# Proactive Look-Ahead Control of Transaction Flows for High-Throughput Payment Channel Network


Wuhui Chen
Sun Yat-sen University
Guangzhou, China
chenwuh@mail.sysu.edu.cn

Xiaoyu Qiu
Sun Yat-sen University
Guangzhou, China
qiuxy23@mail2.sysu.edu.cn

Zicong Hong
The Hong Kong Polytechnic University
Hong Kong, China
zicong.hong@connect.polyu.hk

Zibin Zheng
Sun Yat-sen University
Guangzhou, China
zhzibin@mail.sysu.edu.cn

Hong-Ning Dai*
Hong Kong Baptist University
Hong Kong, China
hndai@ieee.org

Jianting Zhang
Purdue university
West Lafayette, USA
zhan4674@purdue.edu



## ABSTRACT

Blockchain technology has gained popularity owing to the success of cryptocurrencies such as Bitcoin and Ethereum. Nonetheless, the scalability challenge largely limits its applications in many real-world scenarios. Off-chain payment channel networks (PCNs) have recently emerged as a promising solution by conducting payments through off-chain channels. However, the throughput of current PCNs does not yet meet the growing demands of large-scale systems because: 1) most PCN systems only focus on maximizing the instantaneous throughput while failing to consider network dynamics in a long-term perspective; 2) transactions are reactively routed in PCNs, in which intermediate nodes only passively forward every incoming transaction. These limitations of existing PCNs inevitably lead to channel imbalance and the failure of routing subsequent transactions. To address these challenges, we propose a novel proactive look-ahead algorithm (PLAC) that controls transaction flows from a long-term perspective and proactively prevents channel imbalance. In particular, we first conduct a measurement study on two real-world PCNs to explore their characteristics in terms of transaction distribution and topology. On that basis, we propose PLAC based on deep reinforcement learning (DRL), which directly learns the system dynamics from historical interactions of PCNs and aims at maximizing the long-term throughput. Furthermore, we develop a novel graph convolutional network-based model for PLAC, which extracts the inter-dependency between PCN nodes to consequently boost the performance. Extensive evaluations on real-world datasets show that PLAC improves state-of-the-art PCN routing schemes w.r.t the long-term throughput from 6.6% to 34.9%.



## CCS CONCEPTS

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **Networks** → **Network economics**.

## KEYWORDS

Deep reinforcement learning, graph neural network, blockchain, payment channel network, transaction flow scheduling

**ACM Reference Format:**
Wuhui Chen, Xiaoyu Qiu, Zicong Hong, Zibin Zheng, Hong-Ning Dai, and Jianting Zhang. 2022. Proactive Look-Ahead Control of Transaction Flows for High-Throughput Payment Channel Network. In *Symposium on Cloud Computing (SoCC '22), November 7–11, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3542929.3563491


## 1 INTRODUCTION

As the underlying technology, blockchain has spawned a series of promising applications such as decentralized cryptocurrencies like Bitcoin [14]. Due to its transparent, tamperproof, and publicly verifiable nature, blockchain enables decentralized cryptocurrencies with no trusted third party. In recent years, we have witnessed the prosperity of thousands of cryptocurrencies, where the total market capitalization of active cryptocurrencies has surpassed $800 billion [35]. This prosperity also pushes the network throughput of blockchain


*Corresponding author.





*SoCC '22, November 7–11, 2022, San Francisco, CA, USA*





ACM ISBN 978-1-4503-9414-7/22/11.
https://doi.org/10.1145/3542929.3563491

to reach its limit. Bitcoin can only support at most seven transactions per second (tps) [30], while the leading centralized payment solution, like Visa has peaked at 47,000 tps during 2013 holidays [32]. The low throughput of blockchain is mainly caused by huge resource consumption made by protocols for global consensus and consistency [13, 16].

To address the scalability issue of blockchain, a promising proposal called the payment channel network (PCN) resolves the dilemma by conducting transactions in an off-chain manner. The typical PCNs include the Lightning network for Bitcoin [1], the Raiden network for Ethereum [2], and the Ripple network [3]. The main idea of PCNs is to establish channels that allow participants to make multiple payments without committing all transactions to the blockchain. To establish a channel between two users, they must escrow certain funds (i.e., currency tokens, such as the Satoshi in the Lightning network) as initial balances for exchange. To fulfill the payment demands, the sender initiates one or more off-chain transactions and sends them to the receiver through paths consisting of payment channels. The amount of available tokens escrowed in the payment channel is called the *channel balance*. Only a channel with sufficient balances can successfully forward transactions [37]. Therefore, the key problem for PCNs is how to schedule transactions for successful delivery, i.e., achieving a high throughput.

**Challenges.** However, conventional routing/scheduling protocols used in data communication networks (DCNs) cannot be directly used for PCNs. This is because the unique properties of PCNs make them intrinsically different from conventional DCNs. On the one hand, the main concern in DCNs is to find the shortest path to send data packets [19] while the main goal of PCNs is to find the path with sufficient balances [29]. On the other hand, the channel bandwidth of conventional DCNs is immediately released after usage while the channel balances of PCNs are dynamically distributed and are not automatically replenished after usage [28].

In recent years, there have been some proposals on optimizing the transaction throughput of PCNs. Most of these solutions mainly schedule transactions based on end-to-end routing algorithms, such as Flash [32], SpeedyMurmurs [25], and Spider [28]. Despite these advances, the present PCN solutions have remained unsatisfactory due to one-shot optimization. Specifically, they focus on maximizing the instantaneous throughput without considering the consumptive nature of transactions has a long-term effect on the channel balances, which potentially accelerates the channel imbalance or even depletion.

**Motivation and Contributions.** Therefore, it is necessary to design a transaction scheduling algorithm that optimizes long-term throughput. *Long-term throughput* is defined as the overall amount of successful transactions within a given long time. We used this metric instead of traditional metrics (e.g., instantaneous throughput) because PCN transaction forwarding is a consuming behavior and has long-term effects on channel balances. To this end, we propose a novel deep reinforcement learning (DRL)-based algorithm that achieves Proactive Look-Ahead Control (PLAC) of transaction flows. The basic idea of our PLAC routing strategy is to employ DRL to learn the dynamic distribution of channel balances from historical experiences and then to schedule transactions with an accurate anticipation of future payment demands. In particular, PLAC is deployed at nodes that frequently forward transactions while other common nodes do not bear any additional computational overheads. Nonetheless, the implementation of PLAC in PCNs faces the following challenges. 1) Scheduling transactions in PCNs generally involves the decision-making among thousands of nodes, thereby raising a scaling challenge compared with standard DRL tasks in other small-scale networks. 2) The behaviors of PCN nodes may affect each other, resulting in an inter-dependency between PCN nodes. Naively applying off-the-shelf DRL algorithms may not capture the sophisticated relationship between PCN nodes, consequently resulting in the risk of information loss.

To tackle these challenges, we first conduct a measurement study on two real-world PCNs to analyze the topology and transaction characteristics of PCNs. The measurement results reveal that the transaction amount distribution is highly concentrated and the degree distribution of PCNs shows similar characteristics to scale-free networks. Based on these two insights, we design PLAC, whose most important novelty is the integration of Graph Convolutional Network (GCN) with DRL. Specifically, PLAC is tailored for PCN scheduling and extends off-the-shelf DRL algorithms from two directions: 1) To cast off the scalability limit, our neural network design reuses a small set of common operations to extract features of PCNs. Thus, the neural network can be implemented as a reusable small neural network to operate at a relatively low dimension. 2) We employ GCN to extract valuable features with respect to the inter-dependency between PCN nodes, thereby contributing to efficient learning.

Table 1 shows the comparison between our proposed PLAC and four baselines. In contrast to existing PCN routing schemes, our PLAC fully considers the long-term throughput, scalability, and node dependency in a proactive manner. We conduct extensive experiments on two real-world PCNs: Ripple and Lightning. Being endowed with proactive look-ahead control capability, our PLAC outperforms state-of-the-art PCN routing algorithms by 6.6%–34.9% as shown in Table 1. Furthermore, the novel GCN-based network in our PLAC enhances the scalability of standard DRL approaches and thus provides effective methods to capture the inter-dependency among PCN nodes. As a result, the final throughput is increased by about 31% compared with standard DRL.

**Table 1: Comparison of PLAC with four baselines.**

| Routing Algorithm | Normalized Long-term Throughput | | Long-term | Proactive | Scalability | Node Dependency |
|---|---|---|---|---|---|---|
| | Ripple | Lightning | | | | |
| Flash [32] | 23.7% | 41% | ✘ | ✘ | ✓ | ✘ |
| Waterfilling [28] | 11% | 21.8% | ✘ | ✘ | ✓ | ✘ |
| Shortest Path First | 5.1% | 12.3% | ✘ | ✘ | ✓ | ✘ |
| Standard DRL | 14.4% | 14.8% | ✓ | ✓ | ✘ | ✘ |
| Our PLAC | **45.9**% | **47.6**% | ✓ | ✓ | ✓ | ✓ |

We summarize the main contributions as follows:

(1) We conduct a measurement study on the traces of two real-world PCNs and provide insights, which guide the adoption of DRL to PLAC (in Section 3).
(2) We propose PLAC that proactively schedules the transaction flows from a long-term perspective and learns high-throughput policies without prior knowledge of the system model (in Section 4 and Section 5).
(3) We integrate GCN with DRL, which casts off the scalability limit and provides the efficient learning ability by extracting the inter-dependency among PCN nodes (in Section 5).
(4) We conduct extensive experiments on two real-world PCNs to evaluate the performance of PLAC, where PLAC improves the long-term throughput by 6.6%–34.9% compared with the state-of-the-art routing schemes and the standard DRL approach (in Section 6).

## 2 BACKGROUND AND RELATED WORK

### 2.1 Preliminaries on PCNs

As a building block of PCNs, a payment channel is essentially maintained by a pair of blockchain users (aka nodes). To create a channel, two users first deposit some tokens as balances. This action is then committed to the blockchain as a transaction. As illustrated in Figure 1, Alice and Bob deposit 2 tokens and 4 tokens to the channel, respectively. The amount of available tokens in the payment channel is called the channel balance. Once the channel is established, Alice and Bob can settle the intermediate transactions via the channel without committing them to the blockchain. Only transactions that involve opening/closing channels are submitted to the blockchain. In addition, to support payments between users that are not directly connected, a PCN that is composed of multiple payment channels routes transactions in a multi-hop manner. PCN uses a Hashed Timelock Contract (HTLC) to guarantee that no user can spend the tokens until the receiver acknowledges the reception. Because settling transactions in the PCN only needs to reach consensus among users in the sending path rather than all users, PCNs thereby provide a promising alternative solution

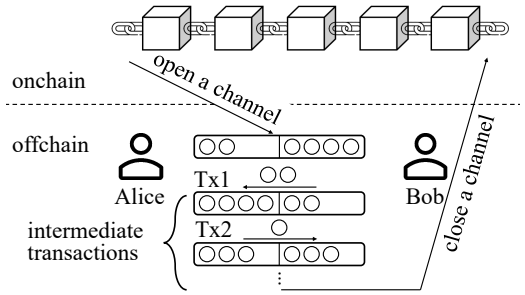


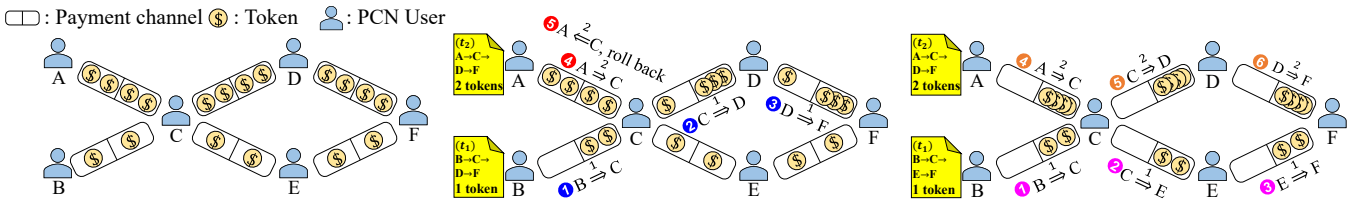

**Figure 1: A toy example of PCN. Alice and Bob can settle multiple transactions without committing to the blockchain.**

to scale blockchains. Accordingly, the key problem to PCNs is how to schedule transactions to achieve high throughput.

### 2.2 Existing PCN Routing Schemes

At a glance, PCN routing seems to be a maximum flow problem [6, 9, 12], which aims to find the widest path between two nodes. However, there are some practical constraints that complicate the problems. For example, in the current Bitcoin Lightning network, routing nodes decide the shortest path to send transactions according to a locally-maintained topology view of the network [1]. Due to the computational complexity and maintenance overhead, this approach has poor scalability. Recent proposals, such as Silentwhispers [17] and Flare [20], lower the computation complexity by using a landmark-based routing method, where nodes with high connectivity are selected as landmarks and are responsible for forwarding transactions. Another proposal is the embedding-based routing [24, 25], which assigns the coordinates to nodes and forwards transactions according to the coordinates that have been recorded in nodes. The above routing algorithms are based on either the topological structures of PCNs or the network addresses of neighboring nodes to forward transactions while failing to consider the condition that there should be sufficient balances on the forwarding channels to support the transactions.

To find channels with sufficient balances, Wang et al. proposed Flash [32], which uses a modified max-flow algorithm




(a) The initial state of PCN. (b) One-shot optimization. (c) Long-term optimization.

**Figure 2: Comparison of scheduling transactions with one-shot optimization and long-term optimization.**

to iteratively search for available channels in PCNs. In practice, this iterative process is time-consuming because it inevitably brings communication delays. Sivaraman et al. designed Spider [28], which uses a set of edge-disjoint paths as candidate paths and a congestion control-based method to adjust transaction rates. Zhang et al. proposed RobustPay$^+$ [37], which uses a set of node-disjoint paths as candidate paths and sends transactions according to their available balances. By considering the available balances of the channels, Flash, Spider, and RobustPay$^+$ [37] achieve a significant improvement over previous approaches. Despite these advances, the present PCN solutions have remained unsatisfactory due to one-shot optimization, i.e., fulfilling the current demands as many as possible. This short-sighted consideration nevertheless potentially accelerates the channel imbalance or even depletion because the consumptive nature of transactions has a long-term effect on the channel balances.

Take Figure 2 as an example. Figure 2(a) depicts the initial state of a PCN, where user $B$ wants to transfer 1 token to user $F$ at time $t_1$ while user $A$ wants to transfer 2 tokens to user $F$ at a later time $t_2$, where $t_2 > t_1$. For one-shot optimization, user $B$ prefers choosing path $B \xrightarrow{❶} C \xrightarrow{❷} D \xrightarrow{❸} F$ at time $t_1$ since the channels along this path has more tokens. However, this short-sighted strategy leads to the failure of subsequent transactions. As shown in Figure 2(b), at time $t_2$, neither channel $C \rightarrow D$ nor channel $C \rightarrow E$ has sufficient tokens to forward the pending transaction (from user $A$) at node $C$. Consequently, the pending transaction fails (i.e., ❹) and rolls back (i.e., ❺). By contrast, Figure 2(c) shows a better routing strategy of transactions from a long-term perspective, where PCN users schedule transactions with anticipation of future payment demands. In particular, user $B$ sends transactions through path $B \xrightarrow{❶} C \xrightarrow{❷} E \xrightarrow{❸} F$ while node $A$ sends transactions through path $A \xrightarrow{❹} C \xrightarrow{❺} D \xrightarrow{❻} F$. Benefit from this optimized design from the long-term perspective, PCN can achieve a higher *long-term throughput* than the one-shot optimization strategies (our experiments in Section 6 will verify this), where the long-term throughput is defined as the overall amount of successful payment demands within a given time in a long run.

## 3 MEASUREMENTS ON REAL-WORLD PCNS: A FEASIBILITY STUDY

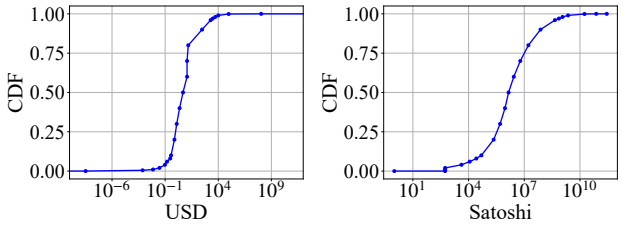
As a first step, we conduct an empirical study to investigate the characteristics of PCNs. Our study is conducted on two representative real-world PCNs: Ripple [3] and Lightning [1].

### 3.1 Analysis of Transactions Distribution

We first analyze the distribution of PCN transactions. For the Ripple network, we downloaded the dataset with over 2.6 million transactions from [25], which contains transactions from January 2013 to August 2016. For the Lightning network, the intermediate transactions conducted on the Lightning network are not publicly available. To address this issue, we adopt a similar approach to [32]. In particular, we extract all the on-chain Bitcoin transactions [23] from October 2018 to December 2020 to construct the transaction dataset for the Lightning network. This is because more on-chain transactions will be conducted in an off-chain manner for high throughput and low cost [32]. In addition, we exclude two kinds of transactions: 1) transactions without a sender or receiver; 2) transactions whose senders and receivers are the same nodes.

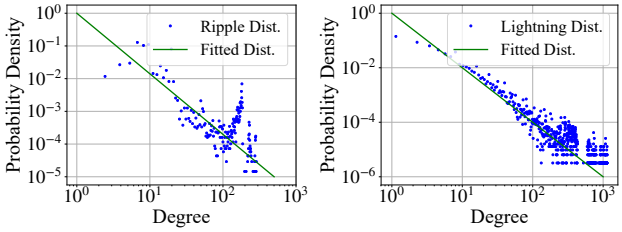
Figure 3 shows the cumulative distribution function (CDF) of transaction amounts. The transaction unit of Ripple is converted into USD based on the currency exchange on November 7, 2016. We observe that most of the transaction amounts are concentrated on a small range of values. Regarding the Ripple network, although the transaction amounts are distributed from $1 \times 10^{-11}$ USD to $1 \times 10^{19}$ USD, nearly 80% of the values are mainly concentrated on a range between $1.5 \times 10^{-1}$ USD and $8 \times 10^{2}$ USD. Similarly, with respect to Bitcoin, the transaction amounts range from $1 \times 10^{2}$ Satoshi to $4 \times 10^{12}$ Satoshi, while 80% of the values are distributed from $2 \times 10^{6}$ Satoshi to $4 \times 10^{9}$ Satoshi[1]. The highly concentrated distribution of transaction amounts offers us salient

[1]Satoshi is the smallest unit of the bitcoin currency.




(a) Ripple (b) Bitcoin

**Figure 3: Transaction amount distribution of Ripple and Bitcoin transactions.**




(a) Ripple (b) Lightning

**Figure 4: Degree distribution of Ripple and Lightning Networks.**

implications on designing our DRL for PCNs. In principle, DRL is built on calculating the *expected utility* of the current system state. Meanwhile, the action to be taken depends on the utilities of all possible next states, which are weighted by the respective probabilities. In PCNs, the generation of new transactions is typically a stochastic process. In this case, depending on the generated transactions, there exist a series of possible next states. Since most of the transaction amounts are concentrated within a small range, the expected utility is essentially traceable in DRL.

## 3.2 Analysis of Network Topology

Next, we analyze the topology characteristics. Considering that the network topology is in continuous evolution, we conduct measurements on a recent data set. We capture the topology snapshots of Ripple and Lightning on July 30, 2021. Figure 4 shows the node degree vs. the probability density (in log-log plot). The degree of a node is defined as the number of edges (i.e., channels) associated with this node. We observe that the degree distributions are highly skewed for both networks. The majority of nodes have low degrees while only a limited fraction of them have high degrees. When the degree increases, the corresponding probability density decreases exponentially; this phenomenon is similar to the scale-free networks [5]. In a scale-free network, a newly added network node can freely choose neighbors to establish edges, and prefers nodes with high degrees. In PCNs, we have comparable situations. Because routing transactions via intermediate nodes needs to pay a fee, nodes tend to establish channels to well-connected nodes with high degrees, thereby they can make transactions with smaller fees. In addition, the scale-free characteristic makes PCN more robust to network failures. In fact, many recent studies have demonstrated the scale-free nature of most PCNs [22, 23, 27].

To evaluate the similarity between a PCN and an actual scale-free network, we study the degree distribution following the steps in [5]. By definition, a network is called a scale-free network if the degrees of its nodes follow or at least approximate to the power-law distribution. In this regard, a negative linear trend is expected in the log-log plot of node degree vs. the probability density, where the slope represents the power-law exponent. We use the maximum likelihood estimation to carry out a power-law fit as in [8]. The results in Figure 4 show that the scaling exponents of the power-law distribution are 1.67 and 1.78 for Ripple and Lightning networks, respectively, suggesting that PCNs are approximated by scale-free networks [27]. In line with [27], we use this insight to divide PCN nodes into *router nodes* and *client nodes* according to their degrees. In particular, router nodes have high degrees and are connected to a large number of client nodes that have low degrees. In general, most transactions in the PCN are forwarded through router nodes, which serve as bridges to link client nodes. To make a payment, a client node first sends the transaction to its connected router node, which then forwards the transaction to the receiver in the next hop. The router nodes undertake the main responsibility of forwarding transactions in the PCN. Therefore, the key to the transaction scheduling problem lies in the *active control* of router nodes. Compared with directly controlling the whole PCN, the overhead of this method is greatly reduced.

To conclude, we have the following insights:

**Insight 1.** The distribution of transaction amounts is *highly concentrated* and most values are distributed within a small range. This leads to a more traceable long-term throughput.

**Insight 2.** The degree distribution of PCN reveals similar characteristics to *scale-free* networks, suggesting that the key to scheduling transactions is the active control of nodes with high degrees.

# 4 MODEL DESIGN

An off-chain PCN can be modeled as a graph $G(\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ denotes the set of vertices (i.e., PCN nodes) and $\mathbb{E}$ denotes the set of edges (i.e., payment channels). Based on the findings in Section 3, we divide the node set $\mathbb{V}$ into router nodes $\mathbb{M}$ and client nodes $\mathbb{C}$ according to their degrees. We take

the top-$K$ nodes with the largest degrees as router nodes and the rest as client nodes. We consider bidirectional channels, where $e = (u, v)$ denotes a channel connecting node $u$ with node $v$ ($e \in \mathbb{E}$, $u, v \in \mathbb{V}$). Each channel $e$ consists of two attributes. The first attribute is the IDs of nodes constituting the channel. The second one is the instantaneous channel balance, which denotes the remaining tokens that can be paid. For a bidirectional channel $e = (u, v)$, we denote the channel balances in both directions as $b^{(u,v)}$ and $b^{(v,u)}$.

To characterize the system dynamics, we consider a discrete time-varying model, where routing decisions are conducted in epochs. At the beginning of epoch $t$, each node estimates its payment demands to other nodes. We use a two-dimensional matrix $D_t$ with size $|\mathbb{V}| \times |\mathbb{V}|$ to denote the demands, where $|\mathbb{V}|$ is the total number of nodes in the PCN. For each pair of nodes $(u, v)$, the amount of payments from $u$ to $v$ at epoch $t$ is denoted by $d_t^{(u,v)}$. Similarly, we use a two-dimensional matrix $B_t$ to denote the balance distribution of the PCN while $b_t^{(u,v)}$ represents the balance of the channel from node $u$ to node $v$. If there is no channel between nodes $u$ and $v$, we have $b_t^{(u,v)} = b_t^{(v,u)} = 0$. Particularly, we only consider transactions that need to be forwarded through router nodes $\mathbb{M}$. This is because direct transmission requires no intermediate node. For example, the optimal routing solution is to send transactions along the channel with sufficient balances connecting two nodes directly. In addition, each transaction is associated with a timeout value. If a transaction does not reach the receiver before the timeout, the transaction fails and the frozen money will be released.

To complete each payment demand, the sender initiates one or more PCN transactions and is responsible for determining: 1) the paths to send transactions; 2) the amount of transactions sent along each path. We assume that every node maintains a local view of the graph consisting of router nodes. This is feasible in practice since the number of router nodes is typically far smaller than the total number of PCN nodes. Based on this sub-graph, each sender can calculate the paths from the source to the destination. We denote the set of candidate paths from nodes $u$ to $v$ by $\mathcal{P}^{(u,v)}$. With respect to demand $d_t^{(u,v)}$, $\lambda_t^p$ is the transaction amount sending along path $p$ at epoch $t$, where $p \in \mathcal{P}^{(u,v)}$ should be positive and not greater than the payment demand. Particularly, we have $\sum_{p \in \mathcal{P}^{(u,v)}} \lambda_t^p = d_t^{(u,v)}$. Note that forwarding transaction in PCN is not an instant process. We assume that the time taken for a transaction to be forwarded to the next hop is $\Delta$, which includes the communication delay, the time for reaching a consensus, etc. If a transaction cannot be forwarded due to insufficient balances, it must queue up at the intermediate node, where the queue length of node $u$ at epoch $t$ by $q_t^u$.

In this model, we study how to fulfill the payment demand with high throughput. In particular, we focus on long-term throughput maximization. Denoting a series of epochs by $\mathcal{T} = \{1, ..., t, ..., T\}$, the long-term throughput can be measured by the overall amount of successful payment demands in period $\mathcal{T}$. Solving the problem of maximizing long-term throughput directly requires the algorithm to be aware of the overall payment demands in the period $\mathcal{T}$. However, the future payment demands are typically not explicitly given beforehand, thereby making it impractical to be directly solved by traditional methods such as mathematical programming.

# 5 ALGORITHM DESIGN

In this section, we present how PLAC works. We first depict how PLAC formulates the problem of maximizing long-term throughput as an Markov Decision Process (MDP). Next, we refine the MDP by encoding PCN states and control actions in a customized manner. Furthermore, we present the detailed design of PLAC and how it is trained.

## 5.1 MDP-based Agent-Environment Interaction

Before diving into the algorithm design, we first present how PLAC works. DRL algorithms consider a general setting where an agent serves as a decision-maker and explores solutions in an interactive environment [7]. At each decision-making epoch, the environment sends its *state* to the agent. In response, the agent makes an *action* according to its *policy*, which will be carried out in the environment. Then, the environment returns a *reward* (i.e., a numerical value) as the estimation of action performance. In addition, the environment transits to a new *state* according to the *action* taken. This process can be modeled as an MDP, which is a sequential decision-making process and widely used in DRL [21].

Specifically, the PCN is treated as the environment in PLAC, which implements an agent as a mapping from environment states to actions to be taken. The mapping is conducted in the form of DNN, which is thereby also referred to as the actor (or policy) network. It is straightforward to view the PCN as a discrete time-varying model, where the agent interacts with the environment when new payment demands are generated. At each epoch $t$, the agent builds a state $s_t$ that characterizes the current condition of the PCN, with which the agent makes an action $a_t$ that determines how the sender sends transactions to fulfill the payment demands. Since the design of the states and actions is critical to the performance of the DRL, we present a detailed analysis in Section 5.2. In our PLAC, the length of an epoch is $\Delta$, i.e., the time taken for a transaction to be forwarded to the next hop. At the end of epoch $t$, the packaged transactions are transferred to the next hops and the agent receives a numerical reward $r_t$ that measures the performance of $a_t$. Based on the system model in

Section 4, we let the reward $r_t$ be the number of transactions that successfully reach the receivers at epoch $t$. The PCN subsequently reveals a new state $s_{t+1}$. After that, the DRL agent constructs an experience tuple $(s_t, a_t, r_t, s_{t+1})$, which is stored at the experience buffer $\mathbb{B}$ to be used for updating the agent. The objective of maximizing long-term throughput can be converted to maximizing the discounted cumulative rewards. To be consistent with existing DRL literature, the objective function is formulated as:

$$\arg\max_{\pi} \mathbb{E}_{r_t \sim \pi} \left[ \sum_{t=1}^{T} \gamma^{t-1} r_t \right], \tag{1}$$

where $\pi$ is the actor (or policy) network for decision making and $\gamma \in (0, 1]$ is the discounted factor, which measures the importance of future states. The main obstacle in solving equation (1) is how the environment transits to a new state $s_{t+1}$ from the current state $s_t$ while the state transition probability is typically unknown. This is because future payment demands are not explicitly given beforehand. This motivates us to develop PLAC that leverages DRL to learn the system dynamics of PCNs and schedule transactions with look-ahead prediction.

In addition, similar to [15, 36], we adopt a distributed deployment methodology combining DRL with blockchain consensus to safely deploy PLAC. Each qualified node has the possibility to be elected as a leader by a leadership rotation strategy for decision-making. Since we focus on long-term throughput maximization in this work, we make no further elaboration on the leadership rotation strategy.

## 5.2 Encoding PCN States and Control Actions

A scalable and effective representation of states and actions is critical to the performance of DRL. We then elaborate on how PLAC encodes the PCN states as the agent input and the control actions as the agent output.

*5.2.1 Encoding PCN states.* At the beginning of each epoch, PLAC converts the required information for decision-making (e.g., network topology, payment demands, and balance distribution) into features, which are then passed to the actor network for decision-making. For a well-performed agent, states need to reflect the most valuable information of the environment. A simple approach is to generate a flat feature vector that contains all the observed information. However, the applicability of this approach has a hard limit on the number of PCN nodes since it results in high-dimensional inputs. Further, only a few values of the flat vector are meaningful. For instance, the balance distribution matrix $B_t$ contains a great number of zero elements since most nodes have only a few neighbors. To address this issue, PLAC is implemented with an effective representation for PCN states. Specifically, PLAC leverages the scale-free nature of PCNs, i.e., router nodes that constitute the core part of PCNs are responsible for the main network overhead. Based on this implication, we develop a structured representation for PCN states from the view of router nodes. For each router node $m \in \mathbb{M}$ at epoch $t$, we have the following considerations:

- $x_t^m$ includes attributes corresponding to the node itself (e.g., node ID, the number of the connected client nodes, the number of the connected router nodes);
- $w_t^m$ combines information about the node's active neighbors (e.g., node ID, current channel balances, queue lengths, and the amount of payment demands);
- $u_t^m$ concludes the historical information of channel balances across a period of time. This time series data is beneficial to make temporal predictions.

*5.2.2 Encoding Control Actions.* Another key component of the actor network is the action representation, which encodes the high-level control decisions as action vectors. In essence, the task of scheduling transactions is to determine the transaction amounts along all paths of all senders. As a naive approach, the actor network can directly output a vector containing the sending amounts along all paths and all senders. However, this approach needs to select actions from an exponentially large space due to the large scale of PCNs. For DRL, a high-dimensional action space is usually accompanied by a high sample complexity and low training speed. Considering that current PCNs typically contain tens of thousands of nodes, it is not reasonable to directly adopt DRL to schedule transactions for all senders.

To address the challenge, we consider this problem from the perspective of router nodes. According to the insights in Section 3 and the system model in Section 4, PCNs share similar characteristics to scale-free networks. Thus, the key to transaction scheduling lies in the control of PCN nodes with a high degree. Motivated by these observations, we design PLAC with an actor network that directly outputs the maximum amount of transactions allowed to be sent through the channels between router nodes. This customized routing action of PLAC acts like congestion control signals to adjust the behaviors of senders. Because PLAC focuses on maximizing the long-term throughput, it is *proactive* in nature. In other words, instead of passively forwarding every incoming transaction, the router nodes in our model use PLAC to predict future payment demands and throttle the sending rates before channel imbalance, thus achieving proactive look-ahead control of transaction flows.

Specifically, for each bidirectional channel linking to router nodes, a two-dimensional action vector is required. In this regard, the dimension of action space is $2 \times |\mathbb{E}'|$, where $\mathbb{E}' = \{(u, v) | \forall u, v \in \mathbb{M}, \forall (u, v) \in \mathbb{E}\}$ is the set of channels linking to router nodes. Due to the scale-free nature

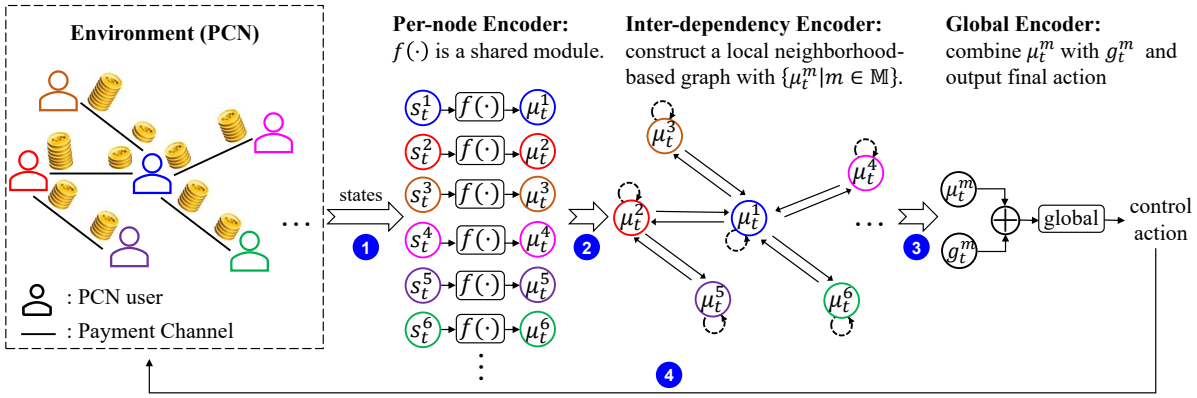



**Figure 5: Overall network structure of PLAC.**

of PCNs, the action dimensions can be greatly reduced. For a demand $d_t^{(u,v)}$, the sender determines the maximum transaction amount allowed through each candidate path $p \in \mathcal{P}^{(u,v)}$ based on the action output of the actor network. For instance, supposing that a node intends to forward transactions via router nodes $\{u, v, w\}$, then the sending limit of this path can be represented by $\min\left(A_t^{(u,v)}, A_t^{(v,w)}\right)$, where $A_t^{(u,v)}$ and $A_t^{(v,w)}$ are derived from the action outputs of the actor network. In addition, the amount of transactions sent out should not exceed the amount of payment demands.

## 5.3 Actor Network Design

The actor network tries to learn a mapping from each $s_t$ to its optimal action that maximizes the long-term reward in the equation (1). Even with the above effective representations of states and actions, there are still dozens of route nodes in PCNs. Traditional DRL network design suffers from poor performance because it requires a large-scale actor network that is nevertheless challenging to train. Even worse, the input state is graph-structured and there may be payment channels connecting two router nodes. In this respect, there is a significant inter-dependency between router nodes while it is not considered in traditional DRL algorithms.

To address this issue, PLAC achieves both scalability and effectiveness by introducing GCN, which has gained popularity in many node-level, edge-level, and graph-level tasks [34]. We design the actor network based on a GCN model in [10] and customize it for PCN routing. Intuitively, the control of a router node depends on its own state and the states of surrounding nodes. Therefore, we design three different encoders (to be described as follows) to extract different state information. Figure 5 depicts the overall network structure, which takes the state $s_t = \{x_t^m, w_t^m, u_t^m | m \in \mathbb{M}\}$ from router nodes and generates the action $a_t$ through the following encoders:

- *Per-node encoder* captures valuable features independently from the respective state information of each router node;
- *Inter-dependency encoder* captures features regarding the inter-dependency between surrounding router nodes with the output of the per-node encoder;
- *Global encoder* combines the outputs of previous encoders and then transforms them into the final action.

These three encoders extract the features of raw state information from three different levels. The per-node encoder is used to extract features from the router nodes' own states. The inter-dependency encoder is used to extract features from the state of surrounding nodes. The global encoder is used to aggregate previous extracted features and transform them into the final action. In addition, PLAC achieves scalability by implementing reusable network components in both per-node encoder and inter-dependency encoder. These encoders store features learned from the end-to-end learning process of DRL (rather than hand-coded).

*5.3.1 Per-node Encoder.* This initial process is straightforward. All router nodes share a common encoder $f(\cdot)$. Each router node $m \in \mathbb{M}$ uses $f(\cdot)$ to extract features from its own state (❶) as shown in Figure 5. We feed the state $s_t^m = \{x_t^m, w_t^m, u_t^m\}$ to this common encoder to build graph-independent features, which can be expressed as:

$$\mu_t^m = f(x_t^m, w_t^m, u_t^m), \forall m \in \mathbb{M}, \tag{2}$$

where $f(\cdot)$ is typically a non-linear transformation and can be implemented with a small neural network. This is because it operates on the state of a single node, rather than the state of the entire PCN, thereby typically having a low dimension.

*5.3.2 Inter-dependency Encoder.* PLAC next uses the outputs of previous encoders to extract the inter-dependency between router nodes. The inter-dependency between router nodes is highly related to their topology. Motivated by this implication, PLAC handles the feature outputs $\{\mu_t^m | m \in \mathbb{M}\}$

from all router nodes with a graphical model. As shown in the second step (❷) in Figure 5, PLAC builds a directed weighted graph from $\{\mu_t^m | m \in \mathbb{M}\}$ and uses a local neighborhood-based convolution process to enrich the inter-dependency features.

Specifically, the directed weighted relational graph constructed by the inter-dependency encoder is denoted by a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, in which vertices $\mathcal{V}$, edges $\mathcal{E}$ and edge weights $\mathcal{W}$ are described as follows:

- Vertices $\mathcal{V}$: The feature vector $\mu_t^m$ of each router node $m \in \mathbb{M}$ acts as a vertex $v_t^m \in \mathcal{V}$ in $\mathcal{G}$, which also represents the vertex feature.
- Edges $\mathcal{E}$: An edge represents the relationship between two vertices. PLAC uses a local neighborhood-based approach to construct edges. Let $\mathbb{N}^m$ denote router node $m$'s neighboring nodes (represented by their indices). Then, for each router node $m \in \mathbb{M}$, its corresponding vertex $v_t^m$ has bi-directional edges with vertices $\{v_t^n | n \in \mathbb{N}^m\}$.
- Edge weights $\mathcal{W}$: Each edge in $\mathcal{E}$ is associated with an edge weight that measures the impact of one node to another node; this configuration is similar to the attention mechanism [31]. Let $W_e$ denote the neural network parameters of this module. For each vertex $v_t^m$, its edge weights can be calculated as:

$$\alpha_t^m = \text{softmax}\left((\mu_t^m)^T W_e [\mu_t^{n_1}, \mu_t^{n_2}, \dots]\right), \tag{3}$$

  where $\alpha_t^m = [\alpha_t^{mn_1}, \alpha_t^{mn_2}, \dots]$ and the superscript $T$ represents the transposition and $\mathbb{N}^m = \{n_1, n_2, \dots\}$. The softmax operation ensures that the sum of weights of all incoming edges associated to node $m$ is 1. In particular, as $\mathcal{G}$ is a directed weighted graph, two connected nodes can have different relations in two opposite directions.

Note that $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is used in the GCN model of inter-dependency encoder and is distinct to the PCN model $G(\mathbb{V}, \mathbb{E})$ in Section 4.

With the above definitions, we now describe how PLAC uses GCN to capture the inter-dependency between router nodes. This encoding process can be considered as a special case of the basic differentiable message passing algorithm in [11]. Specifically, PLAC performs two steps of information propagation in the above relational graph $\mathcal{G}$. In the first step, PLAC aggregates the messages from neighboring nodes with a relation-specific transformation similar to [26], i.e.,

$$h_t^m = \text{ReLU}\left(\sum_{n \in \mathbb{N}^m} \frac{\alpha_t^{mn}}{|\mathbb{N}^m|} W_r \mu_t^n + \alpha^{mm} W_o \mu_t^m\right), \forall m \in \mathbb{M}, \tag{4}$$

where $\text{ReLU}(\cdot)$ is the rectified linear activation function (ReLU), $|\mathbb{N}^m|$ is the number of neighboring nodes of node $m$, $\alpha_t^{mn}$ and $\alpha_t^{mm}$ are edge weights calculated by equation (3). PLAC also constructs an edge that points to the node itself to capture self-dependency. The neural network parameters $W_r$ and $W_o$ can be learned by gradient-based optimization. In the second step, the new feature vector $h_t^m$ is further fed into a local neighborhood-based convolution module, i.e.,

$$g_t^m = \text{ReLU}\left(\sum_{n \in \mathbb{N}^m} W_r' h_t^n + W_o' h_t^m\right), \forall m \in \mathbb{M}, \tag{5}$$

where $W_r'$ and $W_o'$ are the learnable parameters of the neural networks in the second step of Figure 5. The local neighborhood-based transformations in equations (4) and (5) can effectively aggregate the information from neighboring nodes and model the inter-dependency between multiple participants.

*5.3.3 Global Encoder.* Given the outputs of previous encoders, the global encoder computes a summary of the embedding feature vectors and transforms them to the final action, as shown in the third step (❸) in Figure 5. To achieve this goal, PLAC first concatenates the per-node encoding features $\mu_t^m$ and the inter-dependency encoding features $g_t^m$, denoted by $k_t^m = [\mu_t^m, g_t^m]$. Then, PLAC applies a similarity-based attention mechanism to obtain the final feature vector:

$$f_t^m = \beta_t^m \left[k_t^1, k_t^2, \dots, k_t^{|\mathbb{M}|}\right]^T, \forall m \in \mathbb{M}, \tag{6}$$

where $\beta_t^m$ is the attention weight, which can be obtained by the following equation,

$$\beta_t^m = \text{softmax}\left((k_t^m)^T W_\beta \left[k_t^1, k_t^2, \dots, k_t^{|\mathbb{M}|}\right]\right). \tag{7}$$

where $W_\beta$ is the learnable parameters of the attention network.

Finally, PLAC outputs the final actions with two fully connected layers, which complete the actor network, i.e.,

$$l_t^m = \text{ReLU}\left(W_l f_t^m + b_l\right), \tag{8}$$

$$a_t^m = \tanh\left(W_t l_t^m + b_t\right), \tag{9}$$

where $W_l, W_t$ denote the weight matrices and $b_l, b_t$ denote the bias vectors of two fully connected layers. Remarkably, the scalable design of the actor network decomposes the encoding tasks for the entire PCN states into sub-problems to be solved by each router node, as shown in the fourth step (❹) in Figure 5. PLAC reuses the same operations to calculate the above embeddings for each node, which can be implemented as a reusable small neural network and operated on a relatively low dimension, consequently contributing to efficient learning, fast training, and high scalability.

## 5.4 Policy Gradient for Agent Training

With the above network design, the primary problem becomes the problem of updating the network toward maximizing the long-term throughput. Let $\pi_\theta$ denote the actor

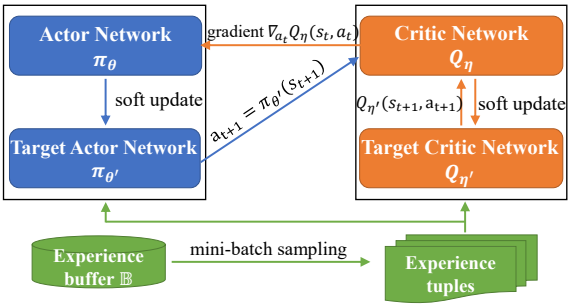



**Figure 6: The training model of the PLAC.**

network parameterized by $\boldsymbol{\theta}$, which contains all the parameters in the per-node encoder, inter-dependency encoder, and global encoder. In PLAC, $\pi_\theta$ updates its parameters based on the feedback reward $r_t$ from PCN. PLAC uses the policy gradient for agent training. The main idea of this approach is to conduct gradient descent on the network parameters by using the expected discounted cumulative rewards, i.e., $\mathbb{E}_{r_t \sim \pi_\theta}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t\right]$. However, since the payment demands of PCN arrive in a continuous and stochastic manner, it prevents us from calculating the exact value of the long-term reward. In addition, the initial policy of PLAC is poor since the network parameters are typically randomly initialized. During early training, a large queue of transactions builds up at router nodes, consequently leading to inefficient interactions and huge overhead.

As a countermeasure, PLAC uses an approximation of the long-term reward and initializes training with PCN payment traces. PLAC first uses DNNs to implement a critic network (also known as Q-network, where "Q" means "quality") as in the ACTOR-CRITIC algorithms [4]. The critic network works as an approximator that maps each state-action pair (say $(s_t, a_t)$) to the corresponding Q-value, i.e., the expected discounted cumulative reward. In practice, the critic network is implemented with a similar network model to the actor network with the exception that it takes a state-action pair as input and outputs its expected long-term rewards. Let $Q_\eta$ denote the critic network parameterized by $\boldsymbol{\eta}$. The update of the critic network is straightforward. Particularly, we use the Mean-Squared Error (MSE) to minimize the difference between predicted values and target values. As mentioned above, the exact values of the long-term reward, i.e., the target values, are intractable. Therefore, PLAC applies the Bellman operator to approximate the target values. The loss of the critic network is defined as the difference between two sides of the Bellman equation, which is expressed as follows:

$$L(\boldsymbol{\eta}) = \mathbb{E}\left[\left(Q_\eta(s_t, a_t) - \mathcal{T}Q_\eta(s_t, a_t)\right)^2\right], \tag{10}$$

where $\mathcal{T}Q_\eta(s_t, a_t)$ is expressed as follows,

$$\mathcal{T}Q_\eta(s_t, a_t) = r_t + \gamma \cdot \mathbb{E}[Q_\eta(s_{t+1}, \pi_\theta(s_{t+1}))], \tag{11}$$

**Algorithm 1** Training methodology of PLAC

**Input:** Experience buffer $\mathbb{B}$; Parameters of the actor networks $\boldsymbol{\theta}$, $\boldsymbol{\theta'}$ and parameters of critic networks $\boldsymbol{\eta}$ and $\boldsymbol{\eta'}$; Soft-update constant $\tau$; Future reward discounted factor $\gamma$; Maximum epoch $T$; Training interval $T_{\text{train}}$.
**Output:** Parameters of the actor networks and critic networks, $\boldsymbol{\theta}$, $\boldsymbol{\theta'}$, $\boldsymbol{\eta}$ and $\boldsymbol{\eta'}$.

**Initialize** the actor network and critic network with random network parameters: $\boldsymbol{\theta}$, $\boldsymbol{\theta'}$, $\boldsymbol{\eta}$ and $\boldsymbol{\eta'}$;
**for** $t = 1, 2, \ldots, T$ **do**
  The environment sends its state $s_t$ to the agent;
  The agent transforms the state input $s_t$ to the final action $a_t$ with per-node encoder, inter-dependency encoder and global encoder;
  The environment executes the action $a_t$, calculates the reward $r_t$ and transmits to a new state $s_{t+1}$.
  Store $(s_t, a_t, r_t, s_{t+1})$ into the experience buffer $\mathbb{B}$;
  **if** $t$ mod $T_{\text{train}} = 0$ **then**
    Sample a mini-batch of experience from $\mathbb{B}$;
    Compute the target Q-values of selected samples using equation (13);
    Update the critic network parameters $\boldsymbol{\eta}$ by minimizing MSE of $L(\boldsymbol{\eta})$ with the target Q-values and equation (10);
    Update the actor network parameters $\boldsymbol{\theta}$ with the sampled policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in equation (12);
    Soft update the target networks' parameters with equation (14);
  **end if**
**end for**
**return** $\boldsymbol{\theta}$, $\boldsymbol{\theta'}$, $\boldsymbol{\eta}$ and $\boldsymbol{\eta'}$.

where $s_{t+1}$ denotes the state after $s_t$ and $\pi_\theta(s_{t+1})$ denotes the output of the actor network given input $s_{t+1}$. Equation (11) is the target Q-values approximated by the Bellman operator. PLAC updates the parameters of the critic network by using the gradient-based method to minimize the loss value in equation (10). According to the Fixed Point Theorem [4], $Q_\eta(s_t, a_t)$ will eventually converge to the true long-term reward.

For the actor network, PLAC updates the parameters according to the back-propagation of the critic network. The actor network aims at maximizing the predicted long-term rewards of the critic network. By applying the chain rule of derivatives, the approximated policy gradient can be expressed as:

$$\begin{aligned}\nabla_\theta J(\boldsymbol{\theta}) &\propto \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} Q_\eta(s, a)|_{s=s_t, a=\pi_\theta(s_t)}\right] \\ &= \mathbb{E}\left[\nabla_a Q_\eta(s, a)|_{s=s_t, a=\pi_\theta(s_t)} \cdot \nabla_{\boldsymbol{\theta}} \pi_\theta(s_t)|s = s_t\right],\end{aligned} \tag{12}$$

where $\nabla_\theta$ and $\nabla_a$ denote the gradient vectors with respect to the actor network's parameters $\boldsymbol{\theta}$ and action $a$, respectively.

To maximize the Q-value, the actor network updates its parameters towards the policy gradient $\nabla_{\theta} J(\theta)$ as suggested by the critic network.

In addition, because DNN is a non-linear function approximator, the approximation in (11) is unstable, thereby leading to oscillation and the weak convergence [18, 33]. To provide a stable approximation of target Q-values, PLAC implements additional DNNs for both actor and critic networks, both of which have the same structure as the original networks. We denote the parameters of additional networks (also known as target networks) by $\boldsymbol{\theta'}$ and $\boldsymbol{\eta'}$, respectively. In this way, the target Q-value for $(s_t, a_t)$ in (11) can be rewritten as:

$$\mathcal{T}Q_{\boldsymbol{\eta}}(s_t, a_t) = r_t + \gamma \cdot \mathbb{E}[Q_{\boldsymbol{\eta'}}(s_{t+1}, \pi_{\boldsymbol{\theta'}}(s_{t+1}))]. \quad (13)$$

To constrain fluctuations, $\boldsymbol{\theta'}$ and $\boldsymbol{\eta'}$ are updated in a slower pace using soft updates, i.e.,

$$\begin{aligned} \boldsymbol{\theta'} &\leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta'}, \\ \boldsymbol{\eta'} &\leftarrow \tau\boldsymbol{\eta} + (1-\tau)\boldsymbol{\eta'}, \end{aligned} \quad (14)$$

where $\tau \ll 1$. Intuitively, because $\tau$ is a small constant, the values of $Q_{\boldsymbol{\eta'}}(s_{t+1}, \pi_{\boldsymbol{\theta'}}(s_{t+1}))$ in equation (13) is much more stable than $Q_{\boldsymbol{\eta}}(s_{t+1}, \pi_{\boldsymbol{\theta}}(s_{t+1}))$. During the training process, PLAC constantly improves the Q-value estimations of the critic network by minimizing equation (10), and improves the actor network based on the approximated gradient with equations (12) and (13). The above training process is shown in Figure 6, where an experience buffer $\mathbb{B}$ is used to store the collected experience. Through periodic updating, the actor network of PLAC eventually converges. Algorithm 1 concludes the training methodology.

## 5.5 Algorithm Overhead

We next discuss the overhead of PLAC, which includes the training overhead and runtime overhead. For the training overhead, PLAC can be trained offline using historical transaction data. In addition, PLAC improves learning efficiency by combining GCN and DRL, resulting in an acceptable convergence rate (see Section 6.2.1 for experiment results). On the other hand, the runtime overhead consists of: 1) Leadership election overhead: Leadership election requires all qualified nodes to exchange information through the network. However, such election needs to be conducted only on an interval basis and thus do not affect the transaction throughput. 2) Encoding overhead: As presented above, PLAC focuses on the scheduling of the router nodes of PCN, which only involves dozens of nodes. Benefiting from this, the scheduling task can be accomplished with small-scale networks and the encoding overhead is acceptable compared to the latency in sending transactions (e.g., encryption and decryption, channel balance probing). 3) Overhead to calculate candidate paths: PLAC uses edge-disjoint paths as candidate paths, which are completely topology-based and can be calculated prior to transaction scheduling.

**Table 2: The hidden size of PLAC's network.**

| Parameters | $W_e$ | $W_r, W_o$ | $W_r', W_o'$ | $W_\beta$ | $W_l, b_l, W_t, b_t$ |
|---|---|---|---|---|---|
| Actor | 40 | 100 | 100 | 200 | 100 |
| Critic | 40 | 50 | 50 | 150 | 100 |

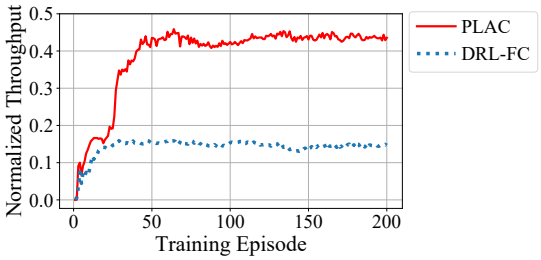
# 6 EVALUATION

## 6.1 Experiment Settings

We conduct extensive experiments based on two real-world PCNs, i.e., Ripple and Lightning. We crawl the topology of Ripple on July 4, 2021 and the topology of Lightning on December 30, 2020. According to Section 3, we deploy the PLAC agent on the top 40 nodes with the largest degrees (as router nodes). For payment demands, we use two separate exponential distributions to generate sender-receiver pairs to simulate the uneven task distribution as in [28]. In addition, we refer the distribution in Figure 3(a) to as Ripple Payment Amount Distribution (RPAD) and Figure 3(b) to as Lightning Payment Amount Distribution (LPAD), respectively. The amount of payments is randomly sampled from RPAD and LPAD for two topologies. For channel funds, we sample the historical data and assign the total funds in both directions evenly. This is because the fund distribution in PCNs is highly skewed [32]. All PCN channels are assumed to be perfectly balanced at the beginning of the evaluation. Each transaction sender uses edge-disjoint paths as candidate paths [28]. We choose the transaction RTTs between two hops as the epoch lengths of the DRL model.

For PLAC implementation, the discounted factor $\gamma$ is 0.9 and the training interval $T_{\text{train}}$ is 10. The per-node encoder is implemented based on RNN-Cells with an output size of 100. Both the actor network and critic network share similar structures, whose hyper-parameters are listed in Table 2. The network parameters are updated with the Adam optimizer, where the learning rates for the actor network and the critic network are 0.0001 and 0.001, respectively. And the soft update weight $\tau$ is 0.005. For benchmarks, we use four representative off-chain routing schemes:

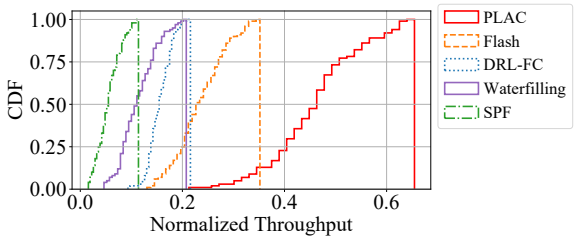
(1) Waterfilling [28] is a dynamic PCN routing scheme that follows the "waterfilling" heuristic and sends transactions through paths with the highest channel balances. The number of candidate paths is 4 as in [28].
(2) Flash [32] is a max-flow-based PCN routing algorithm, which uses a modified Edmonds-Karp algorithm and convex optimization to find paths with sufficient balances. We use the hyper-parameters as suggested in [32].
(3) Shortest Path First (SPF) Routing is a baseline that sends transactions through paths with the fewest hops.



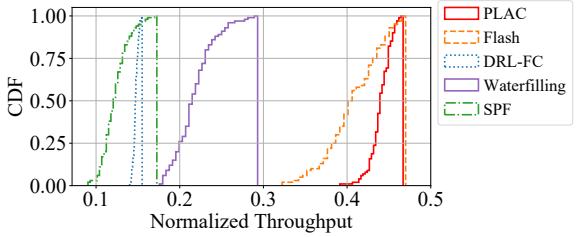

(a) Ripple


(b) Lightning

**Figure 7: Illustration of convergence curves.**




(a) Ripple




(b) Lightning

**Figure 8: CDF results of the normalized throughput for PLAC and other algorithms over 100 runs.**

If the balance of the shortest path is insufficient, the transactions will wait in the queue of the sender.

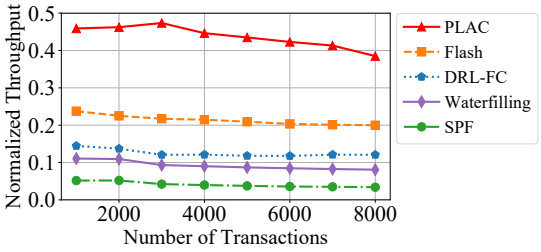
(4) DRL-FC is a DRL-based algorithm similar to our proposed PLAC except for the graph neural network being replaced by three Fully-Connected (FC) layers.

For the evaluation metric, we use the *normalized long-term throughput* over a measure interval [28]. The normalized long-term throughput is the percentage of successful payments over all generated payment demands within a given time. The measurement interval is set to be 1000 epochs.

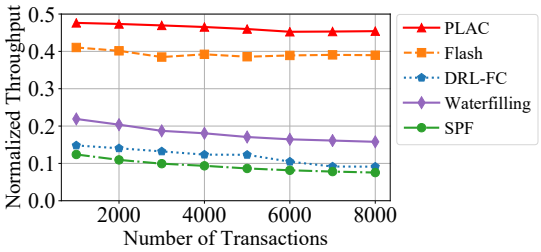
## 6.2 Performance Evaluation

*6.2.1 Convergence Behavior.* We first evaluate the effectiveness of the scalable network design of the PLAC as previously presented in Section 5.3. Figure 7 plots the convergence curves of PLAC and DRL-FC under the Ripple and Lightning networks. Significantly, PLAC converges to a much higher value than DRL-FC over both topologies. For Ripple and Lightning, PLAC achieves the normalized throughputs of about 45.9% and 47.6%, respectively. By contrast, DRL-FC only achieves the normalized throughputs of about 14.4% and 14.8% on Ripple and Lightning, respectively. This is because the fully-connected layers in DRL-FC are not designed for graph-structured data input and thus fail to extract valuable information from historical experience. Moreover, DRL-FC directly operates over the states of all router nodes, thereby requiring a large-scale neural network, which is nevertheless challenging to be trained. On the other hand, PLAC decomposes the encoding tasks for the entire PCN states into subtasks to be solved by each router node, thereby being free from the dimensionality curse. In addition, PLAC implements an effective network based on GCN, which breaks the scalability limit and provides an efficient learning ability to extract valuable features well representing the inter-dependency. As a result, PLAC achieves much higher throughput.

*6.2.2 Performance with Other Benchmarks.* Next, to evaluate the algorithm robustness, we compared PLAC with other benchmarks using the above evaluation settings. Figure 8 shows CDF results of all the methods on both Lightning and Ripple networks with 100 runs. We adopt the converged agents of PLAC and DRL-FC in the evaluations. We have the following observations. First, SPF suffers from poor performance because it selects the shortest path to send transactions and fails to consider available channel balances. Second, despite the consideration of channel balances and the attempt to optimize the long-term throughput, DRL-FC cannot make full use of the advantages of DRL due to the poor design of the neural network (i.e., fully-connected network), thereby limiting its performance. The normalized throughput of DRL-FC is lower than that of SPF in the Lightning network even though it considers the throughput. By contrast, PLAC outperforms other benchmarks on both networks, thereby demonstrating its robustness and its ability to fully exploit the benefits of DRL. The advantages are more obvious in Ripple because its network topology and payment demands are

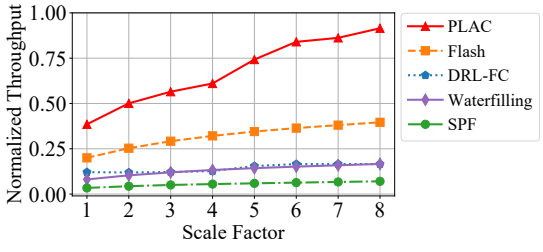



(a) Ripple

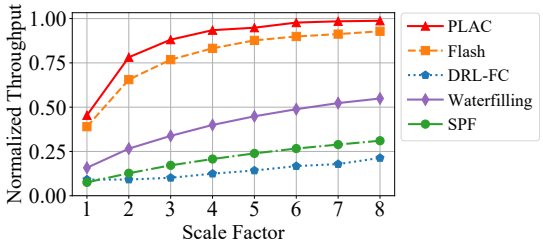



(b) Lightning

**Figure 9: Average normalized throughput vs. different numbers of transactions.**




(a) Ripple




(b) Lightning

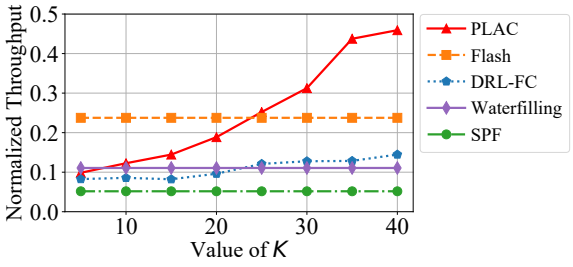
**Figure 10: Average normalized throughput vs. different capacity scale factors.**

more complicated than Lightning. The performance gains are obtained because it schedules transactions and maximizes the long-term throughput with accurate demand prediction.

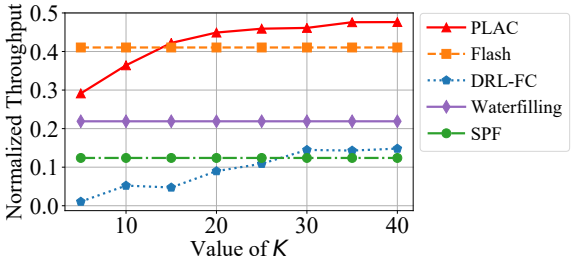
*6.2.3 Performance with Different Transaction Loads.* We next compare PLAC with other baselines over different transaction loads. We vary the number of transactions from 1000 to 8000 to simulate different workloads. As shown in Figure 9, all five routing algorithms show a downward trend in the throughput when the number of transactions increases. The phenomenon occurs mainly because more transactions lead to a heavier workload, which may quickly deplete the channel balances and saturate the channels in one direction, especially for large-size transactions. However, our PLAC consistently achieves the highest normalized throughput among all algorithms. This shows that even under a heavy workload, PLAC can still achieve proactive look-ahead control of transaction flows, so as to maximize the long-term throughput. In particular, when the number of transactions increases from 1000 to 3000, the throughput of our PLAC even shows a small upward trend in the Ripple network. This reveals that by considering the long-term reward, PLAC can schedule transactions to balance the PCN channels and prolong the channel life. By contrast, the throughputs of baselines significantly decline over the evaluations.

*6.2.4 Performance with Different Channel Capacities.* From Figure 9, we can observe that all routing algorithms have not yet achieved the 100% normalized throughput even when the number of transactions reaches 1000. This is because PCN is still in its infancy, the channel capacities sampled from real-world datasets may be limited and are unable to complete all transactions without depositing new funds [32]. In this regard, we investigate the performance of all routing algorithms with varied channel capacities. In particular, we vary the channel capacity by proportionally scaling up the capacities with a ratio of 1 to 8. Figure 10 presents the results. With the increment of channel capacities, all routing algorithms on both Lightning and Ripple networks show a rising tendency in the normalized throughput. This is because a higher channel capacity can mitigate channel depletion effects, so as to complete more transactions. In both topologies, PLAC achieves throughputs approximated to 100%, i.e., 91.5% for Ripple and 98.7% for Lightning. Especially for Ripple, the throughput of PLAC is at least twice as much as that obtained by the best baseline, i.e., Flash. This shows that our algorithm can better utilize the channel funds for higher throughput.

*6.2.5 Performance with Different Values of $K$.* As stated in Section 4, the top-$K$ nodes with the largest degrees are selected as router nodes. We then use PLAC to output the control actions of router nodes to schedule the transaction flows. We vary the value of $K$ from 5 to 40 to evaluate its impact. For those nodes that are not selected, they randomly output actions. Figure 11 shows that PLAC rapidly improves the throughput and eventually exceeds all baselines although it does not achieve high throughput at the beginning. This is because PLAC can achieve cooperative control of router




(a) Ripple




(b) Lightning

**Figure 11: Average normalized throughput vs. different values of $K$.**

nodes. A larger $K$ value allows PLAC to better control the transaction flows, therefore reaching the network balance and obtaining a high throughput. In particular, the scale-free nature of PCNs allows us to obtain excellent results by deploying PLAC to dozens of router nodes in a PCN that typically contains thousands of nodes. For example, PLAC outperforms other baselines when the values of $K$ are larger than 25 and 15 for the Ripple and Lightning, respectively.

*6.2.6 Performance with Different Types of Candidate Paths of Senders.* PLAC schedules the transaction flows from the perspective of router nodes and determines the maximum amount of transactions allowed to be sent through the channels between router nodes. Thereafter, the senders determine the amount of transactions sent through each candidate path. We then investigate different design choices of PLAC with respect to different types of candidate paths. Figure 12 plots the results, where EDJ, NDJ, EDJS represent edge-disjoint paths, node-disjoint paths, and edge-disjoint shortest paths, respectively. It can be observed that PLAC-EDJ outperforms other approaches throughout the evaluations. This is because edge-disjoint paths can make better use of the available channels in PCNs and avoid choosing the same channel multiple times. In addition, PLAC-EDJS has the worst performance among three versions of PLAC. This is because the available channel balance in PCNs also plays a key role in the successful settlements of transactions. It is not sufficient to only consider

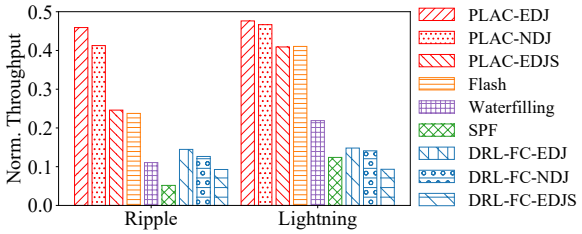



**Figure 12: Average normalized throughput vs. different types of candidate paths.**

the length of sending path. Nevertheless, PLAC-EDJS still achieves a throughput comparable to Flash, thereby further validating the effectiveness of our algorithm.

## 7 CONCLUSION

In this paper, we have studied the transaction scheduling problem for achieving a high throughput of PCNs. Since the future payment demands are typically unavailable beforehand, we propose PLAC, a DRL-based algorithm that learns system dynamics of PCNs through interactions and proactively controls the router nodes with the predictions of future payment demands. Meanwhile, PLAC designs a GCN-based network to effectively extract valuable features regarding the inter-dependency between router nodes. The GCN-based model also allows us to implement reusable network components and operates inputs on a relatively low dimension, consequently contributing to the high scalability. A policy gradient-based training methodology is further proposed to improve the performance of PLAC with real-world PCN payment traces. Compared to state-of-the-art PCN routing algorithms, PLAC increases the long-term throughput by 6.6% to 34.9%. PLAC also outperforms the standard DRL algorithm by about 31%, which shows its effectiveness in learning the dynamics of PCNs. Experimental results further demonstrate the importance of long-term optimization and proactive control of routing nodes in PCNs.

## ACKNOWLEDGEMENT

The work described in this paper was supported by the National Key Research and Development Plan (2021YFB2700302), the National Natural Science Foundation of China (62172453), the National Natural Science Foundation of Guangdong province (2022A1515010154), 6142006200403, XM2021XT1084, the Major Key Project of PCL (PCL2021A06), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (2017ZT07X355), and the Pearl River Talent Recruitment Program (No. 2019QN01X130).

We would like to thank our shepherd Heming Cui and anonymous reviewers for their comments and suggestions on improving this paper.

# REFERENCES


[1] 2022. Lightning Network Daemon. https://github.com/lightningnetwork/lnd.

[2] 2022. Raiden Network. https://raiden.network/.

[3] 2022. Ripple. https://ripple.com/.

[4] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.* 34, 6 (2017), 26–38. https://doi.org/10.1109/MSP.2017.2743240

[5] Albertlaszlo Barabasi and Eric Bonabeau. 2003. Scale-Free Networks. *Scientific American* 288, 5 (2003), 60–69.

[6] Hongliang Bi, Yanjiao Chen, and Xiaotian Zhu. 2022. A Multi-path Routing for Payment Channel Networks for Internet-of-Things Micro-Transactions. *IEEE Internet of Things Journal* (2022), 1–1. https://doi.org/10.1109/JIOT.2022.3167098

[7] Wuhui Chen, Xiaoyu Qiu, Ting Cai, Hong-Ning Dai, Zibin Zheng, and Yan Zhang. 2021. Deep Reinforcement Learning for Internet of Things: A Comprehensive Survey. *IEEE Commun. Surv. Tutorials* 23, 3 (2021), 1659–1692. https://doi.org/10.1109/COMST.2021.3073036

[8] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[9] Vajiheh Farhadi, Fidan Mehmeti, Ting He, Thomas F. La Porta, Hana Khamfroush, Shiqiang Wang, Kevin S. Chan, and Konstantinos Poularakis. 2021. Service Placement and Request Scheduling for Data-Intensive Applications in Edge Clouds. *IEEE/ACM Trans. Netw.* 29, 2 (2021), 779–792. https://doi.org/10.1109/TNET.2020.3048613

[10] Deepanway Ghosal, Navonil Majumder, Soujanya Poria, Niyati Chhaya, and Alexander F. Gelbukh. 2019. DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 154–164. https://doi.org/10.18653/v1/D19-1015

[11] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1263–1272. http://proceedings.mlr.press/v70/gilmer17a.html

[12] Qianyun Gong, Chengjin Zhou, Le Qi, Jianbin Li, Jianzhong Zhang, and Jingdong Xu. 2021. VEIN: High Scalability Routing Algorithm for Blockchain-based Payment Channel Networks. In *20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021, Shenyang, China, October 20-22, 2021*. IEEE, 43–50. https://doi.org/10.1109/TrustCom53373.2021.00024

[13] Yilin Han, Chenxing Li, Peilun Li, Ming Wu, Dong Zhou, and Fan Long. 2020. Shrec: Bandwidth-Efficient Transaction Relay in High-Throughput Blockchain Systems *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 238–252. https://doi.org/10.1145/3419111.3421283

[14] Huawei Huang, Wei Kong, Sicong Zhou, Zibin Zheng, and Song Guo. 2021. A Survey of State-of-the-Art on Blockchains: Theories, Modelings, and Tools. *ACM Comput. Surv.* 54, 2 (2021), 44:1–44:42. https://doi.org/10.1145/3441692

[15] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing Off-Blockchain Payment Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 439–453. https://doi.org/10.1145/3133956.3134033

[16] Peilun Li, Guosai Wang, Xiaoqi Chen, Fan Long, and Wei Xu. 2020. Gosig: A Scalable and High-Performance Byzantine Consensus for Consortium Blockchains *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 223–237. https://doi.org/10.1145/3419111.3421272

[17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2017. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[18] Jiandong Mu, Mengdi Wang, Feiwen Zhu, Jun Yang, Wei Lin, and Wei Zhang. 2021. Boosting the Convergence of Reinforcement Learning-based Auto-pruning Using Historical Data. *CoRR* abs/2107.08815 (2021). arXiv:2107.08815 https://arxiv.org/abs/2107.08815

[19] Konstantinos Poularakis, Jaime Llorca, Antonia Maria Tulino, Ian J. Taylor, and Leandros Tassiulas. 2020. Service Placement and Request Routing in MEC Networks With Storage, Computation, and Communication Constraints. *IEEE/ACM Trans. Netw.* 28, 3 (2020), 1047–1060. https://doi.org/10.1109/TNET.2020.2980175

[20] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. 2016. Flare: An approach to routing in lightning network. *White Paper* (2016).

[21] Xiaoyu Qiu, Luobin Liu, Wuhui Chen, Zicong Hong, and Zibin Zheng. 2019. Online Deep Reinforcement Learning for Computation Offloading in Blockchain-Empowered Mobile Edge Computing. *IEEE Trans. Veh. Technol.* 68, 8 (2019), 8050–8062. https://doi.org/10.1109/TVT.2019.2924015

[22] Gabriel Antonio F. Rebello, Gustavo Franco Camilo, Maria Potop-Butucaru, Miguel Elias M. Campista, Marcelo Dias de Amorim, and Luís Henrique M. K. Costa. 2022. PCNsim: A Flexible and Modular Simulator for Payment Channel Networks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops, INFOCOM 2022 - Workshops, New York, NY, USA, May 2-5, 2022*. IEEE, 1–2. https://doi.org/10.1109/INFOCOMWKSHPS54753.2022.9798003

[23] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. In *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 347–356. https://doi.org/10.1109/EuroSPW.2019.00045

[24] Stefanie Roos, Martin Beck, and Thorsten Strufe. 2016. Anonymous addresses for efficient and resilient routing in F2F overlays. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524553

[25] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2018. Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_09-3_Roos_paper.pdf

[26] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10843)*, Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam (Eds.).

Springer, 593–607. https://doi.org/10.1007/978-3-319-93417-4_38

[27] István András Seres, László Gulyás, Dániel A. Nagy, and Péter Burcsi. 2019. Topological Analysis of Bitcoin's Lightning Network. In *Mathematical Research for Blockchain Economy, 1st International Conference, MARBLE 2019, Santorini, Greece, May 6-9, 2019*, Panos M. Pardalos, Ilias S. Kotsireas, Yike Guo, and William J. Knottenbelt (Eds.). Springer, 1–12. https://doi.org/10.1007/978-3-030-37110-4_1

[28] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia C. Fanti, and Mohammad Alizadeh. 2020. High Throughput Cryptocurrency Routing in Payment Channel Networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Ranjita Bhagwan and George Porter (Eds.). 777–796.

[29] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. 2020. Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2 (2020), 29:1–29:39. https://doi.org/10.1145/3392147

[30] Parth Thakkar and Senthilnathan Natarajan. 2021. Scaling Blockchains Using Pipelined Execution and Sparse Peers. In *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, Carlo Curino, Georgia Koutrika, and Ravi Netravali (Eds.). ACM, 489–502. https://doi.org/10.1145/3472883.3486975

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[32] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, CoNEXT 2019, Orlando, FL, USA, December 09-12, 2019*, Aziz Mohaisen and Zhi-Li Zhang (Eds.). ACM, 370–381. https://doi.org/10.1145/3359989.3365411

[33] Qiong Wu, Xu Chen, Zhi Zhou, Liang Chen, and Junshan Zhang. 2021. Deep Reinforcement Learning With Spatio-Temporal Traffic Forecasting for Data-Driven Base Station Sleep Control. *IEEE/ACM Trans. Netw.* 29, 2 (2021), 935–948. https://doi.org/10.1109/TNET.2021.3053771

[34] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[35] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. 2018. CoinExpress: A Fast Payment Routing Mechanism in Blockchain-Based Payment Channel Networks. In *27th International Conference on Computer Communication and Networks, ICCCN 2018, Hangzhou, China, July 30 - August 2, 2018*. IEEE, 1–9. https://doi.org/10.1109/ICCCN.2018.8487351

[36] Jianting Zhang, Zicong Hong, Xiaoyu Qiu, Yufeng Zhan, Song Guo, and Wuhui Chen. 2020. SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Blockchain Sharding System. In *ICPP 2020: 49th International Conference on Parallel Processing, Edmonton, AB, Canada, August 17-20, 2020*, José Nelson Amaral, Lizy Kurian John, and Xipeng Shen (Eds.). ACM, 3:1–3:11. https://doi.org/10.1145/3404397.3404460

[37] Yuhui Zhang and Dejun Yang. 2021. RobustPay$^+$: Robust Payment Routing With Approximation Guarantee in Blockchain-Based Payment Channel Networks. *IEEE/ACM Trans. Netw.* 29, 4 (2021), 1676–1686. https://doi.org/10.1109/TNET.2021.3069725