be time-consuming. It traverses the nodes around the destination in the topology and analyzes the node' s features, including fees and time-lock settings. Since PCN is a highly dynamic network, it cannot guarantee that failed transactions can be sent to their destination through new paths. The frequent retry operation brings higher overhead.

2.2.3 Implementing atomic payments in PCNs

An HTLC is comprised of a hash lock and a time lock. The hash lock H requires the payment recipient to reveal a key value R to unlock. The time lock is a payment expiration time, which unlocks automatically after a waiting timeout. The detailed process to set and settle the HTLC is depicted in Fig. 2.2. First, the payment recipient C initializes a random key R and gets the hash value H = H(R). The hash value H is then sent from C to the sender A. A holds H and sets a hash lock between her and the intermediate node B. The condition for B to unlock the hash lock is to provide the key R. The sender then locks her funds for the payment and sets a time lock to get the funds back. The intermediate node B receives the request while he cannot get the funds without providing the key. Such HTLCs with a diminishing timeout are then established at the rest hop from B to the recipient C. Once the payment recipient receives the request and thus reveals the key R to the intermediate node B, the established HTLCs and associated channel capacity updating can be settled from the final hop to the first hop as long as the key is routed to payment sender over the payment path.

In this manner, a payer can send payment to the payee without opening a new channel between them. In case of an uncooperative intermediate user on the payment path attempts to interrupt the payment process, the influenced user can unilaterally close the channel between them and publish the last commitment transaction on the public blockchain for arbitration. Although the contract protected payments privacy and security in PCNs, it also induces the limitation on information exchange of each mutually independent payment. This limitation is one of the challenges to implementing a decentralized scheduling scheme.

2.2.4 Existing PCNs

The Lightning Network (LN) is an off-chain payment protocol that performs transactions under the Bitcoin blockchain layer. Payments are instant in LN without running a cost-time consensus of proof-of-work (6 blocks or one hour for confirmation in the Bitcoin Network). As a complete implementation of an LN node, Lightning Network Daemon (LND) has been designed for the Bitcoin blockchain, which is widely adopted modern alternative monetary payment system. The transaction rate of LND is effectively only limited by the network bandwidth between the participating peers.

The Raiden Network (RN) is an off-chain scaling solution for Ethereum blockchain, which aims to realize near-instant, low-fee, and scalable transactions [19]. This project is now in progress and supports ERC20 compatible tokens. On-chain transactions bring large costs on transaction fees. The RN reduces fee costs close to 7 orders of magnitude lower than on-chain transactions [20].

Chapter 3

Priority-Aware Payment Channel Networks

3.1 Introduction

Since the digital cryptocurrency emerged as an alternative monetary payment method, blockchain such as Bitcoin [21] has flourished in recent years. Its low transaction throughput brings scalability issue that becomes a bottleneck restricting the development of public blockchain. Payment channel network (PCN), as a scaling solution of on-chain network, has been proposed for improving the public blockchain scalability. Without frequently interacting with the slow blockchain, PCNs significantly reduces transaction latency and increases blockchain throughput. PCN is an off-chain transaction network consisting of cryptocurrency users and payment channels. Transactions between two PCN users without direct payment channel connection are forwarded through a routing path consisting of intermediate users and multiple channels with sufficient funds. Senders need to pay forwarding fee to intermediate users for transaction forwarding. However, concurrent PCN has serious congestion issues.

To solve the congestion problem in PCN, there exist some efforts on routing mechanisms, transaction flow scheduling, and congestion control protocols. *Yu et.al* [22] have proposed a fast routing mechanism that finds other available routing paths to transfers the overload transactions. Their work can solve the partial payment channel congestion problem. When the entire network is congested, the mechanism brings a large overhead. A multi-path scheduling method is proposed in [23] that sends a transaction over across different routing paths to resist transaction failures, but it spawns extra transaction load. *Sivaraman et.al* [24] explore a novel multi-path transaction flow scheduling method called *Spider*. Its basic idea is to split transactions into several parts and send them through multiple

routing paths. They develop a congestion control algorithm to control the number of transaction units allocated for each path. Their approach can achieve high utilization of payment channel funds, but transaction latency will be higher in a congested network. Because a successful transaction requires all its transaction units to be settled. The transaction latency depends on the complete time of the last transaction unit. In a congested PCN, concurrent transactions compete for the forwarding capacity of payment channels, resulting in fast varying path status. It causes transaction latency and rate jitter which makes the network performance unpredictable.

In this chapter, we first propose a priority-aware payment channel network (PAPCN) in which transactions are assigned different priority classes. The transactions with higher priority will be first forwarded to avoid transaction failure due to transaction deadline expiration. Each intermediate user maintains a priority queue to forward transactions according to their specified priorities. Hence, transactions in our PAPCN can make a multi-hop priority decision to be assigned with different priorities along its path. Due to the PCN is a distributed network without centralized control, we select the forwarding fee as a general priority identifier to classify transactions with various demands. Since the forwarding fee is specified by the transaction sender and encoded into the transaction payload, the priority assignment needs to be completed to determine the forwarding fee of each hop before the transaction is issued. In addition, the priority scheduling method is widely adopted for traffic control and solving the latency and rate jitter issue [25–27] in communication networks.

Each lightning user can be a transaction initiator (normally called a *sender*), and an in intermediate user serving for other senders. As a sender, it requires the knowledge about the forwarding fee information of intermediate users to initiate its transactions. As an intermediate user, it specifies the forwarding fee corresponding to each priority, aiming to formulate its forwarding policy and feed the policy back to associated senders. In our design, the forwarding policy defines a finite and discrete set of forwarding fees corresponding to different priorities. Senders specify the priority for their transactions according to their demand and adhere to the policies to determine the required forwarding fees. Meanwhile, different link statuses also affect priority assignment. For instance, the sender pays more forwarding fee for a higher priority to achieve a higher transaction forwarding rate on the congested hop. In contrast, low-priorities are assigned to uncongested hops for saving coins . Therefore, we elaborate a multi-priorities solution for senders to assign multi-priorities (one for each intermediate user over the transaction path) for their transaction flow.

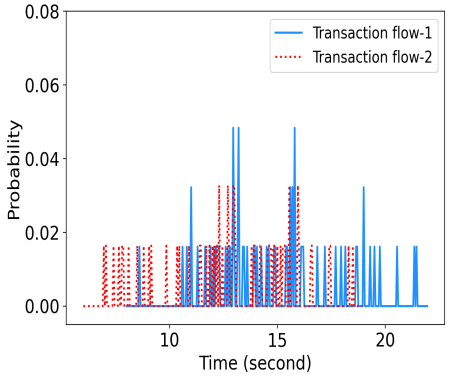
To implement an efficient multi-priorities scheme, we need to overcome several

challenges: 1). Unpredictable path status. PCN provides strong privacy protection that users on the transaction path only have the knowledge of the status of the connected channels. Moreover, dynamic transaction rate causes rapid changes in the number of transaction packets on channel links that lead to unpredictable path status. Hence, it's impossible for senders to track the path status of their transactions under the highly dynamic PCN environment. 2). Link capacity competition. On the shared hops, concurrent transactions compete for forwarding capacity. Due to the capacity of each hop is fixed, the forwarding rate of low-priority transaction flows drops when a higher priority is assigned to other flows. To offset the impact of priority adjustment, senders need to improve the priority of their transaction flows to compete channel capacity with others. 3). Large space for priority assignment. The space of a priority assignment solution for a transaction flow depends on the number of hops and each hop's priority classes over the routing path. Merging the solution of each flow to find the optimal solution for the entire network results in a large solution space.

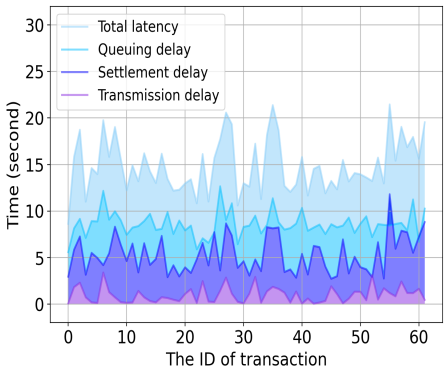
As a promising technique, Reinforcement learning (RL) is proposed for solving the problem in an uncertain, potential complexity situation, which guides the learning agent to maximize its cumulative reward by taking actions according to the status of dynamic environment [28]. We introduce RL into our PAPCN to solve the priority assignment problem under the environment without much information. But single-agent learning cannot ensure solution convergence in a competitive environment [29]. Hence, we propose a multi-agent RL-based priority assignment scheme to assign priorities to each transaction flow by controlling the forwarding fee paid to intermediate users. To handle the large dimensional solution space, we apply the deep Q-leaning (DQN) [30] into our scheme and propose multi-agent DQN-based priority assignment algorithm. In order to guarantee the convergence of the learning process, each participant is required to share partial information of historical priority assignment schemes with others. To achieve a high transaction rate, senders can choose to pay more forwarding fees to intermediate users to ensure that their transactions can be forwarded quickly. When their transaction rate cannot rise to higher in the congested PCNs, they can try to lower the forwarding fee to save their coins. Our goal is to balance the forwarding fee cost and achievable transaction rate by adaptively adjusting the transaction forwarding fee paid to each intermediate user.

We make the following contributions:

- We make a measurement on the real transaction trace in a complete implementation of the off-chain network to explore the traffic characteristics of off-chain transactions.



(a) The PDF of the transaction latency of two transaction flows in the sample.



(b) The breakdown of the transaction latency of transaction flow-1.

Figure 3.1: Transaction latency analysis of Lightning Network.

- We first propose a priority-aware transaction forwarding mechanism, which allows senders to flexibly assign priorities for their transaction flow to balance the transaction rate and its forwarding fee cost.
- We implement a multi-agent DQN-based transaction forwarding mechanism for efficient priority assignment and develop a simulator of LN to simulate the transaction scheduling process in network layer. The learning approach is then applied to the simulator to be proved its efficiency.

The rest of this Chapter is organized as follows. Chapter 3.2 reveals the motivation of our work, while Chapter 3.3 discusses the related works in PCNs and RL. We overview our system design and objective, the challenges of our work in Chapter 3.4. Chapter 3.5 details the multi-agent DQN-based priority assignment algorithm. Chapter 3.6 describes the system implementation. Chapter 3.7 presents the evaluation results, while we conclude this work in Chapter 3.8.

3.2 Motivation

Lightning Network (LN) is an off-chain protocol of the Bitcoin blockchain. To solve the congestion problem of LN, we first study the interaction between concurrent transactions and investigate the characteristics of these transactions. However, the difficulty we encounter is that it is impossible to access the real transactions that occurred in LN channels, due to the privacy protection policy of LN. As one of the off-chain network implementations, Ripple provides real transaction records for developers [31]. We collect 3000 Ripple transactions from [32] and randomly map all sender-receiver pairs into a simulated payment channel

network (the *simnet* mode of LND) with the Watts-Strogatz small-world topology [33]. The timestamp of the first transaction in the transaction records is served as the starting point of the simulation. We launch a thread for each transaction to monitor the transaction process. Through the LND *gRPC* APIs [34], we get the response of the transaction result and record the transaction latency information. We select two transaction flows that share channels with other transactions and show the probability distribution function (PDF) of their transaction latency in Fig. 3.1(a). It reveals transaction latency jitter when concurrent transactions exist.

From the result, we observe that the transaction latency is not stable. The value of the latency is around 10s $\sim$ 20s. At the same time, we monitor various delays in different phases of transaction execution and find that the transaction latency mainly includes the following three parts. 1) The first part denotes the *transmission* delay, which is the sum of the latency spending on transmitting the transaction through channels along the transaction's path. 2) The *queuing* delay represents the waiting time when the transaction is in the queues. Intermediate users receive transactions from the upstream channels and insert them into the forwarding queues. All of the transactions in this queue are then forwarded to the corresponding downstream channels in a First-in First-out (FIFO) manner. 3) The *settlement* delay is defined as the duration from the reception of the transaction to the moment the sender is acknowledged the response of this transaction. The transaction is settled as the transaction *fulfill* message is delivered to the sender along the opposite of the transaction path. The *fulfill* message is also queued on intermediate users and transferred through the channels over the transaction path. Hence, the *settlement* delay contains the *queuing* and *transmission* delay of the *fulfill* message. The Fig. 3.1(b) shows the breakdown of the transaction latency. We can see that the *queuing* delay (include its part in the *settlement* delay) is the key part of the total transaction latency. To solve the congestion problem and lower the transaction latency, we need to formulate a scheme for transaction flow controlling to reduce the *queuing* delay on the intermediate users.

In particular, the user demand on transaction latency is reflected in the time-lock of contracts in traditional PCNs. This time-lock is generally used for aborting the expired transaction but does not affect transaction scheduling. According to the different requirements of transactions submitted by diverse users, we classify all transactions into some categories. However, the scheduling method of traditional PCNs has inherent flaws in handling the transactions with different categories, which motivates us to find a scheduling method to facilitate efficient transaction forwarding of intermediate participants in PCNs. Then, our priority-aware scheduling method is proposed to solve this concern. The intermediate user

applies this method to schedule transactions with different priorities specified by senders. Senders pay forwarding fees associate with different priorities to intermediate users. Compared with the low-priority transactions, the high-priority transactions have higher probabilities to achieve a lower and stable transaction latency.

Since the actual transaction rates may be varying, senders need to dynamically assign priority to their transactions to play games with other adversaries in congested LN. In particular, the issued transactions cannot be intervened by reallocating more forwarding fees on the congested hops. Meanwhile, asynchronous and distributed transaction flows result in a highly dynamic instantaneous status for each hop, which is still a huge challenge for developing a global priority assignment scheme. Multi-agent reinforcement learning is suitable for solving distributed parallel tasks online [35]. Without a centralized node, each transaction flow can be regarded as a single learning agent gathered in a partially observable environment to play a Markov game. Our motivation is to design a multi-agent RL-based priority assignment scheme for all participants to balance the forwarding fee cost and transaction rates in this game.

3.3 RELATED WORK

Current researches focus on the different queuing discipline like Last-in First-out in Spider [24], and provide some congestion control methods [9, 36] to achieve high throughput. However, these researchers ignored that different transactions may have different rate requirements. Transactions with high rate requirements require to be serviced immediately to ensure a higher success ratio. In general, the capacity of the forwarding queue of an intermediate user to route transactions is fixed. These algorithms do not essentially improve the utilization efficiency of the congested hops but balance the transaction flow to other available transaction paths. These methods require several available paths to balance the transaction flow may increase the risk of sensitive information leakage leads to security issues.

Compared with the public communication network, PCNs have unique characteristics such as 'in-flight' transactions and Hash Time-Lock Contracts (HTLCs). Mizrahi *et al.* [37] introduced a channel congestion attack induced by the unresolved HTLCs to hold channel funds 'in-flight' that reduces the liquidity [38] of PCN. Considering that the unidirectional transactions may cause an imbalanced (skewed) payment channel issue, Sivaraman *et al.* [24] and Khalil *et al.* [20] proposed different solutions for balancing channel deposit. Based on their works, we focus on the rate issues instead of the issue induced by transaction amount in the scheduling process. Some approaches [22, 39] focus on the routing issue to improve

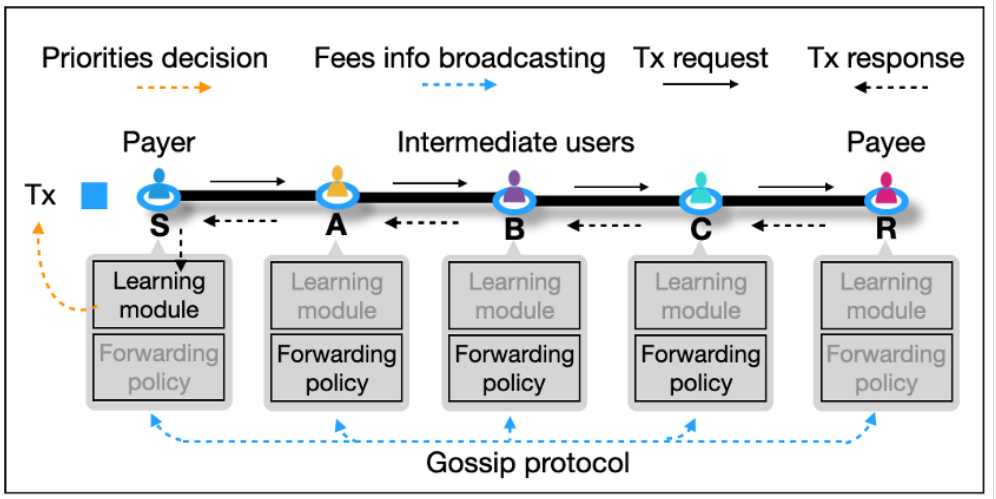


Figure 3.2: System overview of the priority-aware PCN. The trajectory shows the transaction process between payer S and payee R. The module with the gray font is not involved in S-to-R transactions.

the TPS of PCNs.

There are many works on different learning methods to solve multi-agent RL-based problems in the dynamic environment [40, 41]. Geng et al. [42] use Deep RL to solve the distributed multi-region traffic engineering problem. By the collaboration of two types of learning agents for different categories of traffics, the learning method finds near-optimal routing decisions to minimize the maximum edge cost. Liang et al. [43] focus on the spectrum sharing problem in dynamic vehicular networks by using a multi-agent Deep Q-network (DQN) based approach to allocate the vehicular-to-vehicular spectrum and power. In [44], they present a multi-agent advantage actor-critic policy for video caching to reduce the content access latency and traffic cost in a highly dynamic network edge. Similarly, in dynamic PCNs, it is desirable to use a multi-agent RL-based approach to solve the priority assignment problem. Agents must find a suitable scheme to ensure a high transaction rate and low forwarding fee in highly dynamic PCNs. Some of them influence and restrict each other to play a mixed cooperative-competitive game [45].

3.4 SYSTEM DESIGN AND CHALLENGES

3.4.1 System Design

In this session, we present the design of the priority-aware payment channel network. The basic idea is to schedule transactions by assigning them different priorities on their routing path. The overview of our design is shown in Fig. 3.2

and described as follows: We design a priority-aware forwarding queue to schedule transactions for the LN user. Each LN user maintains its own forwarding queue and formulates a forwarding policy for the queue to reveal the forwarding fee corresponding to each priority. The policy and its updated message are broadcast to other participants by a gossip protocol. Based on the original structure of the transaction, our design extends it with an additional field. This field reveals the specified priorities of the transaction on its routing path like $\{x_1, ..., x_h\}$ (e.g. x_h indicates the priority assigned by the sender for the transaction at hop h). Additionally, we update the way to calculate the total forwarding fee of a transaction. The transaction sender calculates the fee based on the field of priority and the forwarding policy of each intermediate user. In the transaction process, intermediate users learn the priority of the current hop from the received transaction and charge the corresponding forwarding fee. If the remaining transaction amount matches the amount that needs to be forwarded within the routing information, the incoming transaction is then added to the tail of the queue associated with this priority. Otherwise, the transaction is aborted. All of the transactions in the priority queue wait for forwarding until it reaches the destination or the expiration time.

Although the priority scheduling scheme can realize the classified scheduling of transactions, concurrent transactions competing for resources on their shared intermediate users will affect the efficiency of transaction forwarding. In addition, priority selection is directly related to the user's expenditure on forwarding fees. Therefore, the sender needs to assign appropriate priorities to control the expenditure on forwarding fees while increasing the network throughput. Due to the PCN being a fully distributed system, we further design a multi-agent RL-based priority assignment algorithm. Our purpose is to balance the forwarding fee expenditure and network throughput. Since the sender cannot track the routing path statuses of the transaction, the learning algorithm uses the transaction response to estimate the degree of congestion of each intermediate user. We launch an additional process for each sender to execute the learning algorithm. This process collects the transaction response and makes a short-time decision on priority assignment for the transactions sent to a specific destination. In highly dynamic PCNs, it's hard to guarantee a stable transaction rate by a determined priority assignment scheme. According to the current state of the transaction sender, the learning algorithm can dynamically adjust the scheme to achieve a customized performance objective.

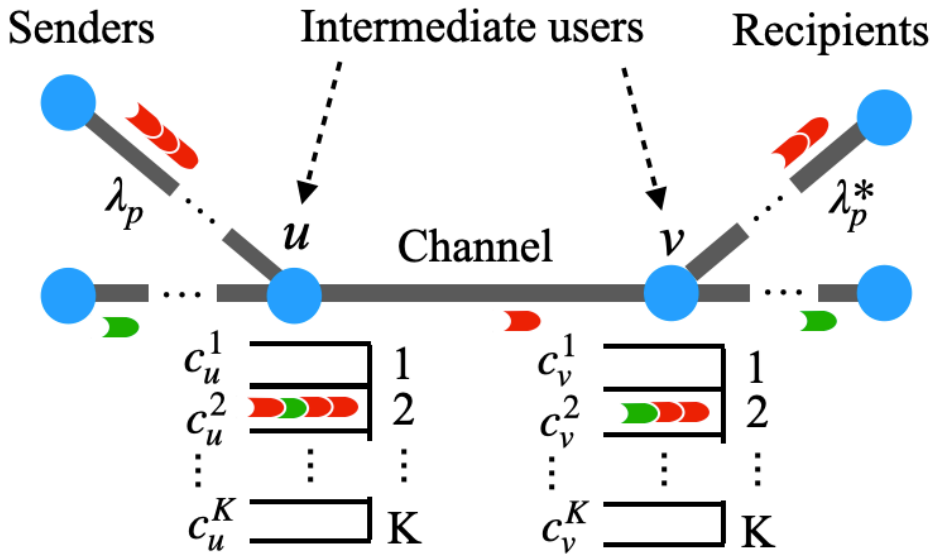


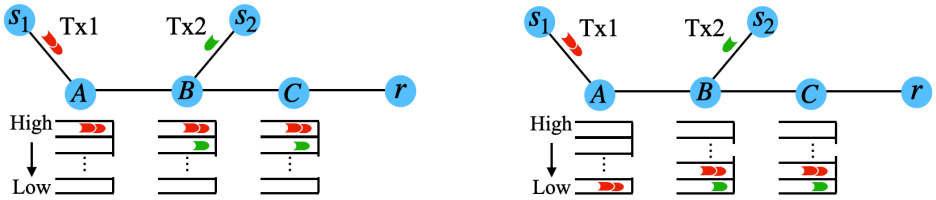
Figure 3.3: The illustration of the priority scheduling in PAPCN.

3.4.2 Performance objective

We consider a PAPCN as a direct graph $G = (V, E)$, where V is the set of all PCN nodes and E is the channel set. Each PCN node has at least one channel connecting to others. Assume that there are N sender-receiver pairs transactions in G. Each sender has a transaction sending rate λ_p, and they use their pre-specified path to route transactions. We consider the transaction sending rate of each sender is varying but fixed in a short time interval. In the PAPCN, intermediate user transactions in the forwarding queue can be partitioned into K different priority classes. We focus on each transaction flow with independently dynamic priority assignment schemes to analyze the transaction process.

For the transaction flow of pair p, we assume the transaction path is given as R_p which is a sequence of intermediate users (hops) over the routing path of pair p. c_h^k is the forwarding fee of the corresponding queue with priority k of hop h. x_h^{pk} is a binary priority assignment indicator with $x_h^{pk} = 1$ implying the transaction of pair p is assigned with priority k at hop h and $x_h^{pk} = 0$ otherwise. As shown in Fig. 3.3, the c_u^1 is the forwarding fee of the queue with the highest priority of intermediate user u. The capacity of the forwarding queue restricts the transaction forwarding rates of the intermediate user. As transactions compete for the forwarding capacity of each shared intermediate user, the transaction rate may gradually decrease along its routing path. Hence, there can be a bottleneck λ_p^* in the transaction rate for each pair of transactions: $\lambda_p \leq \lambda_p^*$.

We use f_h^p to denote the real forwarding fee that the transaction flow of pair p needs to be paid on hop h. It can be determined by c_h^k and the specified priority x_h^{pk}.



(a) Simply set priorities on each hop for transactions

(b) An efficient solution for priority assignment

Figure 3.4: Capacity competition challenge in the priority-aware PCN.

The total forwarding fee f_p can be calculated by: $\sum_{h \in R_p} \sum_k x_h^{pk} c_h^k, k \in [1, .., K]$. A transaction is only allowed to be assigned a unique priority on each hop. Then, we get a constraint on x_h^{pk}: $\sum_k x_h^{pk} = 1, k \in [1, .., K]$. In a crowded PCN, the sender who wants to speed up their transactions can increase the transaction priority to ensure they can be forwarded immediately but lead to higher forwarding fees. Additionally, senders can lower the forwarding fee and maintain lower transaction rates to save their coins. Therefore, we explore how to efficiently allocate forwarding fees for transaction flows with different rate requirements. The basic idea is to allow part of them to be forwarded quickly to achieve a higher TPS and enhance network performance. Our goal is to find a global priority scheme to maximize the gap between the achievable transaction rate λ_p^* and required forwarding fee f_p: $\max \sum_p (\alpha \mid \lambda_p^* \mid -(1-\alpha) \mid f_p \mid)$ where α is a weighted coefficient that satisfies the user preference.

Payers with high transaction rate requirements can augment the value of α to attenuate the impact of fees on decision-making. A big/small α indicates the system goal is to achieve a higher transaction throughput/a lower cost on forwarding fees. For example, when $\alpha = 1$, the goal is to find a maximum transaction throughput regardless of the cost. When $\alpha = 0$, the system prefers to find a minimal total forwarding fee regardless of the throughput. However, due to the privacy protection of PCNs, each transaction sender cannot get the instantaneous capacity information of the forwarding queue of each intermediate user. PCNs are fully decentralized, and there are no centralized servers for overall planning. Therefore, the achievable transaction rate λ_p cannot be directly obtained before the transactions are issued.

3.4.3 Challenges

Information restriction. Due to the privacy protection of PCNs, senders cannot track the status of their issued transactions to adjust the assigned priorities. The priority assignment should be completed before issuing the transaction. As

aforementioned, the path congestion information is crucial for senders to assign priorities to their transactions. However, transactions are unpredictable in highly dynamic PCNs. Senders cannot determine the definite initialization time and path information of other users' transactions. They are required to conjecture future states of each hop over their transaction path base on their knowledge and determine suitable priorities for forwarding fees assignment before the transaction is issued. Additionally, the security and privacy restrictions in PCNs make it impossible to dynamic planning priority assignment based on global information.

Capacity competition. In general, the sender who requires a high transaction rate pays more forwarding fees to compete for more forwarding capacity on shared hops with others. However, other senders may also raise their priority to ensure sufficient capacity, which causes all these transactions to gather in the highest priority queue. Transaction senders trends to assign high priorities for their transactions to achieve a high transaction rate like sender s_1 in Fig. 3.4(a). Obviously, it is inappropriate for s_1 to assign a high priority for the transaction flow *Tx1* on hop A which is uncongested. For example in Fig. 3.4(3.4a), if the high-priority queue of hop B, C are congested, sender s_2 can specify low priorities for the transaction flow *Tx2* on hop B, C rather than to assign a high priority for a weak improvement on transaction rate. After the priorities of *Tx2* are adjusted, sender s_1 can also adjust the priority assignment for *Tx1*. Then, we obtain a more efficient priority assignment solution as shown in Fig. 3.4b. Hence, senders should carefully make their decisions on transaction priority assignments.

While the reinforcement learning technique paves a way to solve the priority assignment problem with limited information. We propose a global transaction priority assignment scheme by developing a multi-agent reinforcement learning algorithm. To enhance the performance of PCNs, the proposed method needs to find an adversarial equilibrium between each transaction flow. None of them can obtain more benefits by breaking the balance. Once a transaction flow modifies its scheme, other transaction flows passively adjust their schemes at the same time to protect their profits. More details of the designed learning method are introduced in Section 3.5.

3.5 RL-BASED PRIORITY ASSIGNMENT SCHEME

In this section, we model our priority assignment problem and propose an RL-based transaction priority assignment scheme to enhance the performance of PCNs. The PCN is fully distributed, and each sender has a partial observation of the PCN environment and determines its priority assignment scheme to achieve a high transaction throughput. Therefore, we exploit to apply the multi-agent RL-

based scheme where each agent learns to find their appropriate priority assignment scheme by interacting with others to maximize their rewards.

3.5.1 Priority assignment problem formulation

A priority assignment problem can be considered as an N-pair transactions Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [46] consisting of a tuple $\mathcal{M} = \langle \Phi, \mathcal{S}, \mathcal{A}, \mathbb{R}, \mathcal{P}, \mathcal{O}, \gamma \rangle$. Each transaction session is a learning agent that maintains a transaction flow to interact with a dynamic PCN environment by sequentially specifying the policy on priority assignment in discrete time. At each time slot, a agent p selects an action a_p from a set of their priority assignment policies $\Phi_p \in \Phi$, jointing with other agents' actions, which can be expressed as $\vec{a} = \langle a_1, ..., a_p, ..., a_N \rangle \in \mathcal{A}$ for the current state $s \in \mathcal{S}$. Then, each agent p can make a partial observation $o_p = \mathcal{O}(s, p) \in O_p$ from the PCN and obtain their current observation of state s, where $\mathcal{O}(s, p)$ is a observation function :$\mathcal{S} \times \{1, ..., N\} \to \mathcal{O}$. The state transition of PCN depends on a probability $Pr(s'|s, \vec{a})$ that has a function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to [0, 1], \mathcal{P}(s, \vec{a}) = Pr(s'|s, \vec{a})$. γ denotes the discount factor, $0 < \gamma < 1$. When senders use the specified policies to route their transactions, each agent can observe the total forwarding fees and the influenced transaction rate to measure the reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Meanwhile, each agent learns, from its experience which contains each step observations and associated rewards, a policy $\pi_p(a_p|o_p) : O_p \times \Phi_p \to [0, 1]$ to conduct a joint policy $\vec{\pi} = \langle \pi(a_1|o_1), ..., \pi(a_p|o_p), ..., \pi(a_N|o_N) \rangle$. From [47], the goal of each agent is to find an optimal policy $\vec{\pi^*}$ to maximize its expected cumulative discounted rewards:

$$\vec{\pi^*}(o_p^0) \in \arg\max_{\vec{a_t} \in \mathcal{A}} \left[\sum_{t=0}^{\infty} \gamma^t r_p(o_p^t, \vec{a_t}) \right] \tag{3.1}$$

Fig.3.5 shows the agent-environment interaction in the priority assignment game. In this game, the state, and observation space, action space and reward are defined as follows:

State and observation space. The practical state s_t of PCN environment at time t, includes the behaviors of all agents and statistics (e.g. the current status of the forwarding queue and connected channels) aggregated from different nodes. However, most state information cannot be obtained or effectively used by each agent since it only observes the status of their transaction flow. In our proposal, the learning agent' s observation space is composed of local success rates and the forwarding fees information. The local success rate is a cumulative ratio of successful transactions and issued transactions. The $\hat{\lambda}_p^*(t)$ defines the status of

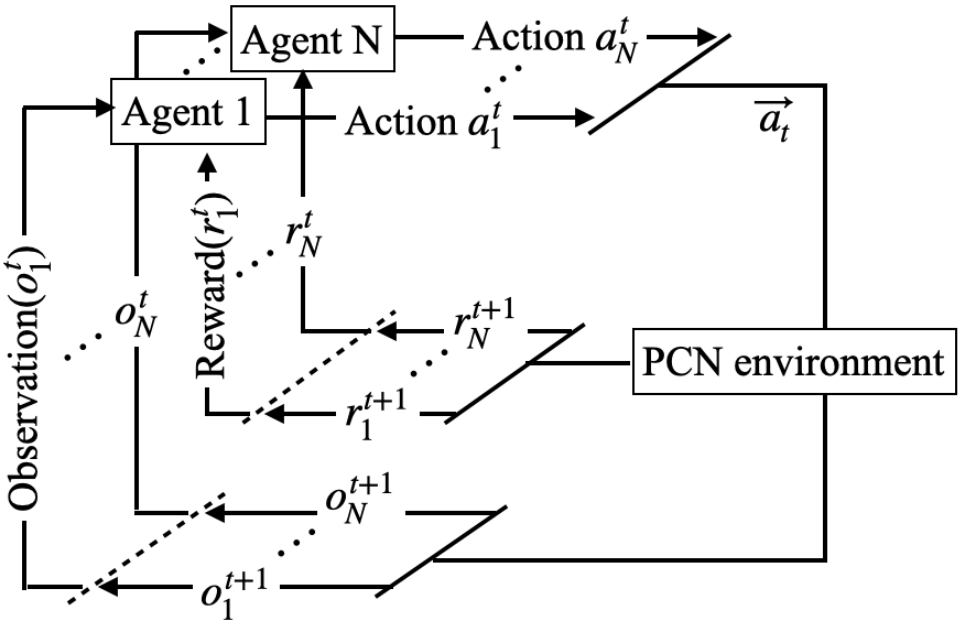


Figure 3.5: The interaction between multi-agents and the PCN environment in priority assignment game.

the local success rate of an agent, which can be directly measured at time step t. The learning agents can assign priorities accordingly, i.e., when the rate is lower than the requirement, the agent can assign high priorities on certain hops. The achieved transaction success rate is significant feedback to affects the decision of the learning agent. The forwarding fee information is the current forwarding fee paid to each intermediate user on the transaction path. In addition, although the total forwarding fee may be the same, the different fee allocation scheme for each hop leads to a different transaction success rate. For example, a 3-hops transaction with two fee allocation schemes: $\{1, 1, 2\}$ and $\{2, 1, 1\}$ may achieve a different transaction success rate. Hence, the observation function can be denoted as:

$$O(s_t, p) = \{\hat{\lambda}_p^*(t), \hat{f}_p(t)\}$$

where $\hat{f}_p(t) = \{f_p^1(t), ..., f_p^h(t)\}, h \in R_p$ is a vector of each hop' s forwarding fee at time step t. The time steps t includes the value 0. Moreover, the information on forwarding fees will be randomly initialized when $t = 0$.

General approaches handle the multi-agent RL problems by using independent Q-learning [48] in which the agent learns a discount return to measure an action-value function defined as Q-value. Q-learning is used for estimating the optimal Q-value function but restricted by its tabular property with no approximation, thereby combines with a deep neural network to handle the problems with large action space. However, in a multi-agent environment, each agent com-

petes for the capacity of the forwarding queue on the shared hop (intermediate user), which leads to a non-stationary issue for independent Q-learning since the behavior of each agent interacts with each other. Especially, this issue seriously affects the convergence of the learning process in the Deep Q-learning (DQN) with experience replay. A fingerprint-based method is proposed in [49] to address the non-stationary issue that the agent learns from the behavior information of other agents. The agent makes current decisions by estimating other agent' s decisions. However, due to introducing all DQN parameters of other agents brings a large-dimension observation space, [49] provide alternative methods that use a low-dimensional fingerprint. We adopt this method by appending a vector of the previous decisions of all agents to the observation space since it implicitly reflects the changing status of the environment. Therefore, the improved observation function can be expressed as:

$$O(s_t, p) = \{\hat{\lambda}_p^*(t), \hat{f}_p(t), \vec{\pi}_{t-1}\}$$

Action space. The action space is the decision of priority selection on each hop over the transaction path. An agent takes an action $a_p^t = \{x_p^{hk}(t)\}$ based on the current priority assignment policy at the beginning of each time step t. The sender sends transaction packets with the assigned priorities in the action of the current time step. For example, the learning method specifies a priority set $a_p^t = \{(1,0,0),(0,0,1),(0,1,0)\} \Leftrightarrow \{0,2,1\}$ as an action to route packet for the transaction flow of pair p with three intermediate users on its path at time step t. The first intermediate user receives the transaction packets and inserts it into the queue with priority 0. Similarly, the other two intermediate users insert it to the 2nd-level and 1st-level of their priority queues, respectively.

Reward. From the model, our objective function is to maximize the gap between the transaction rate and the cost of forwarding fees. In reward design, we tread to use an independent reward for each agent in our learning method. We also need to collect the successful transactions and track the success time of each transaction to obtain the actual transaction success rate of different actions in each time step since they are measured in discrete time. Then, the transaction rate $\lambda_p^*(t)$ of transaction flow p sent in time step t can be calculated. Note that the $\hat{\lambda}_p^*(t)$ and $\lambda_p^*(t)$ are different. The total forwarding fee $f_p(t)$ is easily calculated with a determined action of priority assignment. To better balance the impact of the two metrics, we use normalized rate and fee values. The weighted coefficients α is applied to reveal the user preference. To this end, the independent reward of

an agent at each time step t can be represented as:

$$r_p(t) = \alpha|\lambda_p^*(t)| - (1-\alpha)|f_p(t)|$$

The balance shifts with the value of α. While α is increasing, the learning method trends to find the minimized forwarding fee that guarantees the achieved transaction rate satisfies the requirement. In opposite, it trends to find the maximized achievable transaction rate under a certain forwarding fee.

3.5.2 Multi-agent DQN-based priority assignment scheme

Although traditional PCNs limit the number of hops on the transaction path to less than 20, some agents may have a high-dimensional action space since it corresponds to the hops on the transaction path. In this dissertation, we use the deep Q-learning method to solve our priority assignment problem because it is recommended for processing the action spaces with high dimension [29]. Hence, we develop a multi-agent DQN-based priority assignment algorithm to provide priority assignment policies as actions of each agent. To train our network, each agent selects an action at time t based on a policy $\pi(a_t|o_t)$ to get the reward $r(o_t, a_t)$. Define the expected total discounted reward $Q_p(o, a)$ as the value function of a state-action pair of the agent p with a given policy π, which can be expressed by:

$$Q_p(o, a) = \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t r_p(o_t, a_t)|o_1 = o, a_1 = a, \pi] \tag{3.2}$$

The goal of this agent is to find a policy which maximises the cumulative value of $Q_p(o, a)$ from the start state. Other agents can adjust their policy to protect their profits while it affects the entire environment as a result to decrease the value of $Q_p(o, a)$. The optimal policy for a multi-agent environment guarantees a maximized total reward under an adversarial equilibrium. Hence, combining with Eq.3.1 and Eq.3.2, we get the expected cumulative discounted reward of p :

$$V_p(\vec{\pi^*}) = \left[Q_p(o_p^t, a_p^t)|a_p^t \in \vec{a_t}, \vec{\pi^*} \in \arg\max_{\vec{a_t} \in \mathcal{A}} \sum_{i=1}^{N} Q_i(o_i^t, \vec{a_t}) \right]$$

Deep Q-learning supports Q-value approximation by applying a deep Q-networks (DQNs) [30] in which each agent p maintains their action-value function $Q_p(o_p, a_p, \theta_p)$ with a local DQN parameter θ_p. The DQN stores each agent's transition history $\langle o_p^t, a_p^t, r_p^t, o_p'^t \rangle$ into a replay memory $\mathcal{M}_p$. By sampling batches of B transitions

from $\mathcal{M}_p$, the action-value function is learned to minimize the *error* function:

$$\mathcal{L}_p(\theta_p) = \sum_{j=1}^{B} \left[(y_p^* - Q_p(o_p, a_p, \theta_p))^2 \right]$$

where $y_p^* = r_p + \gamma \max_{a'_p} Q_p(o'_p, a'_p, \hat{\theta}_p)$. $\hat{\theta}_p$ are the target network parameters intermittently updated by θ_p.

3.6 SYSTEM IMPLEMENTATION

The current Basis of Lightning Technology (BOLT) [50] describes the detailed Bitcoin off-chain protocol which guides several LN implementations such as Lightning Network Daemon (LND) [51], c-lightning. In this dissertation, we mainly focus on the LND, an out-and-out implementation of a lightning node, that uses the Lightning service to manage their channel link to transmit transaction packets between each Lightning node.

3.6.1 Lightning network daemon (LND)

We carry out the system in the *LND version 0.11.99-beta*, a GO version implementation of LN [51]. In the LND implementation, the recipient first generates a transaction request (*invoice*) which contains basic information of a transaction. The transaction request is then sent to the transaction sender as a parameter of the *sendpayment* function for path-finding and the transaction session initialization. For privacy protection, the transaction is encrypted to a *sphinx* onion packet composed of per-hop payloads by using the onion routing protocol in BOLT #4. Subsequently, the onion packet is dispatched to the first hop by using *sendHTLC*. When intermediate nodes receive an onion packet from their upstream channel, they verify the integrity of the packet and decode the packet containing the next-hop information and the internal payload. The internal payload is then queued from the *switch* module to the *mailbox* module. The *mailbox* uses a *mailCourier* to fetch the packets from the forwarding queue in order by the packet memory address. The fetched packet is then forwarded to the downstream channel until it reaches the destination.

3.6.2 priority-aware LND Implementation

The priority-aware LND implementation is divided into two stages: LND modification and learning algorithm implementation. We show the network structure

in Fig. 3.6 and elaborate on four parts (Transaction initialization, Switch and link module, MailBox module, and Forwarding policy) of LND modification as below:

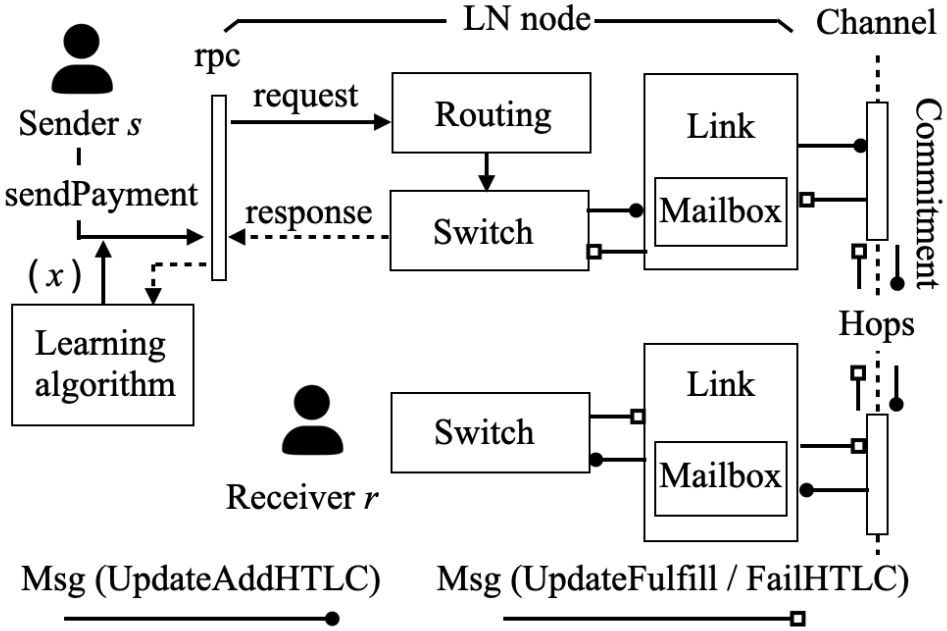


Figure 3.6: The network structure of PAPCN.

Transaction initialization: is the stage from the transaction request generation to the transaction attempt creation. Firstly, sender s uses *sendPayment* command to send a transaction to recipient r with basic transaction information $\{s, r, t\}$, where t is the expiry time. Combining with the current priority assignment action x provided by the learning module, the request is sent to the *rpc* server by using the updated *rpc* protocol. The *rpc* server calls the routing module to find a routing path R and register the transaction. The transaction amount F is then calculated through real amount v that the sender wants to pay and each hop's forwarding fee f^h: $F = v + \sum_{h \in R} f^h$. After that, the transaction attempt is created with the information $\{s, r, F, R, t, x\}$ and dispatched to the switch module. In the priority-aware LND implementation, we just use an available short path for transaction routing instead of implementing a new path-finding mechanism.

Switch and link module: The transaction attempt is first delivered to the *switch* module, a central messaging bus that handles transaction packet forwarding. The *switch* calls the forwarding policy of the connected link h to check if the forwarding fee f_h is sufficient, and then sends a *UpdateAddHTLC* message containing transaction packet to link h according to its priority. We update the structure of this message to include the priority information of the current hop. The channel link h receives the transaction packet with a new HTLC packet *htlcpkt* and current hop's priority x_h, then delivers it to the *mailbox* module of this link. The detailed

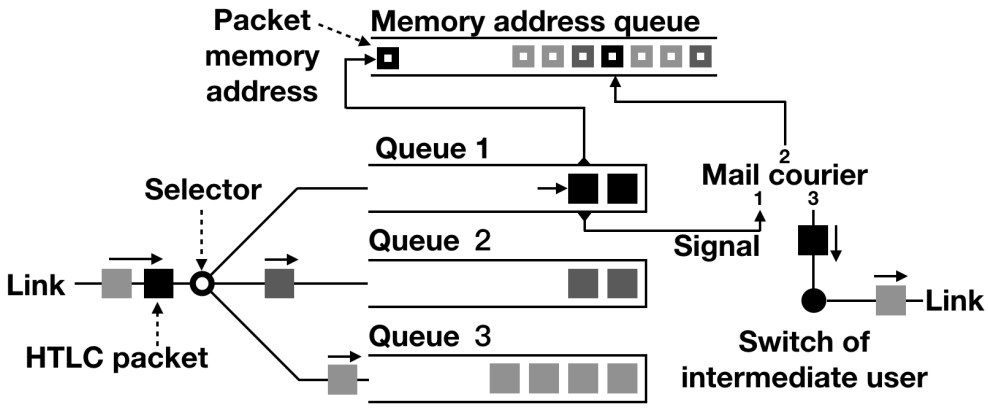


Figure 3.7: The priority implementation in mailbox module.

design and packet scheduling in the *mailbox* module is shown below. Packets are dequeued sequentially and waiting to forward to the next peer. After that, a new HTLC is established on the channel link h which requires a uniquely identifier called circuit key to identify this HTLC. We update the circuit key C_h to include x_h and record x_h in the payment descriptor which is used for channel commitment.

MailBox module: The *mailbox* module modification mainly focus on the structure of *mailbox* interface in *htlcswitch* package. In our implementation, a priority queue with 3-level priorities (high, middle, and low) is applied for observing the impact of priority on transaction forwarding in the LND. The priority was represented by the demand level in our implementation, which can be manually specified together with other basic transaction information or uses the lowest as default. We also adjust the relevant part of the source code to support priority delivery. The detailed process of transaction packets scheduling is shown in Figure.3.7. There are some forwarding transaction packets with different priorities on the link. Firstly, the selector delivers transaction packets to the corresponding priority queue based on its priority. The memory addresses of the assigned packets are returned and sequentially inserted into the packet memory address queue. At the same time, a signal is sent to the mail courier that there is an additional packet to consume. Mail courier catches transaction packets by packet memory addresses. The order is from high priority to low priority and first in first out for same priority. The signal wakes the mail courier to adjust the access location (pointer) and deliver the packet. When the packets in the high priority queue are exhausted, the mail courier turns to fetch the packets from the lower priority queue.

Forwarding policy: In traditional LN, the intermediate user can customize the charged forwarding fee that generally depends on two factors that are base fee and fee rate. While the forwarding fee is extended to the fees associate with different

priority classes in PAPCNs. As the priority raising, the forwarding fee is gradually increasing. Similar to the original information broadcast mechanism in the PCNs, the forwarding fee of a regular user can be broadcast to all users to update their local knowledge through the gossip protocol [52].

3.6.3 Learning Algorithm Implementation

The learning algorithm is implemented in Python code. We launch an independent learning algorithm process for each transaction sender to periodically provide the action for priority assignment. We build the interface for the learning algorithm to supply the current priority set (action) for *sendpayment* request and receive its response as the feedback of the PCN environment for state analysis. As aforementioned, the state of the learning algorithm is the current transaction success rate, current forwarding fees allocation, and the last decision of all other senders. The transaction result can be applied to measure the current transaction success rate. The allocation of current forwarding fees is known at the end of the last step. We assume senders are geographically close and connected by a peer-to-peer network with low latency. Payers share the previous action on priority assignment to collaborate with others by broadcasting the action's index. Meanwhile, the learning algorithm tracks the transactions affected by each action and obtains corresponding rewards to build new MDPs, which will be used to train the learning model. In practice, transactions can be intermittent between two peers. Some users cannot provide the current round decisions for the next round of training since they might be offline when they have no transactions. To address this issue, we give a simple solution that extends the action space with a consensus field to indicate the offline status of cooperators in the system. When a sender is offline, cooperators use this field to construct the observation space.

3.6.4 Security analysis

We first build the attack model in which we consider an efficient attacker can spawn users and use them to pretend to be normal nodes and forward transactions. We then identify the security and privacy notions of interest:

- ***Priority value security.*** Priority value security ensures that the attacker cannot change the priority values on the honest users. This goal can be achieved since the priority information of each hop is separated and encapsulated in the onion routing package.

- ***Priority assignment privacy.*** The leakage of priority assignment information can give an adversary some advantages in channel resource com-

petition. However, the adversary also needs to know the transaction path information to carry out the attack.

In our model. The attacker has two ways to attack the transaction process by changing the priority:

First, we assume the attacker is an intermediate node that can freely schedule the transaction packet and insert it into any priority queue. As an attacker, the node may charge the forwarding fee but leave the packet in a low-priority queue. On the sender' s side, the learning method tries to assign different priorities to each intermediate node. The transaction results reflect the feedback on the priority assignment policy. It learns from the transaction result that the achievable transaction rate is lower and doesn't change with priority. The learning method will assign the lowest priorities on each hop for cost-saving. Then, the attacker cannot benefit from the transaction forwarding.

If the attacker is outside of the path. To protect the security and privacy of transactions, the LN applies the onion routing protocol into transaction execution [17]. Furthermore, source routing is utilized to allow transaction senders to complete control of the transaction path within the LN. These methods decrease the risk of the release of transaction path information so that the attacker cannot track the transaction of a specified sender. If an external attacker changes the priority of a transaction package, the transaction will fail due to a mismatch in the priority and the corresponding fee. By analyzing the failure message, the sender can know the existence of the attacker. A simple way for senders to protect their transactions is to change the routing path.

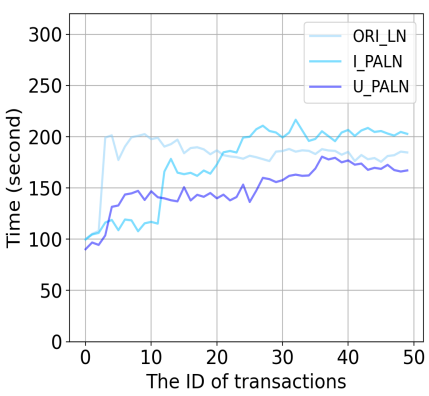
3.7 Performance Evaluation

Our experiments are composed of two parts: 1) we build a testbed under the *simnet* mode of LND and show the performance of our learning approach in a small-scale PCN; 2) we develop a Python-based simulator to simulate the network layer of the LN, and then show the performance of the proposed multi-agent DQN-based priority assignment algorithm.

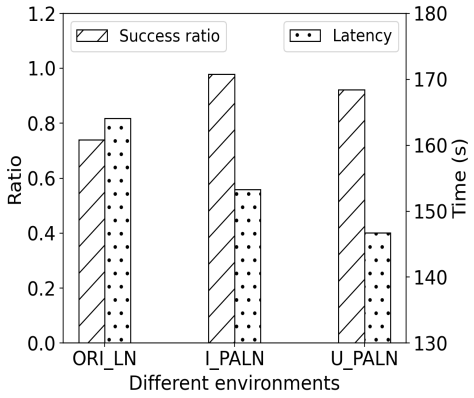
3.7.1 Testbed

Settings

We set up two small-scale PCN environments: Original Lightning Network (ORI_LN) and Priority-aware Lightning Network (PALN). To create the wallet and establish the network, the *LND version 0.11.99-beta* [51] is applied. We use a



(a) The change in transaction latency of a certain transaction flow in different environments.



(b) The transaction success ratio and average latency comparison in different environments.

Figure 3.8: The experimental results in the real lightning network daemon.

Watts-Strogatz small-world topology [33] with 18 nodes and 36 payment channels. We specify ten pairs of sender-to-recipient transaction flows and guarantee that each flow shares payment channels with others. The transaction sending rate of each flow is fixed and within the range [2, 8] transactions per second (Tx/s) in our experiment. In the PALN environment, we test two priority scheduling methods: 1). *Intelligent approach* (I_PALN). Senders use our learning-based algorithm to obtain the priority setting for their transaction flows. In our experiments, we first train our model in the developed simulator (Section 3.7.2). The training results are applied to assign the priority for each transaction flow with a static transaction sending rate in the built PALN environment. 2). *Unbiased method* (U_PALN). Senders assign the priority of each hop along the routing path with no bias for their transaction flows.

Results

We first track the latency of a certain transaction flow in these environments and show its changes in Fig. 3.8a. The transaction latency gradually stabilized and was different under different environments. It reveals that the priority scheduling method affects the transaction latency. Although the transaction flow in learning-based PALN reaches a higher transaction latency, the average transaction latency of the entire network is decreased. Fig. 3.8b shows the transaction success ratio and average transaction latency in different environments. The success ratio of transactions in the learning-based priority-aware PCN is 97.8%, while it only reaches 74% in the traditional PCN. Even if there is no bias for priority assignment in the priority-aware PCN, the success ratio of transactions can reach 92.2%. In our learning-based PALN, the average transaction latency is about 146.7s, which

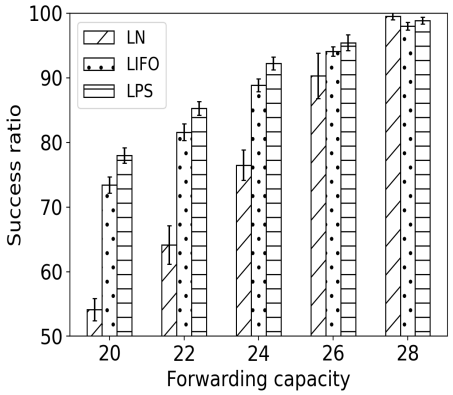
rises to 164.05s in traditional LN. Compared with the traditional LN, our implementation achieves a lower average transaction latency and higher transaction success ratio. The reason is that high-priority transaction packets will be scheduled first leading to a lower delay. Meanwhile, it increases the probability that the transaction packets can be scheduled before the transaction expires, thereby improving the transaction success ratio.

3.7.2 Simulations

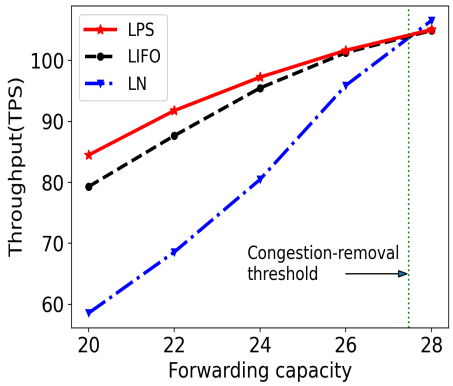
Settings

We have developed a Python-based real-time simulator to evaluate our multi-agent priority assignment algorithm. In the simulation, we set up a PCN environment with a small-world topology consisting of 50 nodes and 100 payment channels. We create 3 priority classes whose forwarding fees are $\{8, 4, 2\}$, respectively. In practice, we can specify more priority classes to achieve more precise scheduling for different user needs. Each user has a fixed transaction processing rate (forwarding capacity) of 20 Tx/s. The per-hop delay is set to 20ms [24]. There are 15 pairs of sender-to-recipient transactions with dynamic transaction sending rates. We assume the transaction path is fixed and specified by its sender. The length of the transaction path is within the range from 4 to 8 and the average value is 6, which is similar to the real LN [53]. The initial transaction rates are randomly selected from the range [4,9] (Tx/s). The changing of transaction rate conforms to the normal distribution. The expiry time of each transaction flow is manually specified, which depends on the length of the routing path and user demands. For example, the expiry time is set to a short time like 1 for a transaction with 5 hops on its path.

Each sender launches a learning process for priority assignment to balance their TPS and total forwarding fee by using our multi-agent DQN-based priority assignment algorithm. The action refresh time interval is 1s. The batch size, discount factor, and learning rate are128, 0.99, and 0.0001, respectively. ε has an initial value of 0.5 and linearly increases to 0.9 with a step $1e-5$. The algorithm selects a new priority sequence for each transaction flow per second. A replay memory with a fixed capacity can store 10000 records of MDP. We set checkpoints in our algorithm to check the staged cumulative rewards. Each checking period is about 100 seconds. A check point is a flag that inspects the training process and stores priority assignment transition tracks.

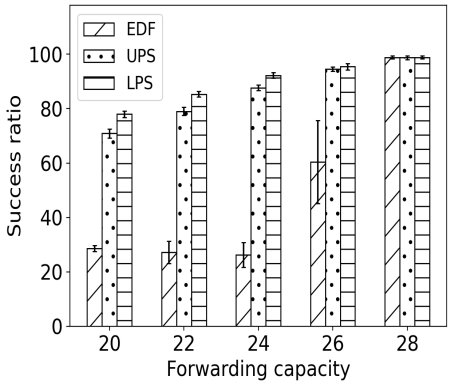


(a) The success ratio comparison in steady-state.

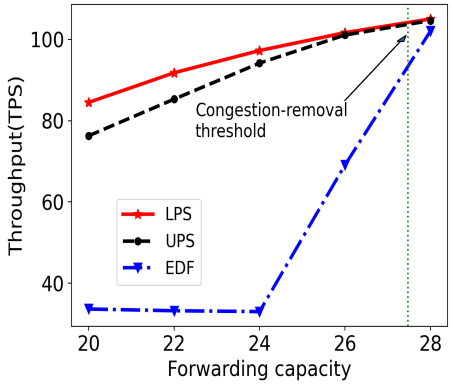


(b) The transaction throughput comparison in steady-state.

Figure 3.9: The two metrics comparison with different scheduling methods.



(a) The success ratio comparison in steady-state.



(b) The transaction throughput comparison in steady-state.

Figure 3.10: The two metrics comparison with different priority assignment algorithms.

Scheduling methods and algorithms comparison

Firstly, we compare our learning-based priority scheduling (LPS) method with other scheduling methods: 1) General scheduling method in the current lightning network (represented by LN in figures), and Last-in First-out (LIFO). Comparison mainly focuses on the two important metrics: network throughput and success ratio. Comparing with original algorithm and other scheduling method, our LPS method achieves a higher success ratio and network throughput than other scheduling methods that shows in the Fig. 3.9(a) and Fig. 3.9(b). The LPS method can control the rate of certain transaction flows by adjusting their priorities assigned for the intermediate users on their path, thereby allowing more transactions to achieve a higher transaction rate. For example, a transaction flow shares different channels with the other two transaction flows on its path. When

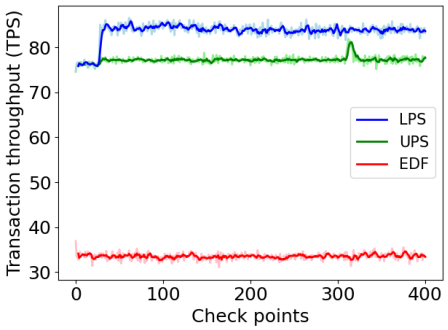


Figure 3.11: The comparison on transaction throughput by applying different scheduling methods.

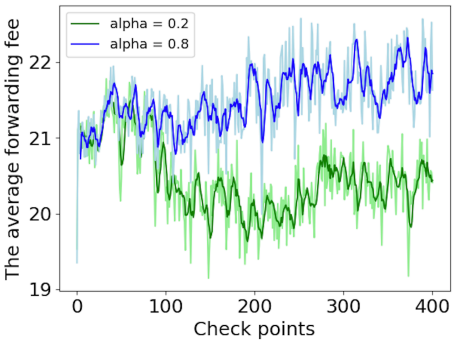


Figure 3.12: The comparison on the average forwarding fee under different weight coefficients.

the flow has a demand to save coins by lowering the priorities assigned to each intermediate user, the occupied forwarding capacity of this flow in the queue corresponding to the previous priority will be released. In contrast, the other two transaction flows may utilize the released forwarding capacity to reach a higher transaction rate as well as improves the network throughput. Additionally, we compare our learning-based priority assignment algorithm with other two priority assignment algorithms, e.g. unbiased priority scheduling (UPS) and earlier deadline first (EDF) in [24]. The simulation results are shown in Fig. 3.10(a) and Fig. 3.10(b). Compared with other priority assignment algorithms, our LPS algorithm achieves higher transaction throughput and success ratio, especially in the case of network congestion. We show the change in transaction throughput by applying different scheduling methods in Fig. 3.11 when the forwarding capacity is fixed to 20 Tx/s.

Learning algorithm evaluation

To achieve a higher transaction rate and a lower expenditure on forwarding fees, we appropriately reconfigure the parameters in our multi-agent DQN-based priority assignment algorithm. We show the change of average forwarding fee of the entire network under different values of the weighting coefficient α within 400 checking periods in Fig. 3.12. We notice that the average forwarding fee of the entire PCN is decreased when the α has a small value. With the value of α increasing, the lowest average forwarding fee will be higher. Hence, the value of α can be used to meet different user demands on transaction throughput and forwarding fee costs. The Fig. 3.13(b) and the Fig. 3.13(a) show the achieving rewards of our learning algorithm under different values of α. We find the achieving reward is relatively stable as the value of α increases. The reason is that priority assignment can directly affect forwarding fees which fluctuate in a discrete set. In

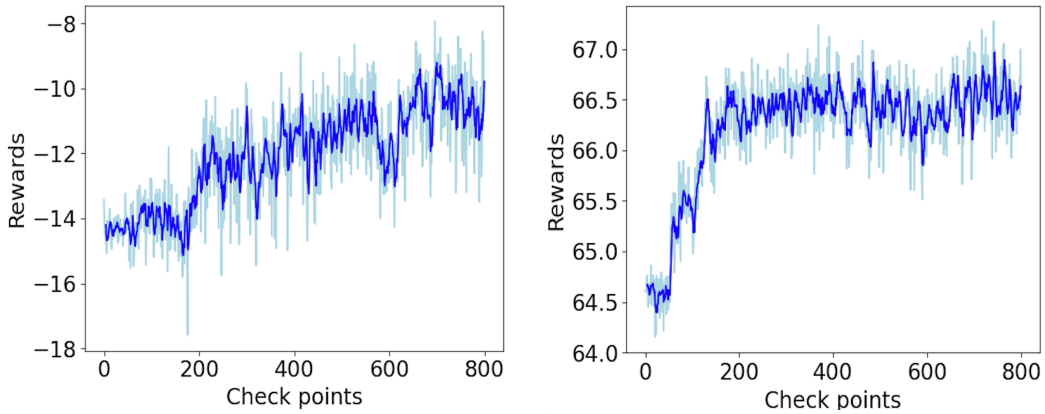
contrast, throughput gradually stabilizes.

From the figure of reward evaluation, the system achieves a long convergence time of about 5 hours. Here we give some reasons: 1) The transaction sending rate of each sender is varying (normal distribution). An unstable rate brings challenges to the dynamic priority decision of each hop along the transaction path. 2) Each sender needs to assign a priority to each hop along the transaction path. The combination of different priorities brings up a huge decision space resulting in a high solution searching time. 3) We set a very small learning rate (0.0001) for the learning process. In the context of machine learning, a model with a too-small learning rate would be a slow learner and it would need more iterations. 4) Every node is greedy and will compete for the forwarding capacity with other transaction senders. A bad case is that all senders assign high priorities to their transactions for fast scheduling on intermediate nodes. All of their packets will be aggregated in the high-priority queue that offsets the benefit of priority scheduling.

As we know, machine learning methods require large amounts of data to train their model. In our proposal, the training data is generated in real-time as the transaction proceeds. In the initialization phase of the algorithm, the learning method needs to collect the transaction data for model training. The speed of training data generation is related to the frequency of transaction requests. In practice, some nodes (like banks, markets, and exchanges) with frequent transaction requests can collect a large amount of transaction data in a short time. The learning model of these nodes may quickly converge. For other nodes without frequent transaction requests, some training data needs to be prior collected. This is also the challenge that methods using reinforcement learning algorithms face at the beginning of training. Furthermore, some nodes in the network can cooperate to form a stable group. The entire network can be divided into many groups. The overall optimization will be split into optimizations within each group. New nodes or pairs of transactions only affect the group they belong to. At the same time, node grouping also speeds up the convergence of the model.

Priority expansion evaluation

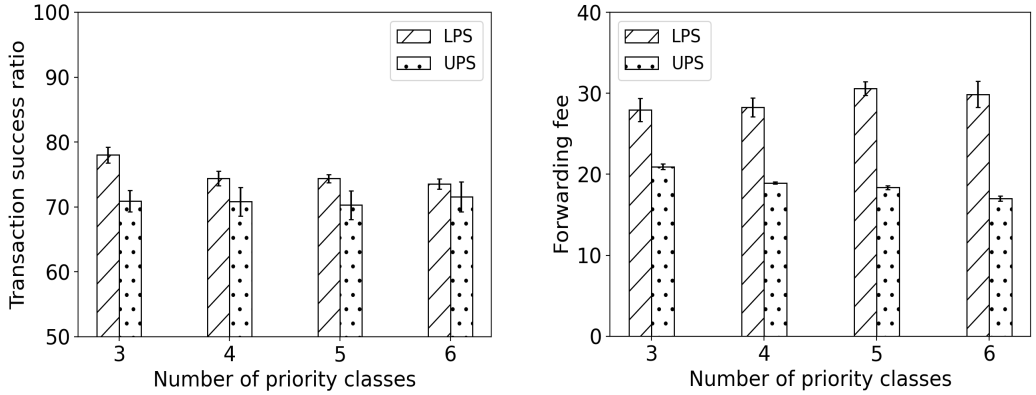
We expand the number of priority classes to $\{4, 5, 6\}$, respectively. The forwarding fee of each intermediate user is in the range $[2, 8]$ and increases with the priority increasing. For example, the forwarding fees of each intermediate user in the PCN with 4 priority classes are $\{2, 4, 6, 8\}$. We compare our learning algorithm with the unbiased priority scheduling (UPS) algorithm. The simulation results are shown in Fig. 3.14. In Fig. 3.14(a) and Fig. 3.14(b), we find that the success ratio of the entire network gradually decreases and the average forwarding



(a) The rewards in each checking period when $\alpha = 0.2$.

(b) The rewards in each checking period when $\alpha = 0.8$.

Figure 3.13: The rewards of the learning method under different weight coefficients.



(a) Transaction success ratio comparison of the entire network.

(b) The average forwarding fee comparison of the entire network.

Figure 3.14: Different metrics comparison under the different number of priority classes.

fee of the entire network is slightly increased as the number of priority classes increases. Due to the network congestion, transactions belonging to the same transaction flow will also compete with each other for the forwarding resources of intermediate users, resulting in a lower transaction success ratio. As the selectable priority classes increase, the learning algorithm can accurately specify appropriate priorities for the transactions. This leads to the priority selection of concurrent transactions on the shared channel to be scattered and increases the corresponding forwarding fees.

3.8 Summary

In this work, we observe the transaction time jitter in LND *simnet* which confirms the impact of concurrent transactions on transaction rates. To meet different transaction demands on transaction rates, we propose a priority-aware PCN for efficient transaction scheduling to achieve a high transaction throughput. In particular, priority assignment is essential to improve the utilization of the forwarding fee for a transaction flow. We further propose a multi-agent DQN-based priority assignment algorithm and develop a real-time simulator to verify the efficiency of our learning algorithm. The simulation result shows that our priority-aware scheduling method can achieve a higher transaction success ratio and network throughput when network congestion occurs in PCNs. We define a regulatory factor to meet different user demands on transaction rate and forwarding fees. Users can balance the trade-off between the achievable transaction rate and forwarding fees by changing the factor value.

Chapter 4

Multi-Branch Routing Mechanism

4.1 INTRODUCTION

In PCNs, payment channels are established between parties with a direct peer-to-peer connection and their deposits as the channel capacity. Off-chain payments between two un-directed parties are forwarded through a tolerant routing path composed of payment channels and intermediate parties. Concurrent payments can share payment channels on their routing path [12]. Due to the limitation of the channel capacity, payment failure incurs when the payment amount surpasses the channel capacity on its routing path. A failure message is then generated and sent back to the payment sender. Meanwhile, all of the established contracts by the payment are revoked immediately. It raises several problems like delaying payments and overhead on contract establishment and revoking, which leads to lower throughput. Additionally, the payment sender needs to find another routing path to route the failed payment. The new path may partially overlap with the old path. The path finding process and contract reconstruction bring additional overhead, thereby decreasing the network performance.

To achieve higher throughput in PCNs, Sivaraman *et al.* [24] propose *Spider* which splits payments into payment units. *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.* [31] 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 [22, 54], the route construction also depends

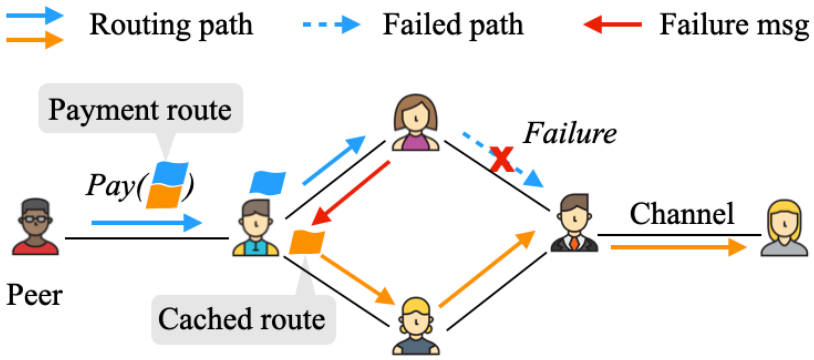


Figure 4.1: The illustrative example of our multi-branch routing system.

on the probe while the validity of time-sensitive probed information cannot be promised in highly dynamic PCNs. Bagaria *et al.* [55, 56] present redundant path schemes to route payments over multiple paths, but these schemes bring redundant resource occupation and require high collateral [57]. Existing works either delay payments or bring extra traffic loads. Moreover, they lack the reaction mechanism for payment failures.

In this chapter, 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 through 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 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 [58,59]. 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) [60] use an onion routing protocol to protect user privacy in payment routing. The Basis of Lightning Technology (BOLT) reveals the specification of LN [50]. 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 proposed 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 chapter is shown as follows. We first present path overlap issues and elaborate on our motivation in Chapter 4.2. In Chapter 4.3, we give an overview of the system design and describe our system model. Chapter 4.4 details the Markov chain-based approximate algorithm for path selection. The

system implementation is described in Chapter 4.5. We conduct experiments in Chapter 4.6 to evaluate the performance of the proposed scheme. Chapter 4.7 concludes this work.

4.2 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 reach the destination. To guarantee the atomicity of payment, a contract called Hash Time-Lock Contract (HTLC) is proposed. The contract is established on the payment channel along the payment path, at the same time partial channel deposit is locked for the associate payment. In highly dynamic PCN, concurrent payments compete with channel funds on shared payment channels leading to payment failure. All of the established contracts shall be canceled. To complete the payment, the sender needs to research a now available path for resending. It brings overhead on pathfinding and contract reestablishment 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 [55, 56]. But it requires high collateral [57] 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 [24], 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.

By analyzing the routing mechanism of LND, we find that it deploys a developed Dijkstra [52, 61] algorithm for pathfinding. Considering the case of payment failures, senders actually route their payments by switching between the shortest paths and a set of short paths with similar lengths. These paths may overlap, causing failed payments to transfer through certain channels repeatedly. To verify the conjecture, we explore the current LN topology and sample a set of short paths between the specified payer-payee pair. In Fig. 4.2, we show the overlap ratio of

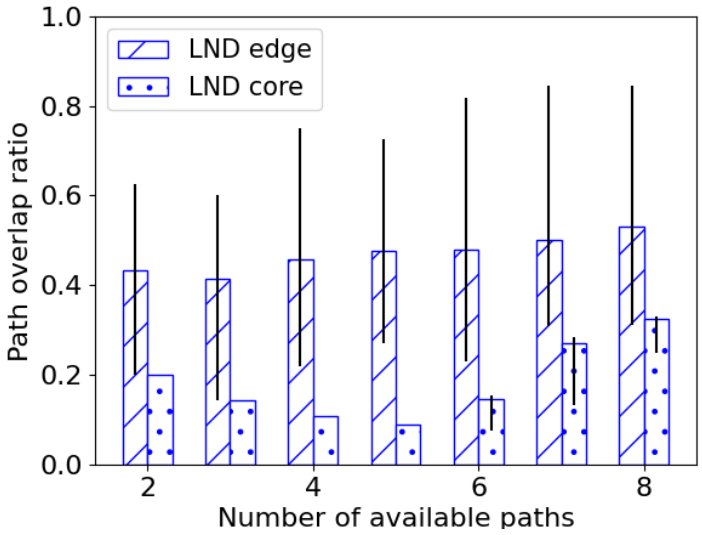


Figure 4.2: The change in the path overlap ratio under different numbers of available paths within different positions of LN.

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.

We attempt to propose a novel routing mechanism to prevent the payment from failure as well as solve the path overlap issue. Inspired by the restoration approaches in [62, 63], 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 [50]. 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

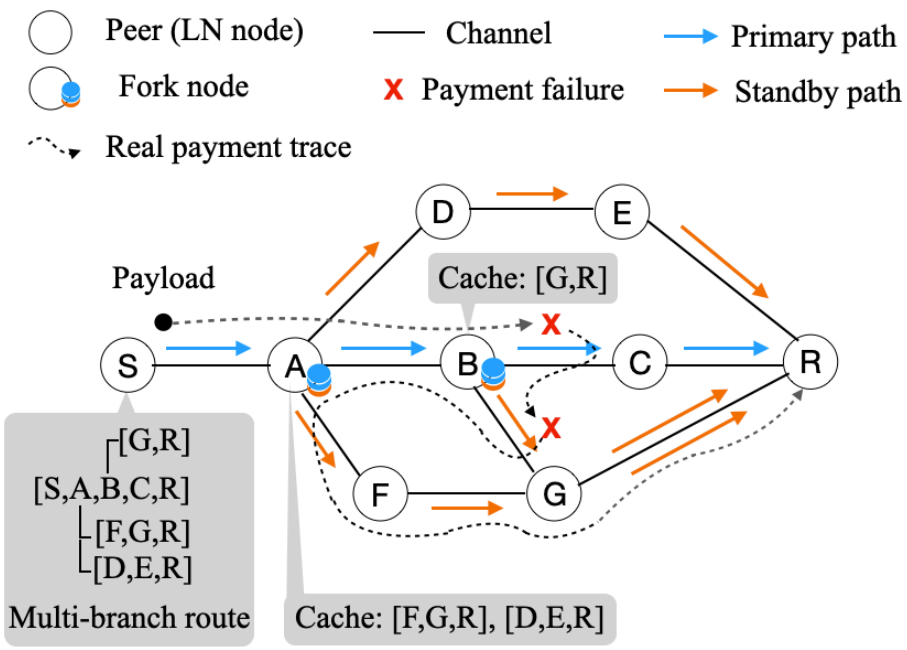


Figure 4.3: The system overview.

payment failure message to activate.

4.3 SYSTEM OVERVIEW AND MODEL

4.3.1 System Overview

In this section, we make an overview of the design of the multi-branch routing mechanism. As shown in Fig. 4.3, 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 route should be pre-processed to build a multi-branch 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 paths as secondary veins leading to the leaf apex. The sender applies the multi-branch 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 multi-branch route can have 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 Fig. 4.3, 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 is described in Chapter 4.5.

4.3.2 Assumptions

We consider each channel in the PCN to have 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 chapter, 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.

4.3.3 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 Fig. 4.3, 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, senders prefer their transactions 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 4.5. 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.

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.

4.3.4 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 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 our multi-branch routing mechanism is to find an 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:

$$x_s^p = \begin{cases} 1, & \text{if the candidate path } p \text{ is selected by} \\ & \text{session } s \text{ as a reserved path.} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Then, we define a 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: $(x_s^p - 1)y_s^p = 0, p \in R_s$ 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: $\sum_{p \in R_s} x_s^p \cdot H(|p|, y_s^p) \leq \Delta$ 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 of standby paths: $\sum_{s \in S} \sum_{p \in R_s, i \in p} x_s^p \leq B_i$. 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 Chapter 4.5. We ignore the impact of fee limitation on the path selection.

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, this probability is related to the payment amount z_s, denoted as $\pi(z_s)$. The payment success probability of session s can be represented as:

$$\pi_s = \sum_{p \in R_s} \mathbb{E}(x_s^p \cdot \pi_s^p(y_s^p)) \tag{4.2}$$

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:

$$l_s = \sum_{p \in R_s} \mathbb{E}(x_s^p \cdot l_s^p(y_s^p)) \tag{4.3}$$

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

$$\phi = \max \sum_{s \in S} (\pi_s - \alpha \mid l_s \mid) \tag{4.4}$$

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

4.4 DISTRIBUTED MULTI-BRANCH ROUTING ALGORITHM

The path selection problem with resource limitation is NP-complete [64, 65]. 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.

4.4.1 Log-sum-exp Approximation

Let $\mathcal{F}$ be a set of all feasible configurations for the MBR problem. A payer can obtain the local performance $\phi_s(f)$ of his payment session s under a given solution f. Then, the system objective function can be computed by aggregating the performance of each payment session: $\sum_s \phi_s(f), (s \in S)$. We use π_f to denote the percentage of time that the available solution f is in use. By adopting the approximation approach proposed in [66], our MBR problem can be approximated as:

$$\begin{aligned} \max \quad & \sum_{f \in \mathcal{F}} \pi_f \sum_{s \in S} \phi_s(f) - \frac{1}{\beta} \sum_{f \in \mathcal{F}} \pi_f \log \pi_f \\ \text{s.t.} \quad & \sum_{f \in \mathcal{F}} \pi_f = 1 \end{aligned} \tag{4.5}$$

where β is a positive constant. The approximation approach introduces an entropy term $-\frac{1}{\beta} \sum_{f \in \mathcal{F}} \pi_f \log \pi_f$ with an enhance approximation gap bounded by $\frac{1}{\beta} \log \mathcal{F}$. As the value of β increases, the approximated function of our MBR problem becomes more exact. We use $\pi_f^*, f \in \mathcal{F}$ to represent the optimal solution of the

approximated function, which can be derived via solving the Karush-Kuhn-Tucker (KKT) conditions [67] and expressed as:

$$\pi_f^* = \frac{\exp\left(\beta \sum_{s \in S} \phi_s(f)\right)}{\sum_{f' \in \mathcal{F}} \exp\left(\beta \sum_{s \in S} \phi_s(f')\right)}, \forall f \in \mathcal{F}. \tag{4.6}$$

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

4.4.2 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 $\pi_f^*, f \in \mathcal{F}$. The transition between 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_f^*. To describe the transition process intuitively, we use a non-negative value $q_{f,f'}$ to denote the transition rate between the two configurations f and f' and set it to zero, unless the two configurations satisfy the two conditions: 1). $\mid f \cup f' \mid - \mid f \cap f' \mid = 2$. 2). $f \cup f' - f \cap f' \in P_{\hat{s}}$. 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 $\pi_f^* q_{f,f'} = \pi_{f'}^* q_{f',f}, \forall f, f' \in \mathcal{F}$ needs to be satisfied.

For the two direct-connect configurations (f, f'), we let the transition rate $q_{f,f'}$ and the difference in system performance be positively correlated. From [66, 68], the transition rate can be expressed as:

$$\begin{cases} q_{f,f'} = \omega \exp\left(\frac{1}{2}\beta \sum_{s \in S}(\phi_s(f) - \phi_s(f'))\right). \\ q_{f',f} = \omega \exp\left(\frac{1}{2}\beta \sum_{s \in S}(\phi_s(f') - \phi_s(f))\right). \end{cases} \tag{4.7}$$

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

4.4.3 Distributed Markov Chain Based Routing Scheme

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,

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($\phi_s(f')$) signal with local performance $\phi_s(f)$ to other payers
    end if
    /*`Listen to RESET Signals`*/
    if a payer receives the RESET($\phi_s(f')$) signal **then**
        $\phi_s(f) \leftarrow \phi_s(f')$
        refresh and reset the timer T_s
    end if
end while

Algorithm 2 `Initialization()`

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

launches a processing-thread for s on 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'

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

$$\frac{\omega \exp\left(\frac{1}{2}\beta \sum_{i \in S}(\phi_i(f) - \phi_i(f'))\right)}{\mid R_s \mid \cdot(\mid P_s \mid -1)} \tag{4.8}$$

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 [66], 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 the different amounts 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 a payment and assume that the payment success probability of two paths is π_1 and π_2, π_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 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 f', the payer can trigger a timer T_s with an exponentially distribution for the corresponding payment session s with mean value: $\omega \exp\left(\frac{1}{2}\beta \sum_{i \in S}(\phi_i(f) - \phi_i(f'))\right) \cdot (\mid R_s \mid \cdot(\mid P_s \mid -1))^{-1}$. The payer then broadcasts the RESET($\phi_s(f')$) signal carrying the local performance $\phi_s(f')$ 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 timers of his payment sessions invoking 4.8.

The convergence of the proposed algorithm can be proved according to the algorithm analysis in [66, 68]..

4.5 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.

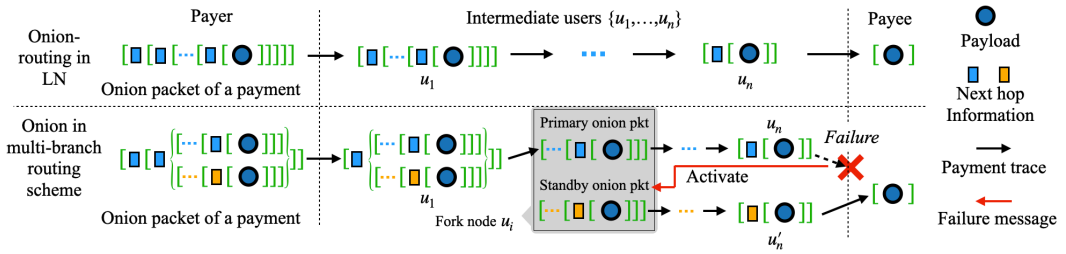


Figure 4.4: The difference of the onion-routing between our multi-branch routing mechanism and LN.

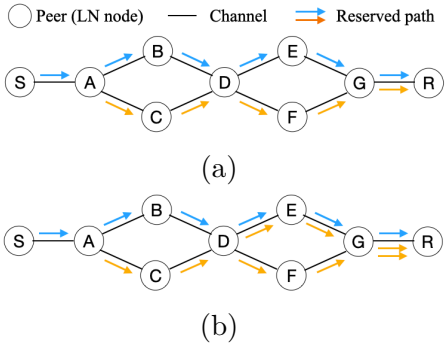


Figure 4.5: Two special cases in onion-routing.

4.5.1 Onion Routing

To protect the security and privacy of payments, the LN applies the onion routing protocol to payment execution. 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 shown in Fig. 4.4. 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 Fig. 4.4, the information of 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. **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 [52]. 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 [12]. 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.

Multi-branch onion packet. To build the multi-branch onion packet, the sender first prepares reserved paths and prunes their overlapping hops. Reserved path preparing 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 shown in Fig. 4.5a, 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, 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 Fig. 4.5b. Here, the fork nodes are nodes $[A, D]$, the standby paths are $\{[C, D, E, G, R], [F, G, R]\}$.

4.5.2 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 Fig. 4.6. 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 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,

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 prepare. $fees$ 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 $\mathsf{Tx}(R', B, V, \Delta T, fees)$.

Send: the packet to the first hop.

Figure 4.6: The payment initialization process in the pay routine of our multi-branch routing scheme.

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 Fig. 4.7. Upon receiving an onion packet, Intermediate users parse it to get the corresponding hop information and the inside 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.

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 4.7: The decision making on the intermediate user when an upstream Tx packet or downstream Msg is received.

4.5.3 Security analysis

We first build the attack model in which we consider an efficient attacker can spawn users and generate massive transaction flow to clog channels.

We then identify the security and privacy notions of interest:

- ***Forwarding security.*** 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 [69].

- ***Path selection privacy.*** The leakage of path selection information can give an adversary some advantages in path planning and implementing clog attacks. Due to PCN applies source-routing in transaction process, the path information will be encapsulated in the transaction packet. To protect the path selection information, we upgrade the onion packet to carry out the information on standby paths.

4.6 PERFORMANCE EVALUATION

To evaluate our multi-branch routing mechanism, we first develop a simulator for PCNs. The details of our simulation are described in 4.6.1. We then present the experimental results under different settings and make a comparison with other routing schemes.

4.6.1 Settings

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 a node 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.

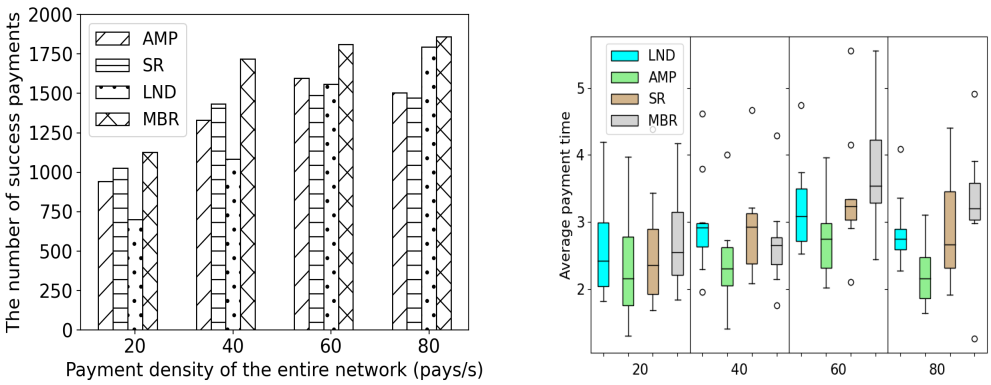
In our simulation, we use an intel Core i7-10700 CPU @2.90GHz $\times$ 16 to run our python-based multithreading simulator. We assume that CPU computing power will not drop when simulating small-scale networks. Task processing in each independent thread is highly parallel. By suspending the execution of threads, we can approximate network communication delays. The computer has the same initial state when we run each routing algorithm.

Benchmarks: We implement five 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.

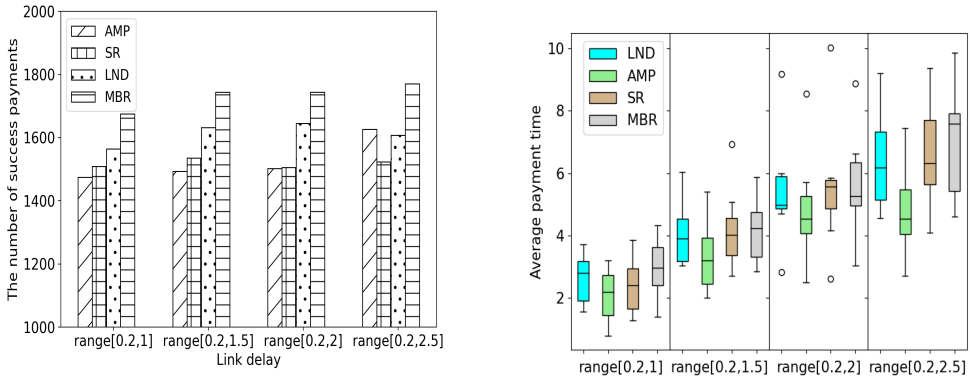
(b) Average payment time.

Figure 4.8: 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 transaction database.

SR: Split Routing [31] 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.

Topology and payments: We clip a small-scale network topology from the main network of LN 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 $[2, 7]$ with a mean equal to 5. The initial balance and delay of a channel are normally distributed in the range $[80, 160]$ 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. Different from the edge-disjoint paths used in [24], the candidate paths in our experiment have overlapping edges. The weighted coefficient is set to 0.5.



(a) Successful payments.

(b) Average payment time.

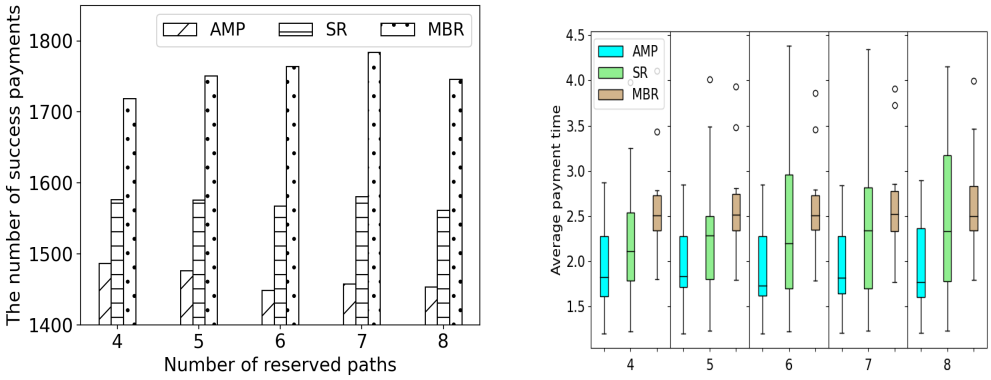
Figure 4.9: Comparison of different scheduling methods under different link delay conditions.

4.6.2 Performance under different network parameter

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

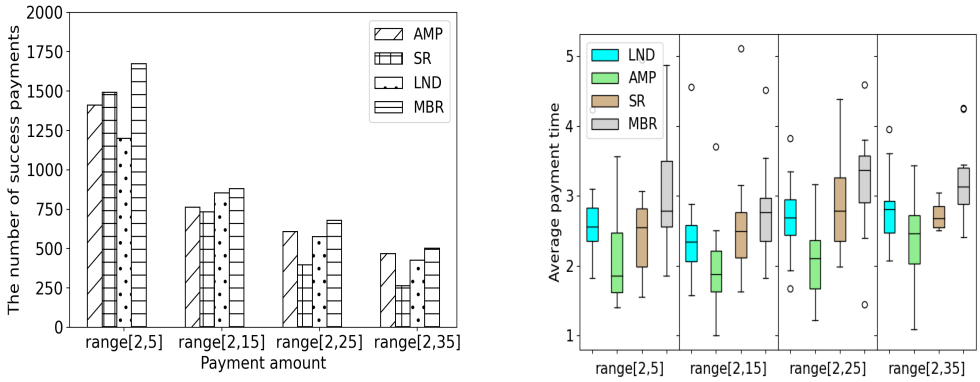
We use new payments initiated per second to denote the payment density of a PCN with 20 *pays/s* as default. As payment density increases, interactions between concurrent payments on shared payment channels are more frequent, especially for multi-path routing schemes with many redundant payments. We show the change in the number of successful payments and average payment time under different payment densities in Fig. 4.8. Our method can achieve a higher success ratio compared with other methods. The number of available payment paths is critical for routing schemes that employ multiple paths. There are 12K payments sent by 10 pairs of payer-payee within 400 seconds in our simulator. Fig. 4.10 shows the performance of relevant routing schemes under different numbers of available paths. We first measure the metric on the payment success ratio as shown in Fig. 4.10b. We then measure another important metric on averaged payment time (Fig.4.10a.). 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. Fig. 4.11 shows the comparison of different scheduling methods under different payment amounts. The payment amount is randomly sampled from each range



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

Figure 4.10: Comparison of different scheduling methods under different number of reserved paths.

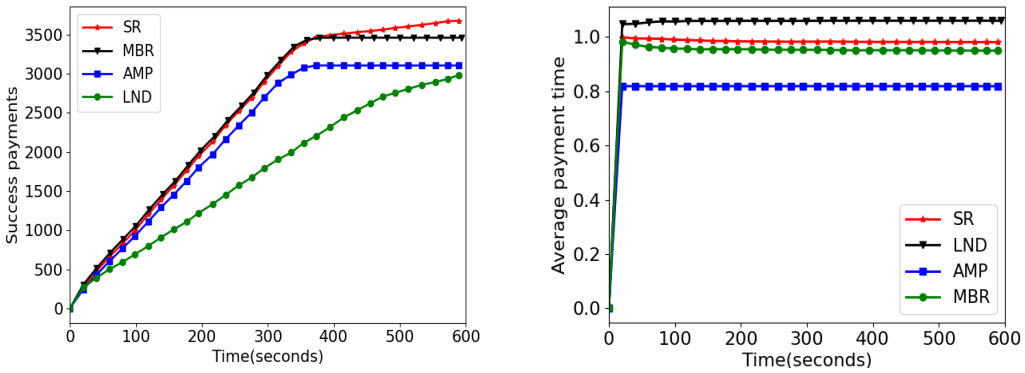


(a) Successful payments. (b) Average payment time.

Figure 4.11: Comparison of different scheduling methods under different payment amounts.

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 Fig. 4.11b. Especially for the *SR*, the payment time is gradually increasing because of the strict payment success conditions.

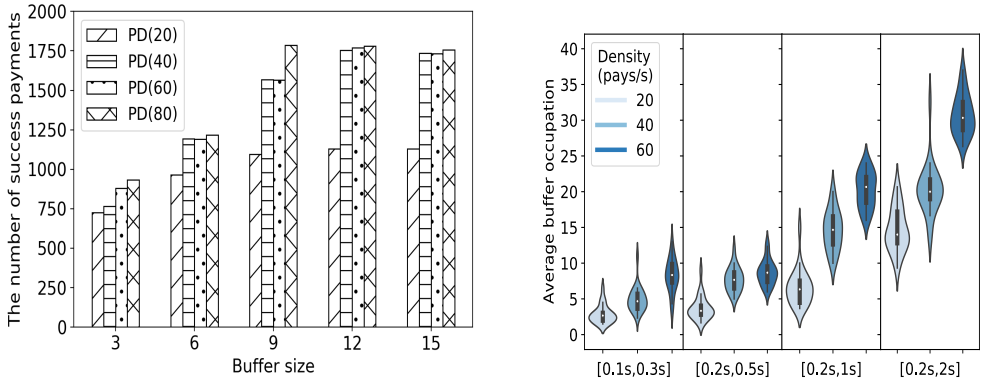
Fig. 4.12a shows the change in the number of success payments with time increases. We reset the channel balance and link delay to range $[100, 200]$ (USD) and range $[0.1s, 0.3s]$, respectively. Notice that the total number of successful payments finally stabilized under different routing schemes. The reason is that payments in the bi-direction channel are imbalanced, thereby leading to a unidirectional channel [20] with no sufficient balance for further payments. Changes in funds at both ends of a payment channel will affect its payment forwarding ca-



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

(b) The transaction throughput comparison in steady-state.

Figure 4.12: The 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 4.13: The impact of buffer size and buffer occupation.

pability, 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 [20]. 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 [70] 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 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 Fig. 4.12b. The *AMP* scheme achieves lower latency than other methods. The average payment latency of *MBR* is a little higher than others.

4.6.3 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. Fig. 4.13b 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. Fig. 4.13a 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.

4.7 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 reach a higher 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.

Chapter 5

Conclusions

In this chapter, we conclude the dissertation by summarizing the conclusions of chapters 3, 4. To improve the network throughput of PCNs, we first focus on the congestion control problem within PCNs. Particularly, we propose a priority-aware PCN for efficient transaction scheduling to achieve a high transaction throughput. By applying the priority scheduling, the transaction with different demands can be classified that we can fine-grained control over transaction flows. Additionally, the transaction in PCNs is source-routing, which is required to prepare a routing path to route transaction and specify the priority of each hop among the path. We further propose a multi-agent DQN-based priority assignment algorithm and develop a real-time simulator to verify the efficiency of our learning algorithm. The simulation result shows that our priority-aware scheduling method can achieve a higher transaction success ratio and network throughput when network congestion occurs in PCNs.

However, we cannot ensure that all of the issued transactions will be successful. Our further work takes attention to the failure transaction processing and transaction routing issues. In highly dynamic PCN, concurrent transactions compete channel capacity on shared payment channels leads to transaction failure. All of the established contracts shall be canceled. To complete the payment, the sender needs to research a now available path for re-sending. It brings overhead on path finding and contract reestablishment over overlap channels. We propose a novel multi-branch routing scheme to build an efficient route for off-chain payments. It's similar to the cold-backup protection strategy in the OpenFlow network. In order to achieve a higher performance, we further propose a Markov Chain based routing algorithm for transaction path selection and ordering. And finally, we develop a simulator of LN to simulate the payment routing process in network layer to verify the high performance of the proposed algorithm. The extensive numerical results show that our proposed algorithm has increased the payment success rate by up to %15 compared with other routing schemes. Meanwhile, it requires the

lower collateral than general multi-path routing schemes.

In this work, we address the throughput problem of off-chain networks from two different directions, and propose two different solutions. The experimental results proves that the proposed methods are efficiency.

References

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

[2] C. Lepore, M. Ceria, A. Visconti, U. P. Rao, K. A. Shah, and L. Zanolini, "A survey on blockchain consensus with a performance comparison of pow, pos and pure pos," *Mathematics*, vol. 8, no. 10, p. 1782, 2020.

[3] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[4] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Society open science*, vol. 5, no. 8, p. 180089, 2018.

[5] Sharding in ethereum. [Online]. Available: https://ethereum.org/en/upgrades/sharding

[6] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," *Available [online].[Accessed: 4-12-2018]*, 2018.

[7] Optimistic rollups in ethereum. [Online]. Available: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups

[8] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[9] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath {TCP}," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[11] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *Ieee Access*, vol. 8, pp. 16 440–16 455, 2020.

[12] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 455–471.

[13] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 931–948.

[14] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 473–489.

[15] "Bolt 7: P2p node and channel discovery." https://github.com/lightning/bolts/blob/master/07-routing-gossip.md.

[16] "Channel characteristics," https://api.lightning.community/#lnrpc-channel.

[17] lightning-onion. [Online]. Available: https://github.com/lightningnetwork/lightning-onion/blob/master/README.md

[18] "Pathfinding in lnd," https://github.com/lightningnetwork/lnd/blob/master/routing/pathfind.go.

[19] "Raiden network," https://raiden.network.

[20] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.

[21] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[22] 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 *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–9.

[23] Y. Zhang and D. Yang, "Robustpay : Robust payment routing with approximation guarantee in blockchain-based payment channel networks," *IEEE/ACM Transactions on Networking*, pp. 1–11, 2021.

[24] 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 *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 777–796.

[25] P. Marbach, "Analysis of a static pricing scheme for priority services," *IEEE/ACM Transactions on networking*, vol. 12, no. 2, pp. 312–325, 2004.

[26] M. H. Yaghmaee and D. A. Adjeroh, "Priority-based rate control for service differentiation and congestion control in wireless multimedia sensor networks," *Computer Networks*, vol. 53, no. 11, pp. 1798–1811, 2009.

[27] P. Balbastre, I. Ripoll, and A. Crespo, "Optimal deadline assignment for periodic real-time tasks in dynamic priority systems," in *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*. IEEE, 2006, pp. 10–pp.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 370–381.

[32] "The ripple data api v2," https://data.ripple.com.

[33] "Watts strogatz graph," https://networkx.github.io/documentation/networkx1.9/reference/generated/networkx.generators.randomgraphs.wattsstrogatzgraph.html.

[34] "Lightning network daemon grpc api reference," https://api.lightning.community/#lnd-grpc-api-reference.

[35] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in multi-agent systems and applications-1*. Springer, 2010, pp. 183–221.

[36] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, 2005.

[37] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," *arXiv preprint arXiv:2002.06564*, 2020.

[38] G. Ramseyer, A. Goel, and D. Mazières, "Liquidity in credit networks with constrained agents," in *Proceedings of The Web Conference 2020*, 2020, pp. 2099–2108.

[39] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," *arXiv preprint arXiv:1709.05748*, 2017.

[40] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[41] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2239–2250, 2019.

[42] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, "A multi-agent reinforcement learning perspective on distributed traffic engineering," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–11.

[43] L. Liang, H. Ye, and G. Y. Li, "Spectrum sharing in vehicular networks based on multi-agent reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2282–2292, 2019.

[44] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2499–2508.

[45] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[46] F. A. Oliehoek, C. Amato *et al.*, *A concise introduction to decentralized POMDPs*. Springer, 2016, vol. 1.

[47] A. Greenwald, K. Hall, and R. Serrano, "Correlated q-learning," in *ICML*, vol. 20, no. 1, 2003, p. 242.

[48] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[49] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1702.08887*, 2017.

[50] "Basis of lightning technology documents," https://github.com/lightningnetwork/lightning-rfc.

[51] "Lightning network daemon," https://github.com/lightningnetwork/lnd.

[52] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, 2016.

[53] "Lightning network statistics." https://bitcoinvisuals.com/lightning.

[54] H. Xue, Q. Huang, and Y. Bao, "Epa-route: Routing payment channel network with high success rate and low payment fees," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 227–237.

[55] V. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment-channel networks," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 304–324.

[56] "Amp:atomic multi-path payments over lightning," https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

[57] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 801–815.

[58] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," *Cryptology ePrint Archive*, 2016.

[59] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy preserving payments in credit networks," in *Network and distributed security symposium*, 2015.

[60] "The bitcoin lightning network: Scalable off-chain instant payments," https://lightning.network/lightning-network-paper.pdf.

[61] "The current path-finding implementation in ln," https://github.com/lightningnetwork/lnd/blob/master/routing/pathfind.go.

[62] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band openflow networks," in *2013 9th international conference on the Design of reliable communication networks (DRCN)*. IEEE, 2013, pp. 52–59.

[63] ——, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[64] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on selected areas in communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[65] X. Yuan, "Heuristic algorithms for multiconstrained quality-of-service routing," *IEEE/ACM transactions on networking*, vol. 10, no. 2, pp. 244–256, 2002.

[66] M. Chen, S. C. Liew, Z. Shao, and C. Kai, "Markov approximation for combinatorial network optimization," *IEEE transactions on information theory*, vol. 59, no. 10, pp. 6301–6327, 2013.

[67] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[68] 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 Journal on Selected Areas in Communications*, vol. 34, no. 11, pp. 2918–2934, 2016.

[69] Y. Kano and T. Nakajima, "A novel approach to solve a mining work centralization problem in blockchain technologies," *International Journal of Pervasive Computing and Communications*, 2018.

[70] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 67–75.