Graph Neural Network-Enhanced Reinforcement Learning for Payment Channel Rebalancing

Wuhui Chen, *Member, IEEE*, Xiaoyu Qiu, Zhongteng Cai, Bingxin Tang, Linlin Du, and Zibin Zheng, *Fellow, IEEE*

***Abstract*—Building on top of blockchain, payment channel networks-backed (PCNs) cryptocurrencies emerge as a promising solution for a mobile payment system with fewer intermediaries, more security, higher speed, and lower cost. A key problem for PCN is payment channel rebalancing, that is, finding a set of circular transactions that restore a PCN with skewed channel balances back into an equilibrium state. However, existing practice on payment channel rebalancing either has a hard limit on the problem size or tends to fall into local optimum. To address these challenges, we propose DRL-PCR, a Deep Reinforcement Learning-based Payment Channel Rebalancing algorithm. On one hand, DRL-PCR leverages the strong approximation ability of deep neural networks to handle large problem spaces. On the other hand, DRL-PCR decomposes the rebalancing problem into a sequence of decision-making problems and progressively builds the final solution. By aiming to find a globally optimized solution and solving the long-term optimization model of DRL, DRL-PCR is superior to greedy-based algorithms and can mitigate the risk of getting trapped in a local optimum. In particular, payment channel rebalancing typically involves dealing with graph-structured data, where the major obstacle lies in understanding the sophisticated circular dependencies between payment channels and routing paths. DRL-PCR achieves this by encoding the input data with a novel graph neural network-based model and capturing the circular dependencies through a customized message passing process. In addition, considering the distributed nature of PCN, DRL-PCR uses a leadership election protocol to elect leaders for decision-making. Evaluations on the historical data of two real-world PCNs demonstrate that DRL-PCR can restore the PCN to a more balanced state and improve the transaction throughput and success ratios by up to 2.1x and 1.6x, respectively.**

***Index Terms*—Payment channel network, payment channel rebalancing, deep reinforcement learning, graph neural network.**

Manuscript received 20 October 2022; revised 18 September 2023; accepted 25 October 2023. Date of publication 30 October 2023; date of current version 7 May 2024. This work was supported in part by the National Key Research and Development Plan under Grant 2021YFB2700302, in part by the National Natural Science Foundation of China under Grant 62172453, in part by the National Natural Science Foundation of Guangdong province under Grant 2022A1515010154, and in part by Pearl River Talent Recruitment Program under Grant 2019QN01X130. Recommended for acceptance by Q. Zhao. *(Corresponding author: Zibin Zheng.)*

Wuhui Chen, Xiaoyu Qiu, Zhongteng Cai, Bingxin Tang, and Zibin Zheng are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China, and also with the National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou 510006, China (e-mail: chenwuh@mail.sysu.edu.cn; qiuxy23@mail2.sysu.edu.cn; caizht3@mail2.sysu.edu.cn; tangbx@mail2.sysu.edu.cn; zibinzheng@yeah.net).

Linlin Du is with the Science and Technology on Complex Systems Simulation Laboratory, Guangzhou 510006, China (e-mail: chx0808@sina.com).

Digital Object Identifier 10.1109/TMC.2023.3328473

I. Introduction

MOST existing mobile payment systems rely on third centralized parties, which face challenges of trust, security, data loss, and high costs. Recently, by taking advantage of decentralization, anonymity and trust, blockchain has been widely adopted in cryptocurrencies. However, blockchain suffers severely from high transaction latency and low throughput, which limits its applications in large-scale scenarios such as mobile payment systems [1], [2]. Fortunately, payment channel network (PCN) has emerged as a promising solution to address the scalability limitations of blockchain. The main idea behind PCN is to establish peer-to-peer offchain channels between two parties, which allows them to settle transactions without resorting to expensive onchain operations. In addition, PCN uses security mechanisms such as Hashed Timelock Contract (HTLC) to support multi-hop transactions. As a result, payment channel networks-backed cryptocurrencies emerge as a promising solution for mobile payment systems with fewer intermediaries, more security, higher speed, and lower cost.

Despite PCN has made some achievements, few have seen its widespread implementation in mobile payment systems. The major bottleneck of the PCN is that because the transaction demands are generally unbalanced, the balance distribution of channels would become heavily skewed as transactions proceed. In this case, the depleted channels need to be refunded using expensive onchain operations. *Rebalancing* is proposed to alleviate the problem, which attempts to restore the network back into an equilibrium state by initiating offchain circular transactions. PCN users with depleted channels can transfer their funds along circular paths to replenish their channels in an offchain manner. Some pioneering work has been proposed for payment channel rebalancing, such as Revive [3], GCBRS_PCNs [4], and mpp-rebalancing [5]. However, these approaches either have a hard limit on the problem scale (e.g., Revive) or rely on greedy heuristics (e.g., GCBRS_PCNs and mpp-rebalancing). As a result, none of these approaches can fulfill the promise of the rebalancing mechanism in improving PCN throughput. This calls for a rebalancing algorithm that can deal with *large problem spaces* and *aim at global optimization*.

Motivation: The core of this work is to solve "the last mile" problem of PCN-based mobile payment systems. To this end, we propose DRL-PCR, a Deep Reinforcement Learning-based Payment Channel Rebalancing algorithm. By combining deep learning (DL) and reinforcement learning (RL), DRL makes a good fit for payment channel rebalancing. On one hand,

DRL-PCR exploits the strong approximation ability of the deep neural networks (DNNs) in DL to handle *large problem spaces*. On the other hand, DRL-PCR leverages the principle of RL to solve *global optimization problems*. In particular, RL is a popular paradigm for solving sequential decision problems, such as robot control and game control. Superior to greedy algorithms, the objective of RL is to maximize the cumulative return of the entire decision sequence, thereby mitigating the risk of getting trapped in a local optimum. In addition, payment channel rebalancing is free from the notorious sim-to-real gap that hinders the practical application of DRL in many scenarios. This is because a rebalancing solution generated by DRL can be accurately evaluated without any assumptions. To the best of our knowledge, DRL-PCR is the first DRL-based circular rebalancing solution for PCN.

Challenges: However, some unique characteristics of payment channel rebalancing give rise to the challenges of applying DRL. There are two main challenges. The first is how to transform the rebalancing problem into a Markov Decision Process (MDP), which is commonly used to model sequential decision problems in DRL and is critical for its applicability. For instance, one straightforward way to apply DRL is using it to directly take the states of all channels as input and output the whole circular transaction set. But this will lead to a high-dimensional state space and action space, and require a large neural network that is extremely hard to train. The second challenge is how to extract effective features from graph-structured state input. Standard DRL tasks generally involve processing simple vector inputs or image inputs, where the fully connected layers and convolutional layers can be competent for this task. In contrast, payment channel rebalancing typically involves processing graph-structured input with an arbitrary number of nodes and channels. The major obstacle lies in understanding the sophisticated relationship between payment channels and routing paths, where a path is constituted of multiple channels and different paths may share the same channels.

Solutions and Contributions: To realize the prospect of DRL, we propose DRL-PCR to address the above challenges. The salient features of DRL-PCR are as follows: 1) MDP formulation: The rebalancing algorithm is triggered automatically once T transactions fail due to insufficient channel capacity. Then, the framework formulates a T-step MDP and progressively builds a circular transaction set as the rebalancing solution. This frees DRL-PCR from high-dimension problem space while providing a forward-looking component to the algorithm. 2) Using GNN to encode system state: DRL-PCR handles the sophisticated circular dependencies by encoding the graph-structured inputs with graph neural network (GNN). The novelty is in the tailored message-passing algorithm designed to encode the circular dependencies existing among the paths and channels. 3) Offline training of RL algorithm: As the training of the actor-critic network can be quite slow compared to the timescale of the transactions, conducting offline training becomes imperative. In particular, by using historical data, the framework learns the optimal policy in an offline fashion. As the formulated MDP is inherently deterministic, it provides stable learning of the RL value and policy network.

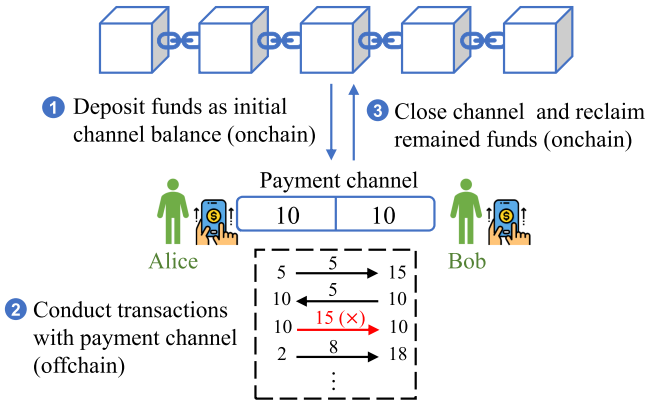


Fig. 1. Simple illustration for the PCN.

The main contributions are summarized as follows:

- To the best of our knowledge, DRL-PCR is the first DRL-based circular channel rebalancing algorithm, which has the ability to handle large problem scales and mitigate the risk of getting trapped in a local optimum.
- DRL-PCR decomposes the rebalancing problem into a sequential decision problem and progressively builds the final solution step-by-step. By maximizing the cumulative return, DRL-PCR is superior to greedy algorithms.
- A novel GNN-based model and a tailored message-passing process are introduced, which enables DRL-PCR to effectively capture the circular dependencies between paths and channels.
- We evaluate the performance of DRL-PCR over the historical data of two real-world PCNs. Compared with state-of-the-art rebalancing algorithms, DRL-PCR can restore PCNs to a more balanced state and improve transaction throughput and success ratios by up to 2.1x and 1.6x, respectively.

The rest of this paper is organized as follows. Section II first provides some necessary background and related work. Section III then presents the rebalancing model. Section IV presents the design of DRL-PCR, including the problem formulation, overall architecture, network model, and training process. Evaluation results based on two real-world PCNs are presented in Section V, followed by a conclusion in Section VII.

II. Background and Related Work

A. Preliminaries on PCN

Payment channels are the building blocks of PCN. If two users want to conduct transactions with each other frequently, they can establish an offchain payment channel by depositing funds through an onchain operation. As shown in Fig. 1, Alice and Bob jointly deposit 10 coins into the payment channel. Once established, Alice and Bob can conduct offchain transactions with the payment channel. For the transaction to be completed successfully, the remaining funds in the channel must not be less than the amount to be sent. For instance, the third transaction in Fig. 1 fails to execute because Alice does not have enough

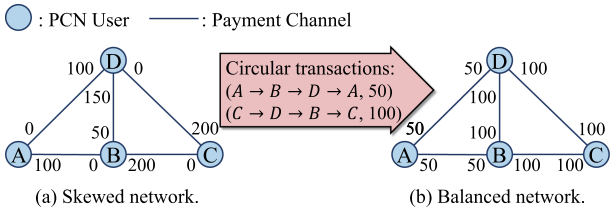


Fig. 2. Simple example of the rebalancing mechanism.

coins. Since this payment channel only involves the transactions between Alice and Bob, it is maintained privately by these two users and the update of the channel balances requires only a consensus between Alice and Bob, not all users. In addition, anyone can close the channel by submitting an onchain transaction to reclaim the remaining funds in the channel.

The advantages of the PCN are further manifested with multi-hop transactions. For users who are not directly connected, they can make transactions through multi-hop paths, where the intermediate nodes will relay the transaction to the destination and update the balances of the payment channels along the paths accordingly. To prevent intermediate nodes from stealing funds, PCN uses security mechanisms such as HTLC. With HTLC, a payment is completed only when the payee shows the preimage of a hash value before a negotiated block height on the blockchain. Otherwise, the payment is considered invalid if the payee does not show the preimage and the timeout expires. In this way, HTLC guarantees that funds forwarded will be locked until the recipient successfully receives the transaction. Since we focus on channel rebalancing in this work, we do not elaborate on the security mechanism of PCN. We direct interested readers to [6] for more details.

B. Payment Channel Rebalancing

With the continuous execution of transactions, some payment channels may be used extensively as the payment demand is not uniform. As a result, PCN may degenerate into a highly skewed network, where transaction routing becomes inefficient and it is difficult to find a path with sufficient balances. Fig. 2(a) illustrates a highly skewed PCN where all the money is transferred to one side of the channels and the bidirectional channels degenerate into uni-directional channels. For example, user B cannot send a payment directly to A through channel $B \rightarrow A$ because there is no longer enough money left in that direction. One possible solution is to replenish the channel with new funds by submitting an onchain transaction. However, onchain transactions come at an expensive cost because they require all blockchain nodes to reach a consensus.

To solve the problem, a rebalancing mechanism is proposed in [3], whose main idea is to restore the network to, or as close as possible to an equilibrium state by initiating offchain circular transactions. In this way, PCN users with depleted channels can transfer their funds along a circular path to replenish their channels in an offchain manner. Fig. 2 shows how the rebalancing mechanism works on a highly skewed network. The circular transactions are $(A \rightarrow B \rightarrow D \rightarrow A, 50)$ and $(C \rightarrow D \rightarrow B \rightarrow C, 100)$, where the numbers represent the amount of the transactions. After successful delivery, the PCN is restored to a balanced state, and the funds of each channel are evenly distributed, as shown in Fig. 2(b). In general, a fee must be paid to the intermediate nodes for the multi-hop transactions PCNs. However, as we focus on the channel rebalancing problem in this work, we assume zero relay fees for simplicity. But the approach in this work can be extended to non-zero fees.

C. Related Work

As a promising solution to scale blockchain, PCN has received wide attention in recent years. Most of the recent work has focused on the transaction routing problem, which is about finding a path with sufficient funds to forward transactions. For instance, Flare [7] was proposed for the Lightning network, where each user maintains a local view of the PCN and sends transactions with a Breadth-First-Search-based method. Wang et al. proposed Flash [8], which uses the atomic multi-path payment protocol and a modified maximum flow algorithm to search paths with sufficient balances. Sivaraman et al. proposed Spider [9], which uses a multi-path congestion control protocol to guide the routing of transactions and uses a packet-switched method to send large transactions over low-capacity payment channels. By taking advantage of deep learning, [10], [11] use deep neural networks to predict the distribution of future transactions and routes transactions accordingly. However, because the distribution of real-world transactions is typically uneven, using only routing algorithms will eventually lead to channel depletion [5]. In this case, a depleted payment channel is unavailable until being closed and refunded, which involves expensive onchain operations.

In view of this, some pioneering work has been proposed for payment channel rebalancing, which can be divided into two major categories: greedy-based rebalancing and linear programming-based rebalancing. Payment channel rebalancing aims at restoring the network to or as close as possible to an equilibrium state by initiating circular transactions. The first decomposes the rebalancing problem into a sequence of subproblems and follows a greedy heuristic to generate circular transactions at each step, which together constitute the final rebalancing solution. Some emerging rebalancing algorithms fall into this category. Huo et al. proposed GCBRS_PCNs [4], which divides the generation of a transaction into multiple transaction units. GCBRS_PCNs realizes the local equilibrium of the PCN by using the Gini coefficient of the PCN as the selection criterion for the next hop and greedily generating the sending path of the transaction. Pickhardt et al. proposed mpp-rebalancing [5], which considers the rebalancing problem from the view of a single user node. Each user tries to rebalance its own payment channels, without considering its impact on the whole network. Because greedy-based rebalancing only considers one-shot optimization, it is easy to fall into local optimum.

Falling into the category of global optimal methods, linear programming-based rebalancing is a method to formulates the rebalancing problem as a maximization/minimization problem

with several constraints and relies on existing linear programming solvers to approximate the global optimal. For instance, a state-of-the-art rebalancing algorithm called Revive [3] uses a leadership rotation strategy to select a decision center and adopts the 'linprog' package of Python to generate a set of circular transactions as the rebalancing solution. Li et al. proposed PnP [12], which provides a balance planning service to determine the minimum funds to be deposited. PnP formulates the balance planning problem as a chance-constrained optimization problem and adopts the interior-point algorithm to approximate the solution. The advantage of the global optimal methods is obvious, i.e., they guarantee the upper limit of performance. However, their disadvantages cannot be ignored, i.e., they have a hard limit on the problem size in practice. On the one hand, the global optimal methods need to collect the balances of a large number of channels, resulting in a huge communication overhead. On the other hand, linear programming-based rebalancing treats the generation of the circular transaction set as a whole and directly uses the linear programming solver to output a high-dimensional solution. As the problem size grows, the expected time towards reaching a solution and the deviation of the solution from optimality increase. Therefore, we propose DRL-PCR, which takes advantage of DRL to achieve the ability to handle large problem scales and mitigate the risk of getting trapped in a local optimum. There are some pioneering works on the rebalancing problem based on DRL, such as RebEL [13]. However, RebEL uses the rebalancing method of submarine swaps and focuses on maximizing the relay nodes' profits from fees. To the best of our knowledge, DRL-PCR is the first DRL-based circular channel rebalancing algorithm that focuses on restoring the PCN to a balanced state.

III. Model Design

A. Network Model

We model the PCN as a connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, along with a balance matrix $\mathcal{C}$. Here, the nodes $\mathcal{V}$ denote the set of active users involved in the PCN, and the edges $\mathcal{E}$ denote the set of available payment channels between PCN users. Each payment channel is associated with two users who establish the channel. In this model, we consider a PCN with bidirectional channels, such as the Lightning network [14]. In this context, for each $u, v \in \mathcal{V}$, we have $(u, v) \in \mathcal{E} \Leftrightarrow (v, u) \in \mathcal{E}$. The size of balance matrix $\mathcal{C}$ is $|\mathcal{V}| \times |\mathcal{V}|$, where $|\mathcal{V}|$ is the number of PCN users. For each channel $(u, v) \in \mathcal{E}$, the value $\mathcal{C}(u, v) \in \mathbb{R}_0^+$ denotes the maximum amount of coins that can be transferred from user u to user v through channel (u, v). Similarly, the value $\mathcal{C}(v, u) \in \mathbb{R}_0^+$ denotes the maximum amount of coins that can be transferred from user v to user u through the channel. And if there is no direct channel between user u and v, the values in the balance matrix are equal to 0, that is, $\mathcal{C}(u, v) = \mathcal{C}(v, u) = 0$.

In the PCN, each user can make payments to other users by submitting a transaction. For security reasons, a source routing protocol is used for transaction sending. Hence, the transaction sender needs to determine the sending path from the source to the destination. Importantly, PCN transactions are conducted in a consumptive manner and would cause changes in channel balances. For instance, assume user u wants to send x coins to user v and the sending path is p, where $p = \{(u, i_1), (i_1, i_2), \ldots, (i_n, v)\}$ is a list of payment channels. $\{i_1, i_2, \ldots, i_n\}$ are intermediate nodes in the path. If all the channels in p are sufficiently funded, that is, their balances are not less than x, the transaction can be successfully delivered. Once user v receives the transaction, the channels in p will update their balances. For all $(i, j) \in p$, we have $\mathcal{C}(i, j) = \mathcal{C}(i, j) - x$ and $\mathcal{C}(j, i) = \mathcal{C}(j, i) + x$. Note that in this process, the available channel balance is the major bottleneck of PCN throughput. The channel balances would not be automatically recovered like bandwidth, which differs PCN from traditional computer networks.

B. Rebalancing Model

As transactions proceed, the PCN may degenerate into a highly skewed network because the payment demand is not uniform and some payment channels may be used extensively. The rebalancing model in this work serves as the recovery mechanism and works in an *automatic* manner. Before the rebalancing protocol can commence, a leader is elected by a leadership rotation strategy for decision-making, as in [3]. Specifically, the leadership rotation strategy fairly elects the leader based on the PCN nodes' public addresses in the global ledger, which are unique and numeric. Let $\mathcal{V}_l$ denote the set of qualified nodes that would like to participate in the election and $ID(u)$ denote the public identifier of a participant $u \in \mathcal{V}_l$. The first leader is the PCN node u with the smallest identifier $ID(u)$. After the predetermined time has elapsed, a new leader will be elected, which has the smallest ID identifier greater than that of the previous leader. This process is performed on a regular basis. If there is no such successor, the leadership is then passed back to the first ever elected leader, i.e., the node with the smallest identifier. To this end, DRL-PCR is deployed across all election participants. Note that this leader election mechanism has limitations in terms of its security model. This election process is predictable, so malicious attackers can use it to manipulate the rebalancing process. Moreover, if there is no distributed mechanism to implement the leader election or enforce it for non-cooperative users, the leader election would rely on a trusted third party, making it unsuitable for distributed permissionless blockchain-based PCNs. Fortunately, the proposed rebalancing algorithm can be adapted to any leader election mechanism as the decision-making steps can be decoupled from how the leader was chosen. For simplicity, we use the same leader election mechanism as Revive in this work.

The elected leader is responsible for calculating a set of circular transactions from the candidate path set for rebalancing. The candidate paths are all circular paths. For instance, channel $A \to B$ in Fig. 2 is unbalanced, the candidate paths can be $A \to B \to D \to A$ and $A \to B \to C \to D \to A$. Generally speaking, the selection of candidate paths should follow the following principles: 1) The selection should be able to be executed in advance, rather than every time the rebalancing protocol is triggered, to avoid adding additional time overhead. 2) Multiple selections of the same channel should be avoided if possible.

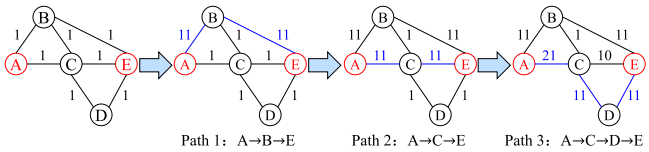


Fig. 3. Illustration of candidate path set generation, where the number next to the channel indicates the weight.

3) Under the premise of following the second point, the shortest path should be chosen if possible. To meet these requirements, we design an algorithm based on the Dijkstra's algorithm to generate the candidate path set. Specifically, a weighted graph is first generated with the same topology as the PCN. The initial weight for each edge is 1, as shown in Fig. 3. Then, we iteratively call the Dijkstra's algorithm to find the path with the smallest weight sum. For the path found at each iteration, we add a positive value to the weights of the edges on the path. The positive value used in Fig. 3 is 10. This process iterates until the number of candidate paths reaches the preset threshold. Note that the selection of candidate paths can be done in advance when the nodes join the network, as it is purely topology-based. In addition, it needs to be clarified that the topology of real-world networks is dynamic, with nodes joining and exiting. In this case, after a period of time, the topology structure may change greatly, and the algorithm needs to be called again to generate a new candidate path set. Each node generates a candidate path set for its own channel separately, and each node's own channel is generally limited, so the time cost of this can be ignored. For simplicity, this dynamic is not considered in this work.

Transaction routing and rebalancing work asynchronously. Whenever a transaction fails, the user automatically reports the unbalanced channel that causes the forwarding failure to the leader. When enough requests are received (e.g., reaching a preset threshold T), the rebalancing algorithm will be triggered and the leader will find a suitable set of circular transactions based on the rebalancing requests and send the calculation results to the corresponding users. For instance, the circular transaction set in Fig. 2 is $(A \rightarrow B \rightarrow D \rightarrow A, 50)$ and $(C \rightarrow D \rightarrow B \rightarrow C, 100)$. Accordingly, these users (i.e., A and C) will initiate circular transactions that eventually lead to a balanced (or as balanced as possible) PCN. In addition, when the number of requests exceeds the threshold T, the leader node may initiate multiple rebalancing processes simultaneously to efficiently rebalance the payment channels.

In this model, the leader node needs to know the instantaneous balance of the channel to be balanced and the channel in the candidate path set for making rebalancing decisions. As pointed out in [15], it is not possible to get large gains in utility by giving up a little privacy, or large gains in privacy by sacrificing a little utility. Therefore, we assume cooperation by all nodes in the rebalancing process and sacrifice a certain degree of privacy in the instantaneous channel balances to achieve large gains in rebalancing performance. This assumption is not outrageous because nodes that initiate rebalance requests have the incentive to balance their channels, and intermediate nodes in the rebalancing paths have the incentive to earn routing fees. We use the same security model as in [3] to guarantee that all users who send the rebalancing requests are honest and will initiate transactions upon receiving the instructions from the leader.

It can be observed that the key to this rebalancing process is to *find a suitable set of circular transactions*, which is the focus of this work. The goal of rebalancing is to restore the PCN to a balanced state or as close as possible. Mathematically, the optimization objective can be formulated as:

$$\max \frac{1}{|\mathcal{E}|} \sum_{(u,v)\in\mathcal{E}} \frac{|\mathcal{C}(u,v)-\mathcal{C}(v,u)|}{\mathcal{C}(u,v)+\mathcal{C}(v,u)} - \frac{|\mathcal{C}'(u,v)-\mathcal{C}'(v,u)|}{\mathcal{C}'(u,v)+\mathcal{C}'(v,u)}, \tag{1}$$

where $\frac{1}{|\mathcal{E}|}$ represents the number of channels in $\mathcal{E}$. $\mathcal{C}(u,v)$ and $\mathcal{C}'(u,v)$ represent the balance matrix before and after rebalancing, respectively. The denominators are used to constrain (1) between -1 and 1, which is widely used in deep learning. $\frac{|\mathcal{C}'(u,v)-\mathcal{C}'(v,u)|}{\mathcal{C}'(u,v)+\mathcal{C}'(v,u)}$ can be considered as the degree of channel imbalance. The more balanced the distribution of channel balance is, the closer $\frac{|\mathcal{C}'(u,v)-\mathcal{C}'(v,u)|}{\mathcal{C}'(u,v)+\mathcal{C}'(v,u)}$ is to 0. Note that the first fraction in (1) is not actually needed if we are to balance the PCN. However, we tend to keep it because it helps to understand the objective function of DRL in the later section.

IV. Algorithm Design

Note that it is not feasible to use brute-force search to solve (1). Mathematically, the time complexity of brute-force search is $O(M^{|\mathcal{E}|})$, where M is the average channel balance. Take the Lightning network as an example, on October 15, 2021 it has 128,984 channels with an average channel balance of 4,066,901 Satoshi. As a compromise, one can apply any common linear programming methods of preference. For instance, Revive [3] directly uses a linear programming library in Python. However, most linear programming algorithms show poor scalability and cannot cope with the participation of a large quantity of PCN users and payment channels. This motivates our design of DRL-PCR.

A. MDP Formulation

First, DRL-PCR transforms the rebalancing problem into a sequential decision problem. It is straightforward to achieve this by dividing the sequence according to the generation of circular transactions, as shown in Fig. 4(a). At the beginning of a sequence, a PCN with a skewed distribution of channel balance is given. Then at each step, a circular transaction is generated and the network updates its channel balance accordingly. This process continues until the PCN is back into equilibrium or the maximum length of the sequence is reached. Notably, this process is consistent with the MDP, which is commonly used to model sequential decision problems in DRL. Mathematically, an MDP can be defined as a four-tuple $< \mathbb{S}, \mathbb{A}, P, R >$. Here, $\mathbb{S}$ is the set of all possible states and $\mathbb{A}$ is the set of all possible actions. $\mathbb{S}$ and $\mathbb{A}$ are also referred to as the state space and action space, respectively. $P(s_{t+1}|s_t, a_t) \rightarrow [0, 1]$ is the state transition probability that maps each state-action pair (s_t, a_t) to the transaction probability to a new state s_{t+1} over the state space

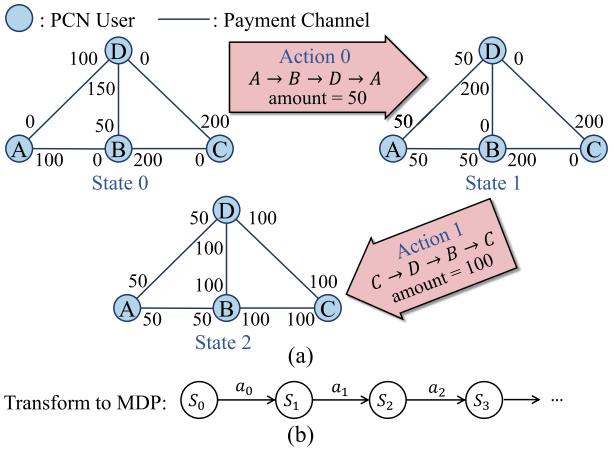


Fig. 4. (a) Process of circular transaction generation is modeled as a sequential decision process. (b) Transform the sequential decision process to an MDP.

$\mathbb{S}$. In addition, $R(s_t, a_t) \to \mathbb{R}$ is the immediate reward function that maps each state-action pair (s_t, a_t) to the immediate reward, which is a numerical value that reflects the performance of the action a under state s.

Let T be the threshold for the number of rebalancing requests. Accordingly, we can have an MDP with T steps. At each step t, an agent generates a circular transaction to resolve the t-th request. Therefore, the corresponding MDP components in the context of rebalancing can be defined as:

1) *State:* The state s_t at step t is the information required to resolve the t-th request, which includes the balance and the node information of the target channel, the candidate path set that can be used to send circular transactions, and the balances of the channels in the candidate paths. In fact, the balance and the node information of the target channel can be omitted as they can be inferred from the candidate path set.
2) *Action:* For better performance, we allow a rebalancing request can be resolved by multiple circular transactions. Let K be the number of paths in the candidate path set. Then we define the action a_t at step t as a vector of length K, in which each value represents the amount of the circular transaction sent to the corresponding candidate path. For instance, if $K = 3$, the candidate path set is $\{p_1, p_2, p_3\}$, and the action $a_t = \{1, 0, 2\}$, the corresponding circular transactions are sending 1 coin to path p_1 and sending 2 coins to path p_3.
3) *Reward:* The reward $R(s_t, a_t)$ is used to evaluate the performance of the action a_t under state s_t. In an MDP, a higher reward typically implies a better action. Hence, for PCN rebalancing, we can define the reward as the difference between the degree of the channel imbalance before and after taking action a_t. Another important metric is the difference in transaction throughput before and after rebalancing, which can also be used as the reward. This metric can be measured in a simulated environment.
4) *State transition probability:* At the end of each step t, the PCN proceeds to the next state based on the generated circular transactions. Specifically, all affected channels update their channel balances according to the network model presented in Section III-A.

Based on the above formulation, the PCN rebalancing problem can be transformed to a general form in the MDP setting. To solve the objective function in (1), the reward $r_t = R(s_t, a_t)$ can be defined as

$$\frac{1}{|\mathcal{E}|} \sum_{(u,v)\in\mathcal{E}} \frac{|\mathcal{C}_t(u,v) - \mathcal{C}_t(v,u)| - |\mathcal{C}_{t+1}(u,v) - \mathcal{C}_{t+1}(v,u)|}{\mathcal{C}_t(u,v) + \mathcal{C}_t(v,u)}, \tag{2}$$

where $\mathcal{C}_t(u,v)$ and $\mathcal{C}_{t+1}(u,v)$ are the channel balances before and after taking action a_t under state s_t. And the final balance matrix after rebalancing in (1), i.e., $\mathcal{C}'$, is the balance matrix at the final step of MDP, i.e., $\mathcal{C}_T$. Compared to using transaction throughput, using the degree of channel imbalance as the reward has the following benefits: 1) Because the update of channel balance is deterministic, it provides a much more stable optimization objective and facilitates stable training. 2) It frees DRL-PCR from high-dimensional state space as only PCN nodes and payment channels in the candidate path set need to be considered. 3) The deterministic nature of the formulated MDP enables offline training, which is generally a lot faster and more cost-efficient than online training (see Section IV-B for details).

Accordingly, (1) can be transformed to the objective function of MDP, i.e., finding a policy π that maximizes the discounted cumulative rewards:

$$\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) | a_t = \pi(s_t)\right]. \tag{3}$$

Here, T is the maximum number of steps in the MDP, which equals the trigger threshold for the number of rebalancing requests. $\gamma \in (0, 1]$ is the discount factor that measures the importance of future states. $R(s_t, a_t)$ is the reward function in (2). π is a parameterized rebalancing policy that maps each state $s_t \in \mathbb{S}$ to an action $a_t \in \mathbb{A}$. π is implemented with a deep neural network in this work. It is worth noting that (3) optimizes the rebalancing performance from a long-term perspective. This is because in MDPs, although the immediate reward for the current action may be low, it may result in a higher reward in the long run. For instance, in Fig. 4(a), although the action 0, i.e., $(A \to B \to D \to A, 50)$, makes the channel $B \to D$ more skewed, it still restores the PCN back to equilibrium at the end of rebalancing. This long-term impact makes it more challenging to find the optimal policy for PCN rebalancing.

B. Overall Architecture

Before diving into the detailed design of DRL-PCR, we need to know how it works. In the context of DRL, our algorithm DRL-PCR considers a general setting where an agent learns to perform rebalancing actions through exploring in an interactive environment, i.e., a PCN. At each step t, the environment builds a state s_t that contains the information required to resolve the

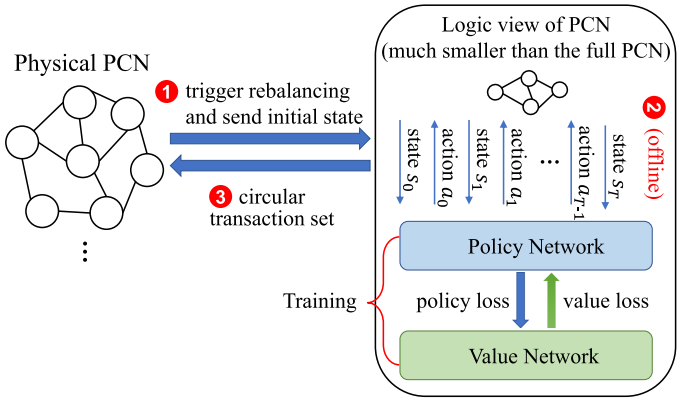


Fig. 5. Overall architecture of DRL-PCR.

t-th rebalancing request. Then, the DRL-PCR agent is responsible to make an action a_t, i.e., a set of circular transactions. According to a_t, the environment updates the channel balances and proceeds to the next state s_{t+1}. A numerical reward r_t is then generated to evaluate the performance of action a_t. The tuple (s_t, a_t, r_t, s_{t+1}) is called *experience* in DRL. This process iterates until all rebalancing requests have been processed. In particular, $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$ constitutes a trajectory and is stored in an experience buffer for later training. Note that this interactive process can be carried out offline without actually executing the circular transaction. This is because the update of channel balance is deterministic and we can accurately simulate this process offline.

In particular, we develop DRL-PCR based on the Actor-Critic framework, which has been widely used in many start-of-the-art DRL algorithms, such as Soft Actor-Critic (SAC) [16] and Proximal Policy Optimization (PPO) [17]. The overall architecture of DRL-PCR is shown in Fig. 5, which includes two parts: the agent-environment interaction part and the training part. In the interaction part, the PCN triggers DRL-PCR for rebalancing and sends the initial state when a preset number of requests are received. Then, DRL-PCR builds a logic view of the PCN, which only includes PCN nodes and payment channels in the candidate path set. This is because the other nodes and channels are not affected by our rebalancing decision and thereby there is no need to consider them in the decision-making. This leads to a much smaller model than the full PCN and saves the rebalancing algorithm from high-dimensional state space. With the logic view of the PCN, DRL-PCR interacts with it in an offline manner. At the end of the interaction, DRL-PCR sends the generated circular transaction set as the rebalancing solution to the PCN. From this point, the circular transactions are truly initiated in the PCN.

In the training part, DRL-PCR consists of two models: an actor and a critic, which are typically implemented with deep neural networks. The actor model corresponds to the policy network and is responsible for choosing actions. The policy network receives the environmental states as the input and outputs a vector representing the action. The critic model, on the other hand, corresponds to the value network and is used to approximate the cumulative reward function. The critic network also takes the environmental states as the input, but outputs a numerical value representing the estimated cumulative reward of the given state. Let π_θ denote the policy network parameterized by θ and V_δ denote the value network parameterized by δ. Therefore, the goal of DRL-PCR is to update the parameters θ and δ towards maximizing the discounted cumulative rewards, i.e., (3).

The reason for choosing the Actor-Critic framework as the foundation of the DRL-PCR is that compared with the traditional value-based frameworks (such as deep Q-learning [18]), the Actor-Critic framework shows great advantages in handling high-dimensional action space. In value-based DRL frameworks, the agent needs to output the expected long-term rewards of all possible actions (or the expected choosing probability of all possible actions), thus having significant limitations in the high-dimensional action space and the continuous action space. In contrast, the agent of the Actor-Critic framework overcomes the curse of dimensionality by directly outputting the target action. Furthermore, the actor model and critic model are complementary to each other during the training process. The actor (i.e., policy network π_θ) propagates the policy loss to the critic (i.e., value network V_δ). On the other hand, the critic propagates the value loss to the actor. The overall architecture with two models learns more efficiently than learning separately [16], [19]. In Section IV-C, we first present the detailed network design of the policy network and value network, revealing how DRL-PCR understands the sophisticated relationship in the PCN topology. In Section IV-D, we detail the training process, especially how the policy network and value network cooperate with each other and progressively improve their parameters.

C. Network Design

1) Motivation: In DRL, the policy network π_θ is responsible to learn a mapping from each state to the action that leads to the optimal performance. The value network V_δ, on the other hand, is responsible to learn a mapping from each state to its cumulative reward. To achieve this, it requires two neural networks to extract valuable features from the input data and understand the sophisticated relationship hidden in the PCN topology. However, off-the-shelf DRL algorithms are not designed to learn from such graph-structured data. They typically apply well-known neural network models such as Fully Connected Layers (commonly used for simple vectors), Convolutional Neural Networks (commonly used for image processing), and Recurrent Neural Networks (commonly used for sequential inputs). These models are strongly limited in processing graph-structured data. As a result, off-the-shelf DRL algorithms fail to understand the sophisticated relationship hidden in the PCN topology and cannot handle the complexity of PCN rebalancing.

Motivated by this, we build the policy network and the value network with a new representation of PCN states and an effective network model based on the GNN model in [20]. GNN has received wide popularity in many node-level, edge-level, and graph-level tasks in recent years, such as social network analysis, link prediction, and graph clustering [21]. In particular, it shows a strong ability in learning knowledge from graph-structured

data. Therefore, DRL-PCR leverages GNN to model the graph-structured data of PCN topology and learn the hidden sophisticated relationship. Specifically, we design DRL-PCR based on GNN, but customized for PCN rebalancing.

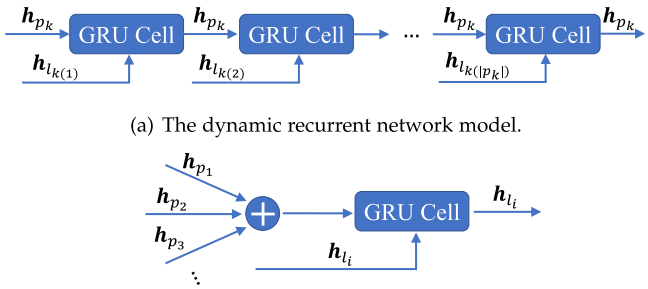
2) Dependencies Between Channels and Paths: In the following, we take the policy network as an example to introduce the network design of DRL-PCR, but note that the value network shares a similar design with the policy network. As mentioned in Section IV-A, the state s_t is the information required to balance the requested channels. Since the balance and the node information of the requested channel can be inferred from the candidate path set, they are omitted from the state s_t. In this case, the state s_t is a vector containing the node IDs and the balances of the channels in the candidate paths, which is typically graph-structured. The task of the policy network π_θ is to learn a mapping from the graph-structured state input to the optimal action. For this purpose, the policy network is expected to extract valuable *features* that contribute to decision-making. The features are typically unknown hidden values that can be learned from the raw data input.

Given that the state inputs are graph-structured, the major obstacle lies in understanding the sophisticated relationship between channels and paths. We can observe that a candidate path is constituted of multiple payment channels and different paths may share the same channels. In this context, it is easy to infer that there is a circular dependency between paths and channels. In principle, we have the following dependencies:

1) *Path-to-channel dependency:* The feature of each candidate path depends on the features of all the payment channels in the path.
2) *Channel-to-path dependency:* The feature of each payment channel depends on the features of all the candidate paths containing the channel.
3) *Path-to-path dependency:* If the channels of two candidate paths intersect, then there is a dependency between these two paths.
4) *Channel-to-channel dependency:* If two payment channels appear on the same path, then there is a dependency between these two channels.

The above dependencies make it impractical to directly apply off-the-shelf DRL algorithms for PCN rebalancing, since they are not designed to deal with the sophisticated dependencies in state s_t. Intuitively, DRL-PCR is expected to operate over graph-structured data based on their dependencies. This leads to the GNN-based network model of DRL-PCR.

3) GNN-Based Network Model: First, we provide some necessary notations. For clarity, we omit the subscript t in the following, e.g., s for s_t. Let $\mathbb{P} = \{p_1, p_2, \ldots, p_K\}$ denote the candidate path set of s and $\mathbb{L}$ denote the set of payment channels in $\mathbb{P}$, where K is the number of candidate paths. On the one hand, each path $p_k \in \mathbb{P}$ is a sequence of payment channels, i.e., $p_k = \{l_{k(1)}, l_{k(2)}, \ldots, l_{k(|p_k|)}\}$, where $k(i)$ is the index of the i-th channel of the path p_k and $|p_k|$ is the number of the channels in p_k. On the other hand, each payment channel l_i is also associated with a set of paths that contain l_i, denoted as $\mathbb{P}_{l_i} = \{p_1, p_2, \ldots, p_{|\mathbb{P}_{l_i}|}\}$. For each path $p_j \in \mathbb{P}_{l_i}$, we have $l_i \in p_j$. In addition, the action a is a vector of size K, i.e., $a = (y_{p_1}, y_{p_2}, \ldots, y_{p_K})$, where y_{p_k} is the amount of circular transaction sent to path p_k.



(a) The dynamic recurrent network model.

(b) The simple aggregation model.

Fig. 6. Network models to extract: (a) the feature of path $\boldsymbol{h}_{p_k}$; (b) the feature of the channel $\boldsymbol{h}_{l_i}$.

Let $\boldsymbol{h}_{p_k}$ and $\boldsymbol{h}_{l_i}$ denote the *features* of the path p_k and channel l_i. According to the path-to-channel dependency and the channel-to-path dependency, we can build the following expressions:

$$\boldsymbol{h}_{p_k} = f(\boldsymbol{h}_{l_{k(1)}}, \boldsymbol{h}_{l_{k(2)}}, \ldots, \boldsymbol{h}_{l_{k(|p_k|)}}), \tag{4}$$

$$\boldsymbol{h}_{l_i} = g(\boldsymbol{h}_{p_1}, \boldsymbol{h}_{p_2}, \ldots, \boldsymbol{h}_{p_{|\mathbb{P}_{l_i}|}}), \tag{5}$$

where $f(\cdot)$ and $g(\cdot)$ are learnable unknown functions. The subscripts $\{p_1, p_2, \ldots, p_{|\mathbb{P}_{l_i}|}\}$ are used to identify the candidate paths containing the channel, which are the actual indices of the paths in the candidate path set $mathbbP$. As shown in (4) and (5), the feature of path p_k is dependent to the features of the payment channels in the path, i.e., $\{l_{k(1)}, l_{k(2)}, \ldots, l_{k(|p_k|)}\}$. Meanwhile, the feature of channel l_i is dependent to the features of all the candidate paths containing the channel, i.e., $\{p_1, p_2, \ldots, p_{|\mathbb{P}_{l_i}|}\}$.

Considering that a path is defined by a sequence of channels, there is a sequential dependency in the channels that form the path. In addition, the function $f(\cdot)$ should be invariant to the topological model and able to handle paths of arbitrary length. Therefore, DRL-PCR implements $f(\cdot)$ with a Dynamic Recurrent Network Model (DRNM), which is effective in handling arbitrary sequences of inputs and extracting sequential dependence. In addition, DRNM overcomes the curse of dimensionality and compresses arbitrary input sequences into fixed-dimensional features by reusing a shared network module. The model to extract $\boldsymbol{h}_{p_k}$ is shown in Fig. 6(a), the Gate Recurrent Unit (GRU) is reused and processes the features of the channels that make up the path in a sequential manner. Mathematically, the feature of path p_k can be calculated by:

$$\boldsymbol{h}_{p_k} \leftarrow \text{GRUCell}_1\left(\boldsymbol{h}_{p_k}, \boldsymbol{h}_{l_{k(i)}}\right), \quad \text{for } i = 1, 2, \ldots, |p_k|, \tag{6}$$

where $|p_k|$ is the number of channels in path p_k. $\text{GRUCell}_1(\cdot)$ is a shared module that extracts features from each part of the input sequence in order. The GRUCell [22] is the standard component used in RNNs and their variants, which forms the basis for processing sequential data. Equation (6) sequentially feeds the $\text{GRUCell}_1(\cdot)$ with the features of the channels constituting the

path and the output vector in the previous step. In addition, the initial input vector $\boldsymbol{h}_{p_k}$ to the $\text{GRUCell}_1(\cdot)$ is the feature of the path p_k. Specifically, $\boldsymbol{h}_{p_k}$ contains the basic information of the path and is set as $\boldsymbol{h}_{p_k} = [\boldsymbol{x}_{p_k}, 0, \ldots, 0]$, where $\boldsymbol{x}_{p_k}$ is a vector composed of the node IDs and balances of all channels in the path p_k. Considering that the lengths of paths are variable and the input of $\text{GRUCell}_1(\cdot)$ is fixed, we initialize $\boldsymbol{h}_{p_k}$ with sufficient size and add zero to the extra space.

On the other hand, considering that the order of the paths containing the channel l_i does not matter, DRL-PCR uses a Simple Aggregation Model (SAM) to implement $g(\cdot)$, which aggregates the features of all relative paths and then passes the result to a GRU, as shown in Fig. 6(b). Mathematically, the feature of channel l_i can be calculated by:

$$\boldsymbol{h}_{l_i} \leftarrow \text{GRUCell}_2\left(\boldsymbol{h}_{l_i}, \sum_{p \in \mathbb{P}_{l_i}} \boldsymbol{h}_p\right), \tag{7}$$

where $\mathbb{P}_{l_i} = \{p_1, p_2, \ldots, p_{|\mathbb{P}_{l_i}|}\}$ is the set of paths that contains channel l_i. Similar to $\boldsymbol{h}_{p_k}$, the initial input vector $\boldsymbol{h}_{l_i}$ to the $\text{GRUCell}_2(\cdot)$ contains the basic information of the channel l_i and is initialized to $\boldsymbol{h}_{l_i} = [\boldsymbol{x}_{l_i}, 0, \ldots, 0]$, where $\boldsymbol{x}_{l_i}$ is a vector composed of the node IDs associated with the channel and the channel balances in both directions. $\boldsymbol{h}_{l_i}$ is also initialized with sufficient size by adding zero to the extra space.

However, as shown in (6) and (7), $f(\cdot)$ and $g(\cdot)$ do not reflect the path-to-path dependency and channel-to-channel dependency. In this context, directly learning $f(\cdot)$ and $g(\cdot)$ may face the risk of information loss. For instance, if we use a fully connected layer to approximate $f(\cdot)$ and $g(\cdot)$, the message passing from input to output is one-way. As mentioned, because the channels of two candidate paths may intersect and two payment channels may appear on the same path, there are circular dependencies between paths and channels. The fully connected layer may fail to deal with the graph-structured data effectively and lose valuable features for decision-making.

To deal with circular dependencies, DRL-PCR processes the graph-structured input data with a loop operation. Specifically, DRL-PCR repeats the messaging passing operation (i.e., repeatedly feeds $f(\cdot)$ with the channel features and $g(\cdot)$ with the path features) until a preset number of times M is reached. We use the superscript m to represent the m-th iteration. The features of path p_k and channel l_i at the m-th iteration are denoted as $\boldsymbol{h}_{p_k}^m$ and $\boldsymbol{h}_{l_i}^m$, respectively. At the beginning of the loop operation, the path feature $\boldsymbol{h}_{p_k}^0$ is initialized as:

$$\boldsymbol{h}_{p_k}^0 = [\boldsymbol{x}_{p_k}, 0, \ldots, 0]. \tag{8}$$

Similarly, the channel feature $\boldsymbol{h}_{l_i}^0$ is initialized as:

$$\boldsymbol{h}_{l_i}^0 = [\boldsymbol{x}_{l_i}, 0, \ldots, 0] \tag{9}$$

Then, for each iteration in the loop, DRL-PCR feeds the $\text{GRUCell}_1(\cdot)$ and $\text{GRUCell}_2(\cdot)$ with the calculated features of the previous iteration. Mathematically, the path features $\boldsymbol{h}_{p_k}^m$ at the m-th iteration can be formulated as:

$$\boldsymbol{h}_{p_k}^m \leftarrow \text{GRUCell}_1\left(\boldsymbol{h}_{p_k}^m, \boldsymbol{h}_{l_{k(i)}}^{m-1}\right), \quad \text{for } i = 1, 2, \ldots, |p_k|, \tag{10}$$

Here, $\boldsymbol{h}_{p_k}^m$ is initialized to $\boldsymbol{h}_{p_k}^{m-1}$ at the beginning of the for-loop (i.e., when $i = 1$). Similarly, the channel feature $\boldsymbol{h}_{l_i}^m$ at the m-th iteration can be formulated as:

$$\boldsymbol{h}_{l_i}^m \leftarrow \text{GRUCell}_2\left(\boldsymbol{h}_{l_i}^{m-1}, \sum_{p \in \mathbb{P}_{l_i}} \boldsymbol{h}_p^m\right). \tag{11}$$

As shown in (10) and (11), DRL-PCR can be interpreted as an augmentation of vanilla message passing neural networks, which is tailored for capturing the complicated circular dependencies between paths and channels. This forward propagation of the GNN-based neural network model of DRL-PCR is shown in Fig. 7. At the beginning of the loop operation, DRL-PCR is given with the channel and path features initialized by the raw state input. For each iteration, DRL-PCR first feeds the DRNM with the previous channel features and previous path features to calculate new path features, as in (10). Second, DRL-PCR feeds the SAM with the previous channel features and the new path features to calculate the new channel features, as in (11). Note that the first two steps are calculated based on how the paths are constructed from the channels. For instance, as in Fig. 7, the path feature $\boldsymbol{h}_{p_1}^m$ is calculated by the channel features $\boldsymbol{h}_{l_1}^{m-1}$ and $\boldsymbol{h}_{l_2}^{m-1}$, which in turn is used to calculate the latest channel features $\boldsymbol{h}_{l_1}^m$ and $\boldsymbol{h}_{l_2}^m$. Third, DRL-PCR replaces the channel features and path features with the latest values for the next iteration. This process continues until the maximum number of iterations M is reached. This loop operation enables DRL-PCR to handle circular dependencies and arbitrary PCN topologies at the same time.

After using the GNN-based network model for M iterations, we have the latest path features, which can be denoted by $H^M = \{\boldsymbol{h}_{p_k}^M | p_k \in \mathbb{P}\}$. As mentioned in the MDP model of Section IV-A, the action a is a vector of length K, in which each value represents the amount of circular transactions sent to the corresponding candidate path. Hence, DRL-PCR takes the latest path features H^M as input and uses two readout modules to output the expectation and variance of the action distribution, i.e.,

$$\mu = \text{Readout}_1(H^M), \tag{12}$$

$$\sigma = \text{Readout}_2(H^M). \tag{13}$$

In deep learning, a "readout module" typically refers to a component or layer that is responsible for producing the final output or prediction of the network. In DRL-PCR, the readout modules use a linear layer, a Long Short-Term Memory (LSTM) layer, and a linear layer in turn to process the input. With μ and σ, the policy network outputs the final action vector by sampling from a normal distribution, which is a common practice in DRL.

Algorithm 1 summarizes the forward propagation of DRL-PCR. For each input state s, DRL-PCR outputs the action vector with a loop operation. At the beginning of the loop operation, DRL-PCR uses (8) and (9) to initialize the path features $\boldsymbol{h}_{p_k}^0$ and channel features $\boldsymbol{h}_{l_i}^0$ (line 1). Then, DRL-PCR uses M-round iterations to capture circular dependencies and generate path features (line 3). For each iteration, the DRNM of DRL-PCR

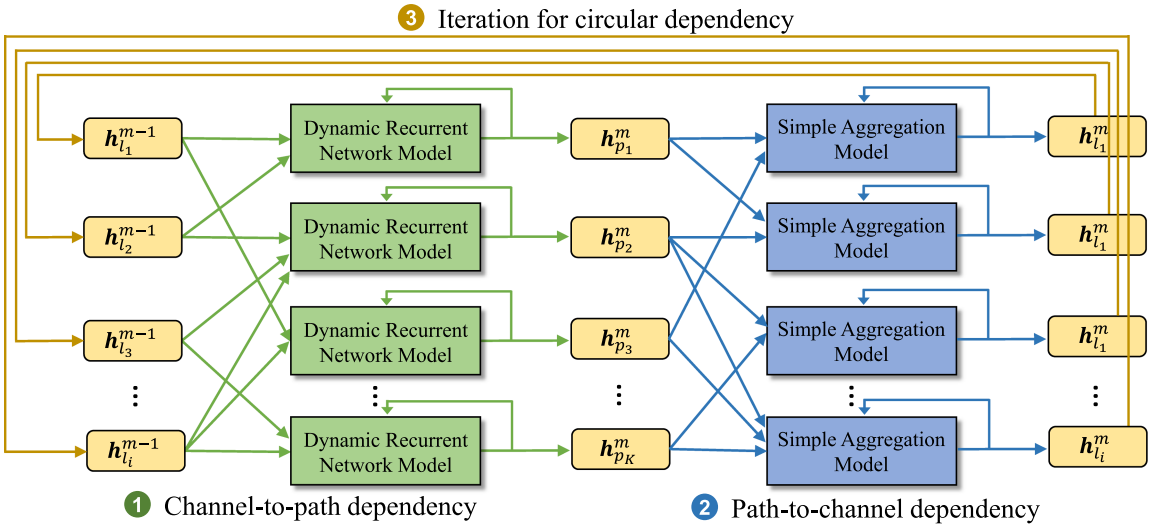


Fig. 7. Detailed design of the GNN-based network model, which processes the graph-structured input data with a loop operation.

uses (10) to update the path features (lines 4–9). Then, the SAM of DRL-PCR uses (11) to update the channel features (lines 10–12). This iteration repeats until the maximum number of iterations M is reached (lines 3–13). After M iterations, DRL-PCR concatenates the latest path features H^M and uses two readout modules to output the expectation μ and variance σ of the action distribution (lines 14–15). With μ and σ, the policy network samples the final action vector from a normal distribution (line 16).

D. Training

The above GNN-based network model shows how the DRL-PCR works with graph-structured input states in the forward propagation process. Leveraging the strong approximation ability of GNN, a well-trained DRL-PCR agent should be able to capture the circular dependencies from the states and find the optimal actions. In this section, we present the training process of DRL-PCR.

As presented in Section IV-B, the overall architecture of DRL-PCR consists of two models, a policy network denoted by π_θ and a value network denoted by V_δ. Here, θ and δ are learnable network parameters, which are typically composed of the network parameters in the forward model in Section IV-C. The policy network π_θ is responsible to learn a mapping from each state to the action that leads to the optimal performance. In turn, the value network V_δ is responsible to learn a mapping from each state to the cumulative reward. To achieve this goal, DRL-PCR updates the value network towards approximating the cumulative reward function and uses the value network to guide the update of the policy network.

Specifically, the training of DRL-PCR performs in episodes, where an episode τ consists of a sequence of agent-environment interactions and can be denoted as $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$. For the value network, it is responsible to output the average value of the expected cumulative rewards for each given state. With the trajectory τ, the target value can be easily calculated with the weighted sum of rewards. In this regard, the parameters δ of value network V_δ can be updated by minimizing the mean squared error between the target value and the prediction of V_δ. Mathematically, given a batch of trajectories $\mathcal{G} = \{\tau_1, \tau_2, \ldots\}$, the loss value to update δ can be formulated as:

$$L(\delta) = \frac{1}{|\mathcal{G}|} \sum_{\tau \in \mathcal{G}} \sum_{t=0}^{T} \left(\sum_{i=0}^{T-t} \gamma^i r_{t+i} - V_\delta(s_t) \right)^2, \tag{14}$$

where $\sum_{i=0}^{T-t} \gamma^i r_{t+i}$ is the target value and $V_\delta(s_t)$ is the prediction of value network. $\gamma \in (0, 1]$ is the discount factor that measures the significance of future states. Obviously, the lower the loss value $L(\delta)$, the more accurate the prediction of long-term rewards and the better we can guide the update of the policy network.

For the policy network, we want to improve its policy so that it can output better and better actions over time. Driven from the Proximal Policy Optimization [17], DRL-PCR updates the policy network with three important techniques, that is, probability ratios based on importance sampling, advantage function, and clipped policy gradient. First, DRL-PCR uses importance sampling to constrain the policy gap between the behavior policy and the policy to be updated. The behavior policy, denoted by π_{behavior}, is the policy that collects the trajectory τ. Considering that the policy network is continuously updated, it will gradually deviate from the behavior policy. In this case, if there is a large gap between π_θ and π_{behavior}, the training data (i.e., trajectory τ) would be obsolete and endanger the training stability [23]. Therefore, it is necessary to constrain the policy gap to a trust region. To achieve this, importance sampling uses a probability ratio to measure the gap between π_{behavior} and π_θ. Given an experience tuple (s_t, a_t, r_t, s_{t+1}), the probability ratio is defined as $p_t = \pi_\theta(a_t|s_t)/\pi_{\text{behavior}}(a_t|s_t)$, where $\pi_\theta(a_t|s_t)$ is the probability that the policy π_θ chooses action a_t under state s_t. In DRL, the probability ratio is generally expressed in logarithmic

Algorithm 1: The Forward Propagation of DRL-PCR.

Input: State s, candidate path set $\mathbb{P}$, payment channel set $\mathbb{L}$.

For each path and payment channel, construct $\boldsymbol{x}_{p_k}$ and $\boldsymbol{x}_{l_i}$ from s.
For each path and payment channel, initialize the features as $\boldsymbol{h}_{p_k}^0 \leftarrow [\boldsymbol{x}_{p_k}, 0, \ldots, 0]$ and $\boldsymbol{h}_{l_i}^0 \leftarrow [\boldsymbol{x}_{l_i}, 0, \ldots, 0]$, respectively.
// Iterations for circular dependencies.
for $m = 1, \ldots, M$ **do**
// For path-to-channel dependency.
  for $p_k \in \mathbb{P}$ **do**
    Set $\boldsymbol{h}_{p_k}^m \leftarrow \boldsymbol{h}_{p_k}^{m-1}$.
    for $i = 1, 2, \ldots, |p_k|$ **do** ,
      Update the path features as: $\boldsymbol{h}_{p_k}^m \leftarrow \text{GRUCell}_1(\boldsymbol{h}_{p_k}^m, \boldsymbol{h}_{l_{k(i)}}^{m-1})$.
    end for
  end for
// For channel-to-path dependency.
  for $l_i \in \mathbb{L}$ **do**
    Update the channel features as: $\boldsymbol{h}_{l_i}^m \leftarrow \text{GRUCell}_2(\boldsymbol{h}_{l_i}^{m-1}, \sum_{p \in \mathbb{P}_{l_i}} \boldsymbol{h}_p^m)$.
  end for
end for
Concatenate the latest path features $H^M = \{\boldsymbol{h}_{p_k}^M | p_k \in \mathbb{P}\}$.
Estimate the expectation and variance of the action distribution: $\mu = \text{Readout}_1(H^M)$, $\sigma = \text{Readout}_2(H^M)$.
Sample the final action vector a from a normal distribution with μ and σ.
returnaction a

form, i.e.,

$$p_t = \exp\left(\log \pi_\theta(a_t|s_t) - \log \pi_{\text{behavior}}(a_t|s_t)\right), \tag{15}$$

where $\exp(\cdot)$ is the exponential function. Using the ratio p_t, DRL-PCR can determine how much gap we can tolerate between π_{behavior} and π_θ.

Second, DRL-PCR uses the advantage function to obtain a low-variance estimation of agent performance. It is well-known that the vanilla policy gradient-based DRL suffers from high gradient variance and poor stability. This is because it directly uses the discounted cumulative rewards to measure the agent's performance, where the bias is zero but the variance is high [24]. In contrast, the advantage function measures how good an action is compared to the average performance for a given state, which can be regarded as a low-variance relative evaluation with estimation bias. In essence, the use of the advantage function is to make a trade-off between the estimation bias and gradient variance.

There are many implementations of advantage functions in DRL. In this work, we use one of the most popular versions, i.e., the Generalized Advantage Estimation (GAE) [25]. In GAE, the advantage function trade-offs the variance and bias by calculating the cumulative rewards for J steps forward. For an experience tuple (s_t, a_t, r_t, s_{t+1}), the advantage value can be expressed as follows:

$$A_t = \sum_{j=0}^{J} \gamma^j \left[r_t + \gamma V_\delta(s_{t+1}) - V_\delta(s_t)\right]. \tag{16}$$

Here, γ is the discount factor and J is the number of steps that the advantage function looks forward. If J equals the maximum number of steps T, it becomes the vanilla policy gradient-based DRL that has zero bias but high variance. If J equals zero, it becomes one-step optimization that has zero variance but high bias (since we focus on long-term optimization). In practice, J is a constant between 1 and the length of the episode. Intuitively, if the advantage value A_t is positive, i.e., $r_t + \gamma V_\delta(s_{t+1}) > V_\delta(s_t)$, it means the output action of the policy network is good and can achieve a better-than-average reward. In this case, we can increase the action selection probability for better performance. Otherwise, if the advantage value A_t is negative, we should decrease the action selection probability of a_t.

Third, DRL-PCR combines the probability ratio and advantage function to construct a clipped surrogate gradient as the policy loss. Specifically, the policy loss takes the minimum of two objective values, L_t^{CPI} and L_t^{CLIP}. On the one hand, L_t^{CPI} simply multiplies the probability ratio and the advantage value, i.e.,

$$L_t^{CPI} = p_t \cdot A_t. \tag{17}$$

The superscript CPI refers to Conservative Policy Iteration [26]. Note that if there is a large gap between the current policy π_θ and behavior policy π_{behavior}, the probability ratio p_t would be excessively large, leading to drastic updates and poor stability. In view of this, L_t^{CLIP} constrains p_t within a small interval around 1 with a clip operation, which can be defined as:

$$L_t^{CLIP} = \text{clip}(p_t, 1-\epsilon, 1+\epsilon) \cdot A_t, \tag{18}$$

where:

$$\text{clip}(p_t, 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon & r_t < 1-\epsilon \\ 1+\epsilon & r_t > 1+\epsilon \\ p_t & \text{Otherwise} \end{cases}. \tag{19}$$

The clipping parameter ϵ is generally a small value and is suggested to be 0.2 in [17]. Finally, DRL-PCR takes the minimum of L_t^{CPI} and L_t^{CLIP} as the final policy loss. Given a batch of trajectories $\mathcal{G} = \{\tau_1, \tau_2, \ldots\}$, the policy loss can be formulated as:

$$L(\theta) = -\frac{1}{|\mathcal{G}|} \sum_{\tau \in \mathcal{G}} \sum_{t=0}^{T} \left[\min\left(L_t^{CPI}, L_t^{CLIP}\right)\right]. \tag{20}$$

In practice, DRL-PCR combines the value loss and policy loss to form the final loss value. In addition, an entropy term is added to encourage DRL-PCR to explore different policies. A higher entropy means the policy is more random and the agent is more exploratory, while a lower entropy means the policy is more stable, but it may also lose the opportunity to find the optimal action. Therefore, DRL-PCR adds an annealing entropy term to the final loss value. In the early training stage, the entropy value is large, which encourages policy exploration and avoids

Algorithm 2: The Training of DRL-PCR.

Input: Policy network π_θ, initial value network V_δ.

1: **Initialize** parameters θ and δ with random small values close to zero.
2: **for** training iteration $= 1, 2, \dots$ **do**
3: Sample a batch of trajectories $\mathcal{G} = \{\tau_1, \tau_2, \dots\}$, which are collected by running the forward propagation of DRL-PCR in the environment.
4: Compute the value loss $L(\delta)$ based on (14).
5: Compute the clipped policy loss $L(\theta)$ based on (20).
6: Compute the policy entropy $\text{Ent}(\pi_\theta)$ based on (22).
7: Compute the final loss L^{Final} based on (21).
8: Update the network parameters θ and δ by minimizing the final loss L^{Final} with the Adam optimizer.
9: **end for**

getting trapped in a local optimum. As the training proceeds, the entropy decreases gradually, and finally the algorithm converges to a stable policy. Mathematically, the final loss value can be formulated as:

$$L^{Final} = L(\theta) + c_1 L(\delta) - c_2 \text{Ent}(\pi_\theta), \tag{21}$$

where $L(\theta)$ and $L(\delta)$ are the policy loss and value loss, respectively. $\text{Ent}(\pi_\theta)$ is the entropy of policy π_θ. c_1 is a constant coefficient for the value loss and c_2 is the entropy coefficient, which decreases as the training proceeds. Given a batch of trajectories, $\text{Ent}(\pi_\theta)$ can be calculated by

$$\text{Ent}(\pi_\theta) = -\frac{1}{|\mathcal{G}|} \sum_{\tau \in \mathcal{G}} \sum_{t=0}^{T} \sum_{a_i \in \mathbb{A}} \pi_\theta(a_i|s_t) \log \pi_\theta(a_i|s_t). \tag{22}$$

Algorithm 2 summarizes the training process of DRL-PCR. First, DRL-PCR samples a batch of trajectories $\mathcal{G} = \{\tau_1, \tau_2, \dots\}$ from the experience buffer, where $\mathcal{G}$ are collected by running the forward propagation of DRL-PCR in the environment (line 3). With $\mathcal{G}$, DRL-PCR computes the value loss $L(\delta)$, clipped policy loss $L(\theta)$, and the policy entropy $\text{Ent}(\pi_\theta)$ according to (14), (20), and (22), respectively (lines 4-6). By combining these three terms, DRL-PCR computes the final loss value L^{Final} based on (21) and updates the parameters by minimizing L^{Final} (lines 7-8). The Adam optimizer is used for parameter updates. After the update, DRL-PCR starts a new training iteration until the maximum number of iterations is reached or the network converges. In addition, because the formulated MDP model of rebalancing is deterministic, the entire training process uses offline training instead of online training. Offline training can be a useful technique for training agents in complex environments where online interaction is difficult or costly. This is particularly important in the context of PCN, where online training can be difficult and costly due to the high stakes involved.

E. Analysis of Time Cost

Lastly, we analyze the time cost of DRL-PCR, which can be divided into two parts: training time and running time. On the one hand, the training of DRL-PCR can be conducted in an offline manner, which will not affect online transactions. This is because we carefully design the rebalancing model as a deterministic MDP and define the reward as the degree of channel imbalance. In this case, we can accurately simulate the rebalancing process offline. Even if the prediction effect may be poor, no user would suffer from the resulting costs. In addition, we implement DRL-PCR with a customized GNN-based model to improve learning efficiency, leading to an acceptable convergence rate (see Section V-B1 for experiment results). On the other hand, the running time consists of:

1) *Leadership election overhead:* As mentioned in Section III-B, the leadership election only needs to be conducted every period of time. Hence, although it requires all qualified nodes to exchange information and reach consensus through the network, it does not introduce additional overhead for rebalancing operations.
2) *Encoding overhead:* At each rebalancing step, DRL-PCR encodes the input state, extracts features, and output the action. As shown in the MDP model in Section IV-A, only the nodes on the candidate paths are involved in this encoding process. This saves the rebalancing algorithm from high-dimensional state space. More importantly, the encoding tasks can be accomplished with a small-scale network and the time cost is in the millisecond level.

Note that when the number of requests exceeds the threshold T, the leader node may initiate multiple rebalancing processes in parallel, which increases the decision cost. However, this problem can be easily solved in many ways. For instance, learning from the idea of blockchain sharding [27], we can divide the whole payment channel network into multiple shards, where each shard will independently elect its own leader node. Each leader node processes rebalancing requests from nodes in the shard it is responsible for in parallel. In this way, the decision cost on the leader nodes can be apportioned and reduced.

V. Evaluation

A. Evaluation Setup

In this section, we run evaluations with historical data from real-world PCNs to measure the performance of DRL-PCR. The evaluations are conducted on a desktop with an Intel Core i5-7500 3.4 GHz CPU and 16 GB memory. The environment, i.e., PCN, is implemented with the NetworkX package of Python. Since we focus on PCN throughput, the security mechanisms such as HTLC are not implemented in our environment. To initialize the PCN, we crawl the active nodes and payment channels of Ripple and Lightning networks on November 3, 2021 and October 15, 2021, respectively. We remove nodes with no channel connection and nodes with only a single neighbor since these nodes cannot be rebalanced. For Ripple, we get a PCN with 259 nodes and 1,162 channels. And for Lightning, we get a PCN with 9,021 nodes and 128,984 channels. Similar to [8], [9], we evenly distribute the crawled channel balances to both sides as initialization. The average channel balances of Ripple

and Lightning networks are 13,836,327 Drop[1] and 4,066,901 Satoshi.[2]

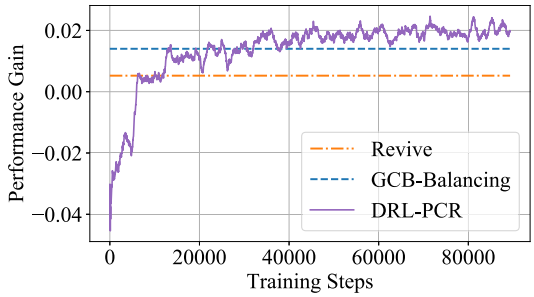
To model a dynamic PCN, we keep generating transactions and executing them in the PCN. Similar to [9], to simulate the unbalanced workload in the PCN, we use an exponential distribution to sample the senders and receivers of the transactions so that some nodes have a higher probability of being selected as senders or receivers. In addition, we remove sender-receiver pairs that are unreachable. As for the amount of the generated transactions, it is uniformly sampled from the dataset of real-world historical transactions. Note that since transactions on the Lightning network are not publicly available, we adopt a similar approach to [8] and use the onchain Bitcoin transactions in [28]. If not specified, the default number of generated transactions is 1,000 by default. The generated transactions are routed according to the shortest-path-first algorithm. As described in the rebalancing model of Section III-B, when a transaction fails to route, the elected leader will receive a rebalancing request. When the number of requests reaches a preset threshold T, DRL-PCR is triggered for channel rebalancing. The value of T is set as 25. The positive value for candidate path generation is 10, the size of the candidate path set is 5, and the maximum length of each candidate path is 6. For the computed candidate paths, the average path length in the Ripple network is 3.46 and in the Lightning network is 3.87.

For DRL-PCR, the value network and policy network share a similar structure. In both networks, the dimensions of path features and channel features are set as 32 and the iteration number M for circular dependencies is 5. The difference between the two networks is the final readout module. The value network maps each state to a single value representing the long-term reward, thereby its readout module consists of a linear layer of size $[32, 8]$, an LSTM layer of size $[40, 128]$, and a linear layer of size $[128, 1]$. The policy network maps each state to the expectation μ and variance σ of the action distribution, thereby it has two corresponding readout modules similar to the value network, except that the final linear layer is of size $[128, 5]$. In addition, the discount factor γ is 0.95, the size of the experience buffer is 20,000 and the batch size of training trajectories $\mathcal{G}$ is 128. During training, the maximum gradient norm is 5. The clipping parameter ϵ for the policy loss is 0.2. For the value coefficient and entropy coefficient in L^{Final}, we set $c_1 = 0.25$ and $c_2 = 0.2$. Lastly, the Adam optimizer is used for parameter updates, where the learning rate is 1×10^{-3}. Our code is available at: github.com/qiuxy23/rebalancing.

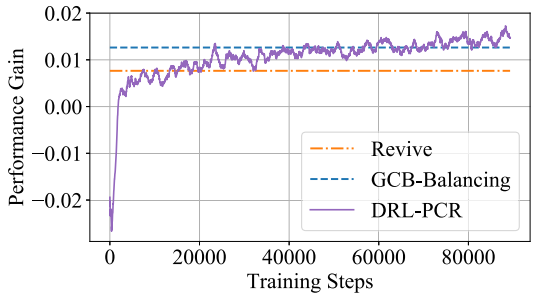
For comparison, we compare DRL-PCR with the following three baselines:

1) *Without Rebalancing (w/o R.):* This baseline processes the generated transactions with no rebalancing mechanism. In this case, failed transactions do not get another chance to be completed.
2) *GCB-Balancing:* GCB-Balancing [4] is a state-of-the-art rebalancing algorithm in PCNs. Similar to DRL-PCR, GCB-Balancing models the rebalancing problem as a sequential decision problem, except that it solves the problem with a greedy algorithm. In each step of the sequence, it attempts to maximize the current reward.
3) *Revive [3]:* Revive is a state-of-the-art rebalancing algorithm in PCNs, which relies on the linear programming solver in Python to approximate the optimal solution. There may be approximation errors for large solution space.

[1]The smallest denomination of Ripple.
[2]the smallest denomination of Bitcoin.



(a) Ripple.



(b) Lightning.

Fig. 8. Convergence results of DRL-PCR.

For evaluation metrics, we use the difference in the degree of channel imbalance before and after rebalancing as the *performance gain* of the rebalancing algorithms, which is defined in (1). In addition, similar to [8], [9], we also use the *throughput* and *success ratio* as the evaluation metrics, where the throughput is defined as the overall amount of successful transactions and the success ratio is defined as the ratio of the number of successful transactions to the number of all generated transactions.

B. Evaluation Results

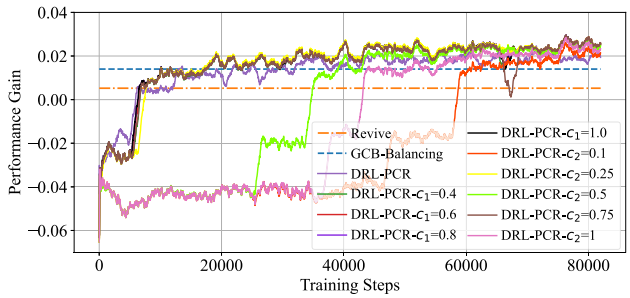
1) Convergence Behavior: First, to evaluate the convergence performance of DRL-PCR, we analyze the dynamic changes of the performance gain throughout the whole training process. Fig. 8 shows the convergence curves on the Ripple and Lightning networks. At the beginning of training, DRL-PCR has a negative performance gain. This is because the network parameters θ and δ are randomly initialized. Nonetheless, the performance gain of DRL-PCR continues to improve as training progresses and eventually surpasses all baselines. In particular, the performance gain rises rapidly in the early stages, implying that the effect of random initialization is small. The performance gains of GCB-Balancing and Revive are also plotted in Fig. 8, which are static and invariant against the number of training steps. The

performance gain of w/o R. is 0 because it has no rebalancing mechanism and is therefore omitted from the Fig. 8. In Ripple, DRL-PCR outperforms all baselines after 32,000 training steps, where the convergence result is around 0.021 and is 50% better than the best baseline (i.e., GCB-Balancing). And in Lightning, DRL-PCR outperforms all baselines after 70,000 training steps, where the convergence result is around 0.015 and is 25% better than the GCB-Balancing algorithm. This indicates that the delicate network design of DRL-PCR empowers it with good training stability and superior convergence results. Note that although the performance gain in Fig. 8 may appear low, it does not mean that the rebalancing algorithm can only slightly help in terms of pushing the system back to equilibrium. This is because the performance gain defined in (1) is divided by the number of channels in the candidate path set. To further evaluate the performance of the algorithm, we next evaluate the improvement in transaction throughput and success ratio under different experimental settings.

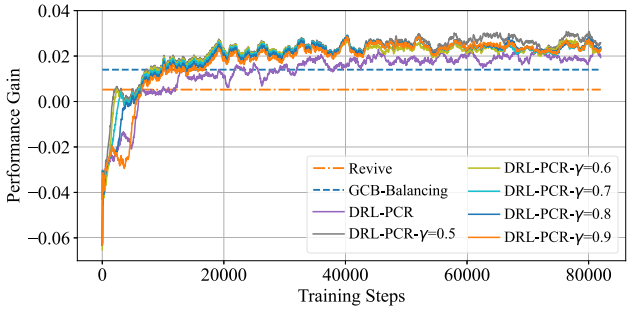
We also study the performance gap between DRL-PCR and the upper bound. Because the payment channel rebalancing problem is NP-hard problem [29], it is impractical to use brute-force search to obtain the optimal solution. As a compromise, we use the interior point method to approximate the optimal solution and do not impose a limit on the runtime of the solver. Specifically, the interior point method works in an iterative manner. We repeat the iteration and observe the performance of the solution obtained in each iteration. The stopping criteria is that the performance does not improve within a certain number of steps (similar to the convergence in DNN training). The experiment results show that the near-optimal performance gains are about 0.032 and 0.027, respectively. Although our algorithm has not yet reached the upper bound of performance, the DRL response time is in the millisecond range. In contrast, it takes about 1-10 seconds to search the near-optimal solutions.

In addition, to study how sensitive are the results to the hyper-parameters of DRL-PCR, we run evaluations with different value coefficients c_1, different entropy coefficients c_2, and different discount factors γ. Fig. 9 shows the convergence curves on the Ripple network, where DRL-PCR uses the default parameters in Section V-A, DRL-PCR-c_1=0.4 means changing c_1 to 0.4, and other parameters use the default settings. It can be seen from the results that DRL-PCR has high robustness. The convergence speeds in most cases are not affected by the hyper-parameter setting, and the final convergence results of all algorithms are higher than that of the baselines.

2) Performance Under Different Workloads: Next, we investigate the robustness of DRL-PCR against different workloads. To simulate different workloads, we vary the number of generated transactions from 1,000 to 10,000. More transactions generally mean heavier workloads. Fig. 10 presents the throughput and success ratios of four rebalancing algorithms in the Ripple and Lightning networks. As the number of transactions increases, the overall amount of successful transactions increases and the success ratios decrease. The reduction in success ratios is inevitable because the senders and receivers are sampled from an exponential distribution. This uneven distribution of



(a) Different coefficients c_1 and c_2.



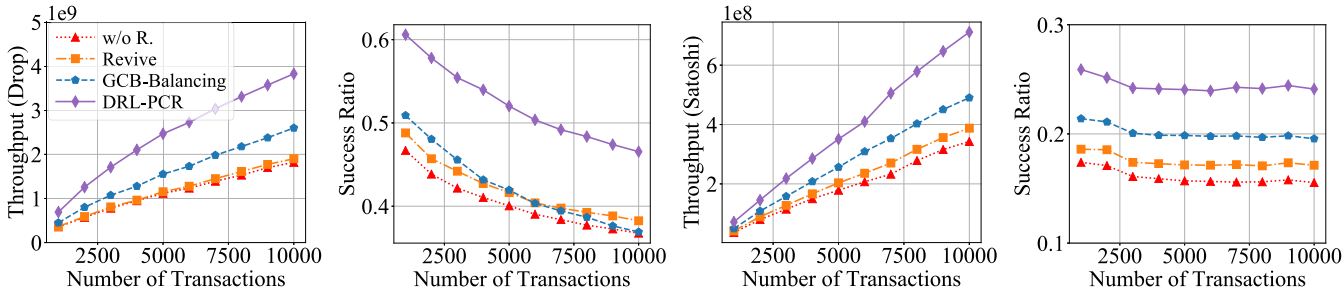
(b) Different discount factors γ.

Fig. 9. Convergence results under different parameter settings.

sender-receiver pairs exacerbates the channel imbalance as the number of transactions increases. Nevertheless, DRL-PCR still outperforms the baselines through the evaluations.

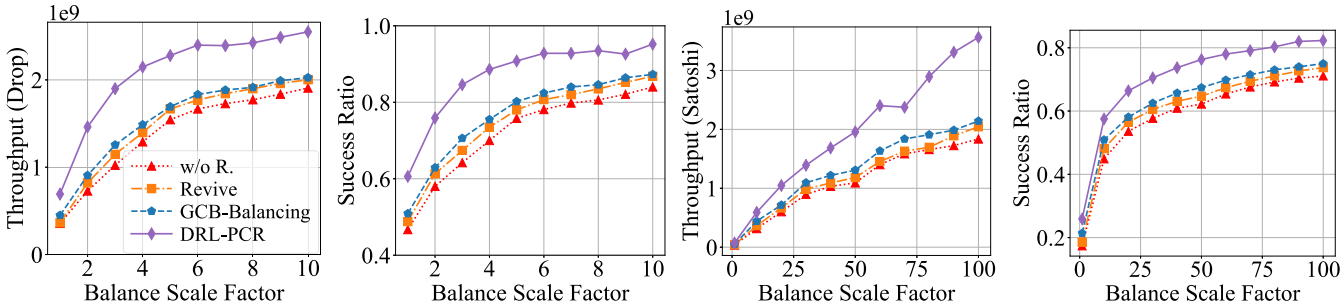
As shown in Fig. 10, the advance of DRL-PCR is most pronounced under the heaviest workloads. When the number of transactions is 10,000, the throughput of DRL-PCR in Ripple is about 3.8×10^9 Drop, which is up to 2.1x better over w/o R., 2x better over Revive, and 1.52x better over GCB-Balancing. And in Lightning, the throughput of DRL-PCR is about 7.1×10^8 Satoshi, which is up to 2x better over w/o R., 1.86x better over Revive, and 1.44x better over GCB-Balancing. The advance of DRL-PCR in throughput is because it models the rebalancing problem as a sequential decision problem and leverages the principle of DRL to maximize the cumulative rewards, ending with a more equilibrium PCN. In contrast, GCB-Balancing seeks to maximize the immediate reward of each decision step, without considering that the current best action may not bring the overall maximum rewards. As a result, the throughput and success ratios of GCB-Balancing are much smaller than those of DRL-PCR. The performance of Revive is only better than that of the w/o R. baseline due to the approximation errors in the large solution space.

3) Performance Under Different Balance Scaling Factors: As observed from the previous evaluations, the throughput and success ratios of all rebalancing algorithms have not yet reached the approximated optimal value in Section V-B1. Considering that PCN technology is still in its early development, the channel balances from real-world datasets may be limited and cannot meet the need to complete all transactions [8]. Given this, we



(a) Throughput in Ripple. (b) Success ratio in Ripple. (c) Throughput in Lightning. (d) Success ratio in Lightning.

Fig. 10. Performance results under different workloads.



(a) Throughput in Ripple. (b) Success ratio in Ripple. (c) Throughput in Lightning. (d) Success ratio in Lightning.

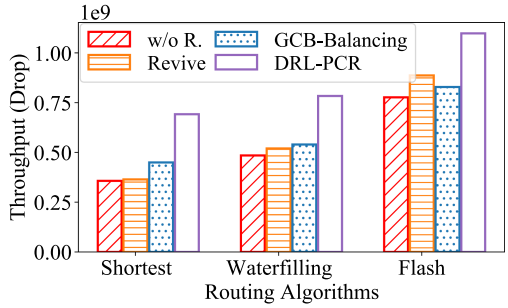
Fig. 11. Performance results under different balance scaling factors.

vary the channel balance with a scale factor to investigate the performance upper bound of the rebalancing algorithms. For Ripple, we scale the channel balance by a factor of 1 to 10. And for Lightning, the scale factor ranges from 1 to 100. The number of generated transactions in this evaluation is 1,000.

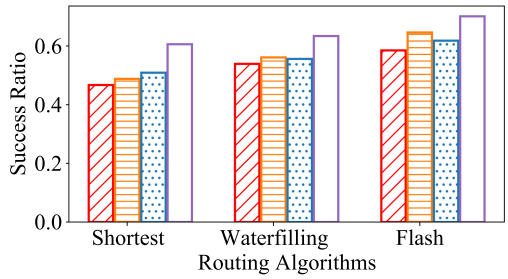
Fig. 11 presents the throughput and success ratios of the rebalancing algorithms under different balance scaling factors. With the increase of scale factor, both the throughput and success ratios increase. It is clear that with higher balances, channels are more resistant to uneven transaction demands and have a longer lifetime, thus contributing to completing more transactions. As shown in Fig. 11, the throughput and success ratios of DRL-PCR are the highest for both Ripple and Lightning networks across the evaluations. DRL-PCR shows a significant advantage in throughput, which is up to 1.25x and 1.62x higher than the baselines for the Ripple and Lightning networks, respectively. In particular, the success ratio of DRL-PCR is close to the theoretical upper bound. In the Lightning, the success ratio is around 0.95. In contrast, even with the largest balance scale factor, the success ratios of other baselines are only around 0.74 in the Ripple and around 0.87 in the Lightning. This suggests that our algorithm can better exploit the benefits of increasing channel balance to achieve higher throughput and success ratios.

4) Performance Under Different Routing Algorithms: Lastly, we analyze the impact of routing algorithms used in the PCN. As mentioned in the system model of Section III, PCN users apply routing algorithms to complete the generated transactions and the rebalancing algorithm works as a recovery mechanism for failed transactions. To validate the robustness of DRL-PCR, we run evaluations with three routing algorithms: 1) Shortest-path-first, which finds the shortest path between the sender and the receiver to send the transaction; 2) Waterfilling [9], which uses a heuristic "waterfilling" idea and sends transactions through channels with the largest balance; 3) Flash [8], which divides transactions into mice transactions and elephant transactions. The mice transactions in Flash are routed with a trial-and-error method and the elephant transactions are routed with a modified maximum flow algorithm.

Figs. 12 and 13 show the evaluation results in the Ripple and Lightning networks. We can observe that choosing a good routing algorithm can significantly improve the throughput and success ratios, where the throughput of Flash is around 2x higher than that of the shortest-path-first algorithm and is around 1.5x higher than that of Waterfilling. Nevertheless, a good rebalancing algorithm is still indispensable. Compared with the case without the rebalancing algorithm (i.e., w.o R.), DRL-PCR outperforms the baselines by more than 41% throughput in Ripple and more than 48% throughput in Lightning. The success ratio of DRL-PCR is at least 4.7% higher than the baselines in Ripple and at least 10% higher in Lightning. Remarkably, DRL-PCR consistently achieves a higher performance across

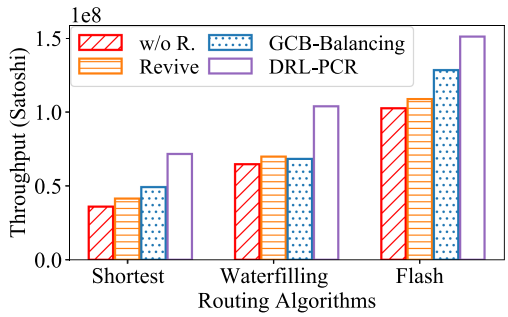


(a) Throughput in Ripple.

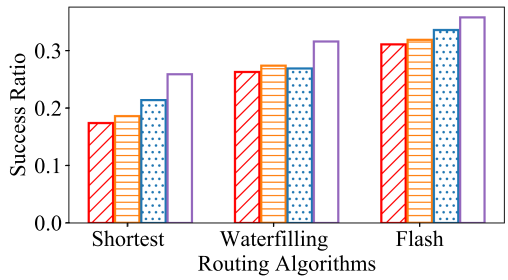


(b) Success Ratio in Ripple.

Fig. 12. Performance results in the Ripple network under different routing algorithms.



(a) Throughput in Lightning.



(b) Success Ratio in Lightning.

Fig. 13. Performance results in the Lightning network under different routing algorithms.

varied networks and routing algorithms. This demonstrates the robustness of DRL-PCR. In addition, the significant improvement in the throughput and success ratios of DRL-PCR indicates that it can further exploit the benefits of a good routing algorithm such as Flash.

VI. Future Directions

The future work can go in the following directions. First, we focus on solving the rebalancing problem and assume cooperation by all nodes in this work. Investigating scenarios where nodes may act in self-interest or even maliciously can shed light on the real-world challenges of decentralized networks. Exploring the impact of strategic behavior by individual nodes and developing mechanisms to incentivize cooperation can be an interesting area for investigation. Second, there are limitations in terms of security and privacy protection for this work. As we move forward, it is essential to explore strategies to enhance security and protect the privacy of participants. We may investigate the feasibility of incorporating Verifiable Random Functions or similar distributed mechanisms into the leader election.

Third, there is room for significant improvement in optimizing the proposed DRL-based model. By considering various factors such as the input information, network structure, and message passing processes, we can fine-tune the model for even better performance. The optimization of the DRL model makes it possible to reduce training costs and improve performance after convergence. In addition, Transformer-based architectures have gained significant attention in recent years for solving various combinatorial optimization problems. The proposed algorithm in this work is based on GNN, which is inherently designed to work with graph-structured data. This makes it a natural choice for the rebalancing problem. Nonetheless, due to the self-attention mechanisms, Transformer-based architectures excel at capturing global dependencies and long-range interactions in data. We encourage further exploration of Transformer-based models in payment channel rebalancing. In addition, PCNs are inherently dynamic, with users joining and leaving, new transaction edges forming, and existing ones closing. These dynamics can impact the performance of DRL-PCR. To tackle the evolving nature of PCNs, one common approach is to periodically retrain DRL-PCR. The frequency of retraining can vary depending on the rate of PCN changes and the desired level of performance. Future work may consider the trade-off between computational costs and the need for up-to-date policies.

VII. Conclusion

In this work, we have studied the channel rebalancing problem of PCN. To deal with the problem scale and mitigate the risk of getting trapped in a local optimum, we propose DRL-PCR, a DRL-based rebalancing algorithm. On the one hand, DRL-PCR leverages the strong approximation ability of DL to handle large problem spaces. On the other hand, DRL-PCR decomposes the original problem into a sequential decision problem and aims at global optimality with the long-term optimization of DRL. In particular, to handle the graph-structured inputs, DRL-PCR is implemented with a novel GNN-based model that efficiently extracts the sophisticated circular dependencies between paths and channels through a customized message passing process and a loop operation. Evaluation results based on two real-world PCNs show that DRL-PCR can restore the PCN to a more balanced state than state-of-the-art rebalancing algorithms. Meanwhile,

a PCN can conduct more transactions after rebalancing. DRL-PCR significantly outperforms existing algorithms in terms of transaction throughput and success ratios by up to 2.1x and 1.6x, respectively.

REFERENCES

[1] M. Shayan, C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1513–1525, Jul. 2021. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3044223

[2] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data*, G. Karypis and J. Zhang, Eds., Honolulu, HI, USA, Jun. 25–30, 2017, pp. 557–564. [Online]. Available: https://doi.org/10.1109/BigDataCongress.2017.85

[3] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., Dallas, TX, USA, 2017, pp. 439–453. [Online]. Available: https://doi.org/10.1145/3133956.3134033

[4] R. Huo, D. Ni, Q. Chen, and Y. Xia, "Greedy channel balanced routing strategy of payment channel networks," in *Proc. 4th Int. Conf. Hot Inf.-Centric Netw.*, 2021, pp. 13–18.

[5] R. Pickhardt and M. Nowostawski, "Imbalance measure and proactive channel rebalancing algorithm for the lightning network," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, Toronto, ON, Canada, May 2–6, 2020, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ICBC48266.2020.9169456

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

[7] P. Prihodko, S. N. Zhigulin, M. Sahno, A. B. Ostrovskiy, and O. Osuntokun, "Flare : An approach to routing in lightning network white paper," 2016.

[8] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proc. 15th Int. Conf. Emerg. Netw. Experiments Technol.*, A. Mohaisen and Z. Zhang, Eds., Orlando, FL, USA, 2019, pp. 370–381. [Online]. Available: https://doi.org/10.1145/3359989.3365411

[9] V. Sivaraman et al., "High throughput cryptocurrency routing in payment channel networks," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, R. Bhagwan and G. Porter, Eds., Santa Clara, CA, USA, Feb. 25–27, 2020, pp. 777–796. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/sivaraman

[10] W. Chen, X. Qiu, Z. Hong, Z. Zheng, H. Dai, and J. Zhang, "Proactive look-ahead control of transaction flows for high-throughput payment channel network," in *Proc. 13th Symp. Cloud Comput.*, A. Gavrilovska, D. Altinbüken, and C. Binnig, Eds., San Francisco, CA, USA, Nov. 07–11, 2022, pp. 429–444. [Online]. Available: https://doi.org/10.1145/3542929.3563491

[11] X. Qiu, W. Chen, B. Tang, J. Liang, H. Dai, and Z. Zheng, "A distributed and privacy-aware high-throughput transaction scheduling approach for scaling blockchain," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 4372–4386, Sep./Oct. 2023.

[12] P. Li, T. Miyazaki, and W. Zhou, "Secure balance planning of off-blockchain payment channel networks," in *Proc. 39th IEEE Conf. Comput. Commun.*, Toronto, ON, Canada, Jul. 06–09, 2020, pp. 1728–1737. [Online]. Available: https://doi.org/10.1109/INFOCOM41043.2020.9155375

[13] N. Papadis and L. Tassiulas, "Deep reinforcement learning-based rebalancing policies for profit maximization of relay nodes in payment channel networks," 2022, *arXiv:2210.07302v2*. [Online]. Available: https://doi.org/10.48550/arXiv.2210.07302

[14] The lightning network, 2019. [Online]. Available: https://lightning.network

[15] W. Tang, W. Wang, G. Fanti, and S. Oh, "Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks," in *Proc. Abstr. SIGMETRICS/Perform. Joint Int. Conf. Meas. Model. Comput. Syst.*, E. Yeh, A. Markopoulou, and Y. C. Tay, Eds., Boston, MA, USA, Jun. 08–12, 2020, pp. 81–82. [Online]. Available: https://doi.org/10.1145/3393691.3394213

[16] T. Haarnoja et al., "Soft actor-critic algorithms and applications," 2018, *arXiv: 1812.05905*. [Online]. Available: http://arxiv.org/abs/1812.05905

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv: 1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[18] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[19] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1603–1614, Jul. 2021. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3046737

[20] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in SDN," in *Proc. ACM Symp. SDN Res.*, San Jose, CA, USA, Apr. 03–04, 2019, pp. 140–151. [Online]. Available: https://doi.org/10.1145/3314148.3314357

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605

[22] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, A. Moschitti, B. Pang, and W. Daelemans, Eds., 2014, pp. 1724–1734. [Online]. Available: https://doi.org/10.3115/v1/d14--1179

[23] M. Langer, Z. He, W. Rahayu, and Y. Xue, "Distributed training of deep learning models: A taxonomic perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2802–2818, Dec. 2020. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3003307

[24] K. Zhong, Z. Yang, G. Xiao, X. Li, W. Yang, and K. Li, "An efficient parallel reinforcement learning approach to cross-layer defense mechanism in industrial control systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2979–2990, Nov. 2022. [Online]. Available: https://doi.org/10.1109/TPDS.2021.3135412

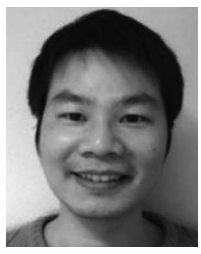
[25] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. 4th Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1506.02438

[26] S. M. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn.*, C. Sammut and A. G. Hoffmann, Eds., Morgan Kaufmann, 2002, pp. 267–274.

[27] H. Dang, T. T. A. Dinh, D. Loghin, E. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manage. Data*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds., Amsterdam, The Netherlands, 2019, pp. 123–140. [Online]. Available: https://doi.org/10.1145/3299869.3319889

[28] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, Feb. 18–21, 2018. [Online]. Available: https://arxiv.org/pdf/1709.05748.pdf

[29] K. Lange, E. Rohrer, and F. Tschorsch, "On the impact of attachment strategies for payment channel networks," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, Sydney, Australia, 2021, pp. 1–9. [Online]. Available: https://doi.org/10.1109/ICBC51069.2021.9461104



Wuhui Chen (Member, IEEE) received the bachelor's degree from Northeast University, Shenyang, China, in 2008, and the master's and PhD degrees from the University of Aizu, Aizu–Wakamatsu, Japan, in 2011 and 2014, respectively. From 2014 to 2016, he was a research fellow with the Japan Society for the Promotion of Science, Japan. From 2016 to 2017, he was a researcher with the University of Aizu. He is currently an associate professor with Sun Yat-Sen University, Guangzhou, China. His research interests include edge/cloud computing, cloud robotics, and blockchain.

Xiaoyu Qiu received the BS degree from Sun Yat-Sen University, Guangzhou, China, in 2020. He is currently working toward the MS degree with the School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China. He is proactively working on edge computing, cloud computing, cloud robotics and computation offloading, with emphasis on artificial intelligence in edge/cloud computing.

Zhongteng Cai received the BS degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2020. He is currently working toward the MS degree with Sun Yat-sen University, Guangzhou, China. His current research interests include blockchain technologies and consensus algorithm.

Bingxin Tang received the BS degree from Sun Yat-Sen University, Guangzhou, China, in 2021. He is currently working toward the MS degree with the School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China. His current research interests include the payment channel network, offchain transactions, and deep reinforcement learning.

Linlin Du is with Science and Technology on Complex Systems Simulation Laboratory, her research area is focusing on complex systems simulation, Blockchain.

Zibin Zheng (Fellow, IEEE) is currently a professor and the deputy dean with the School of Software Engineering, Sun Yat-sen University, Guangzhou, China. He authored or coauthored more than 200 international journal and conference papers, including one ESI hot paper and six ESI highly cited papers. According to Google Scholar, his papers have more than 15,000 citations. His research interests include blockchain, software engineering, and services computing. He was the BlockSys'19 and CollaborateCom16 general co-chair, SC2'19, ICIOT18 and IoV14 PC Co-Chair. He is a fellow of the IET. He was the recipient of several awards, including the Top 50 Influential Papers in Blockchain of 2018, the ACM SIGSOFT Distinguished Paper Award at ICSE2010, the Best Student Paper Award at ICWS2010.