A Distributed and Privacy-Aware High-Throughput Transaction Scheduling Approach for Scaling Blockchain

Xiaoyu Qiu, Wuhui Chen, *Member, IEEE*, Bingxin Tang, Junyuan Liang, Hong-Ning Dai, *Senior Member, IEEE*, and Zibin Zheng, *Senior Member, IEEE*

Abstract—Payment channel networks (PCNs) are considered as a prominent solution for scaling blockchain, where users can establish payment channels and complete transactions in an off-chain manner. However, it is non-trivial to schedule transactions in PCNs and most existing routing algorithms suffer from the following challenges: 1) one-shot optimization, 2) privacy-invasive channel probing, 3) vulnerability to DoS attacks. To address these challenges, we propose a privacy-aware transaction scheduling algorithm with defence against DoS attacks based on deep reinforcement learning (DRL), namely PTRD. Specifically, considering both the privacy preservation and long-term throughput into the optimization criteria, we formulate the transaction-scheduling problem as a Constrained Markov Decision Process. We then design PTRD, which extends off-the-shelf DRL algorithms to constrained optimization with an additional cost critic-network and an adaptive Lagrangian multiplier. Moreover, considering the distribution nature of PCNs, in which each user schedules transactions independently, we develop a distributed training framework to collect the knowledge learned by each agent so as to enhance learning effectiveness. With the customized network design and the distributed training framework, PTRD achieves a good balance between the optimization of the throughput and the minimization of privacy risks. Evaluations show that PTRD outperforms the state-of-the-art PCN routing algorithms by 2.7%–62.5% in terms of the long-term throughput while satisfying privacy constraints.

Index Terms—Blockchain, transaction scheduling, privacy-aware, deep reinforcement learning, distributed training

1 Introduction

In recent years, blockchain has received enhanced interests in many domains. Despite the advances in booming lots of cryptocurrencies as well as other applications, the throughput bottleneck of blockchains also restricts their further adoptions [1], [2]. For example, the transaction processing capacity of Bitcoin [3] is limited to 10 transactions per second (tps), which is far from meeting the demands of large-scale trading scenarios. The recent off-chain payment channel network (PCN) is emerging as a feasible scaling solution to address this issue. PCNs have been deployed to the prevailing cryptocurrencies, such as the Lightning Network [4] adopted in Bitcoin.

PCNs allow two nodes to establish a private channel, which acts as a two-party ledger privately maintained by both parties. Through the private channel, two nodes can settle multiple payments without resorting to on-chain operations. Take Fig. 1 as an example, in which nodes A and B need to conduct transactions with each other frequently. By contrast, if all transactions are conducted on the blockchain, it will introduce huge overhead and high latency. To process transactions in an off-chain way, nodes A and B open a channel by depositing funds as initial collateral, e.g., each of them deposits 10 tokens to the channel, as shown in Fig. 1. Subsequently, nodes A and B can execute off-chain transactions by updating the available balances in the channel. Either party can close the channel by submitting a transaction containing the latest balance messages to the blockchain. In practice, PCNs can not only be used for executing transactions between two directly-connected nodes. PCNs use a smart contract called Hash Time-Lock Contract (HTLC) to guarantee transaction atomicity and thus empowers two unconnected nodes to settle payments through a multi-hop path. Therefore, the key to settle payment in PCNs is to find a path with sufficient channel balances; this process is also called *PCN routing*.

Challenges. To make PCNs economically viable, it is necessary to design an efficient and robust routing algorithm to schedule transactions for successful executions. Routing

- *Xiaoyu Qiu, Wuhui Chen, Bingxin Tang, and Junyuan Liang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China. E-mail: {qiuxy23, tangbx, liangjy53}@mail2.sysu.edu.cn, chenwuh@mail.sysu.edu.cn.*
- *Hong-Ning Dai is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China. E-mail: hndai@ieee.org.*
- *Zibin Zheng is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China, and also with the School of Software Engineering, Sun Yat-sen University, Zhuhai, Guangdong 519082, China. E-mail: zhzibin@mail.sysu.edu.cn.*

Manuscript received 11 March 2022; revised 14 September 2022; accepted 15 October 2022. Date of publication 25 October 2022; date of current version 1 September 2023.
This work was supported in part by 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 Key-Area Research and Development Program of Shandong Province under Grant 2021CXGC010108, in part by the National Natural Science Foundation of Guangdong province under Grants 2022A1515010154, 6142006200403, and XM2021XT1084, in part by the Major Key Project of PCL under Grant PCL2021A06, and in part by Pearl River Talent Recruitment Program under Grant 2019QN01X130.
(Corresponding author: Wuhui Chen.)
Digital Object Identifier no. 10.1109/TDSC.2022.3216571

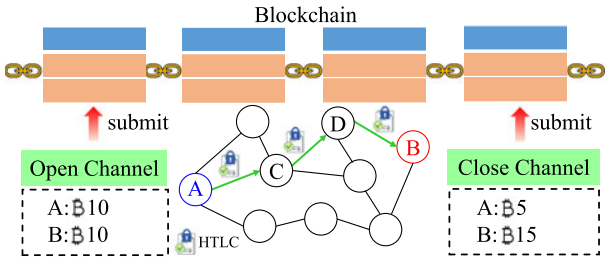


Fig. 1. An example of PCN, through which nodes A and B can make payments via the routing path A → C → D → B.

protocols in PCN essentially work quite different from conventional routing protocols in data communication networks. Rather than finding paths with the shortest length (or with the lowest load) like traditional data networks, PCN routing aims to find paths with *sufficient channel balances* to support transaction-forwarding operations. Moreover, PCN routing is highly sensitive to the dynamics of channel balances due to the consumptive nature of transaction forwarding. Despite some recent advances in PCN routing schemes, such as Flash[5], SpeedyMurmurs[7], and Flare[8], there exist three fundamental challenges that remain unstudied.

1) One-shot Optimization: Because the consumed funds (balances) are not automatically restored in PCNs unlike the auto-released bandwidth in traditional data networks, PCN routing is highly sensitive to system dynamics and a short-term routing decision may not reach an optimal solution from a long-term perspective. In this regard, despite the high instantaneous throughput, the *long-term throughput* may be low, where the long-term throughput is defined as the overall number of successful transactions within a given period. Unfortunately, most existing PCN routing algorithms only consider one-shot optimization, which only focuses on the successful settlement of current transactions and may accelerate the channel imbalance. As a result, these greedy schemes may not achieve a long-term optimal throughput even if the instantaneous throughput is achieved.

2) Privacy-invasive Channel Probing: To preserve privacy, the instantaneous channel balances are not publicly announced in PCNs. Thus, the transaction senders need to iteratively probe the available balances of candidate paths before sending transactions [9] though the highly frequent channel-probing operations are *privacy-invasive* and also violate the design intent of PCNs. The *privacy constraint* in PCN requires that the number of probed channels when a payment is processing is below a preset threshold. However, most existing work fails to consider the trade-off between the transaction throughput and the privacy risks caused by channel probing.

2) Vulnerability to DoS Attacks: A participant may attack PCNs rather than forwarding transactions. As indicated in [10], [11], [12], Denial-of-Service (DoS) attacks based on route hijacking have emerged. However, most PCN routing algorithms are memoryless. To maximize the throughput, they tend to choose the channels with higher balances, thereby making them be vulnerable to DoS attacks based on route hijacking [10], [13]. Moreover, the scale-free nature of PCNs further exacerbates the damage of DoS attacks [14], [15]. Section 2 will elaborate details of the above three challenges.

Motivation and Contributions. To address the above challenges, we present a Privacy-aware high-Throughput Routing algorithm with defence against DoS attacks based on Deep reinforcement learning (DRL), namely PTRD. DRL [16] essentially integrates both the long-term optimization principle of reinforcement learning (RL) and the strong function approximation capability of deep learning (DL). In particular, DRL directly learns from the historical experiences without human intervention by operating in an interactive manner.

However, PCN routing poses non-trivial challenges in designing DRL schemes in contrast to standard DRL tasks. In particular, there are multiple objectives in PCN routing, which need to be optimized while standard DRL tasks can usually be modeled as a maximization problem (i.e., maximizing the cumulative reward). More specifically, in our PTRD, we need to minimize the channel-probing operations in addition to maximizing the long-term throughput. This is because high-frequency balance probing is extremely privacy-invasive. Thus, our PTRD essentially needs to make a trade-off between improving the network throughput and reducing privacy risks caused by channel probing. An intuitive idea is to add a penalty term to the original objective function if there are frequent channel probing operations. This design nevertheless requires designers to carefully choose an appropriate penalty value. For instance, if the penalty value is too small, the DRL agent may deliberately violate constraints to achieve a higher throughput; while DRL may fail to learn anything [17] if the penalty is too large. This critical requirement also poses a challenge in designing an effective DRL scheme.

Our PTRD has well addressed these challenges. To consider both the PCN throughput (as the reward) and privacy risks (as the constraint) together, we start by formulating the routing problem as a Constrained Markov Decision Process (CMDP). The major difference between CMDP and traditional MDP is that additional constraints are added to the objective function. This significantly increases the difficulty of finding the optimal policy. Inspired by [17], we then design an additional network module called the *cost critic network* to bound the expected number of probed channels within a predefined privacy constraint. In addition, we convert the constrained optimization problem to an unconstrained equivalent problem by introducing a *parameterized adaptive Lagrangian multiplier*. In particular, considering that PCN is a distributed system where each user routes transactions independently, we develop a distributed training framework to collect the knowledge from each distributed agent to boost learning.

Table 1 compares our PTRD with three state-of-the-art PCN routing algorithms and naive DRL approaches (i.e., PPO-0.01, PPO-1, and PPO-100). The evaluations are conducted over two real-world PCNs, i.e., Ripple [18] and Lightning [4]. After integrating a long-term optimization goal with both a privacy-aware constraint and DoS protection, PTRD outperforms the state-of-the-art PCN routing algorithms. For example, PTRD improves the *long-term throughput* of existing PCN routing schemes by 2.7%–62.5% while keeping the *privacy violation rate* near 0, where the

TABLE 1
Comparison of PTRD With State-of-the-Art PCN Routing Algorithms

Routing Algorithms	Normalized Throughput		Privacy Violation Rate		Long-term	Privacy-aware	DoS Attack	Channel Dynamics
	Ripple	Lightning	Ripple	Lightning				
Flash [5]	43.6%	12.1%	34.1%	86.4%	✘	✘	✘	✓
Waterfilling [6]	27.9%	61.5%	42%	24.4%	✘	✘	✘	✓
SpeedyMurmurs [7]	37.8%	71.9%	0%	0%	✘	✓	✓	✘
PPO-0.01	67.1%	76.6%	50.1%	61.9%	✓	✓	✓	✓
PPO-1	63.4%	74.3%	3.3%	6.1%	✓	✓	✓	✓
PPO-100	59.7%	70.9%	0.06%	0.05%	✓	✓	✓	✓
PTRD	**63**%	**74.6**%	**0.2**%	**0.07**%	✓	✓	✓	✓

The bold values to emphasize the experimental results of our proposed algorithm, PTRD.

long-term throughput is the ratio of the number of completed payments to the total number of payment demands and the privacy violation rate is the ratio of the number of payments that violate the privacy constraint (during routing) to the total number of payment demands (details to be given in Section 5).

In summary, the main contributions are as follows:

1) We present PTRD, a novel privacy-aware high-throughput routing algorithm to schedule transactions based on DRL. To the best of our knowledge, *this is the first PCN routing scheme considering both the throughput and privacy constraints*. In addition, the learning ability of DRL empowers PTRD to detect DoS attacks.
2) We introduce an additional cost critic network and a novel training method based on the Lagrangian-multiplier technique to PTRD so as to achieve a good trade-off between the transaction throughput and privacy constraints.
3) We conduct extensive experiments over the traces of two real-world PCNs to validate the performance of PTRD. Compared with state-of-the-art PCN routing algorithms and standard DRL, our PTRD improves the transaction throughput by 2.7%–62.5% while keeping the privacy violation rate near 0.

The rest of this paper is organized as follows. Section 2 first gives the motivation of this work. Section 3 then presents the system model. Section 4 formulates the routing problem as a CMDP and presents the design of PTRD. Evaluation results are shown in Section 5, followed by related work given in Section 6. The paper is finally concluded in Section 7. The main notations are summarized in Table 2.

2 Preliminaries and Motivation

2.1 Payment Channel Network

In a nutshell, PCN allows participants to execute transactions in an off-chain manner by establishing payment channels. Suppose nodes A and B need to conduct transactions with each other frequently. If all transactions are conducted on the blockchain, it will introduce huge overhead and high latency. To process transactions in an off-chain way, nodes A and B can open a channel by depositing funds as initial channel balances. Subsequently, they can conduct off-chain transactions by updating the available balances in the channel. Either party can close the channel by submitting a transaction containing the latest balance messages. This transaction is finally submitted to the blockchain.

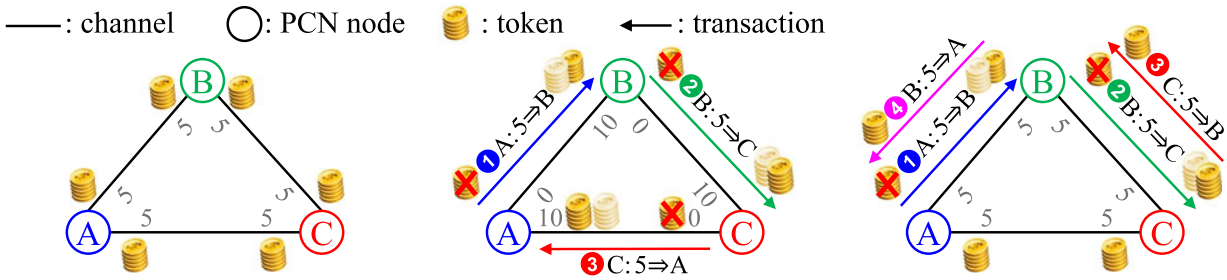
More importantly, PCN supports multi-hop routing. In particular, if nodes A and B are not directly connected, they can still settle transactions along a path consisting of multiple payment channels, e.g., A → C → D → B. For the transactions to be successful delivered, the channels along the path should have sufficient funds, i.e., the available balances of channels A → C, C → D, and D → B should be larger than the amount of the transaction to be sent. Therefore, routing algorithms are the key components of PCNs. Despite advances made by emerging PCN routing protocols, such as Flash [5], SpeedyMurmurs[7], and Flare[8], they only consider either the topological reachability or the instantaneous throughput. As a result, they suffer from the following challenges.

2.2 Challenges and Motivations

Challenge 1: how to achieve a long-term optimization on the throughput of PCN when routing transactions? A critical issue in the PCN is how to route transactions from the sender to the receiver. At the first glance, PCNs are similar to traditional communication networks. However, unlike

TABLE 2
List of Notation Definitions

Notation	Definition
$\mathcal{V}, \mathcal{E}$	Set of PCN nodes and channels
C^{uv}	Initial capacity of channel (u, v)
b^{uv}	Channel balance of channel (u, v)
$\mathcal{D} = \{d_1, d_2, \ldots, d_T\}$	Payment demand sequence
T	Number of payment demand
$\mathcal{P}_C, \mathcal{P}_S$	Candidate path set and selected path set
$I(d_t)$	Indicator for payment success
$Pr(d_t)$	Number of probed channels for routing d_t
ρ	Maximum number of probed channels
s_t, a_t, r_t, c_t	State, action, reward, cost
γ	Discount factor
θ	Parameters of the actor network
ω	Parameters of the reward-critic network
δ	Parameters of the cost-critic network
π_θ	Actor-network
V_ω^c	Reward critic-network
V_δ^c	Cost critic-network
κ	Lagrangian-multiplier
d	Hyper-parameter for privacy constraint
$\hat{A}_t$	Advantage function



(a) Initial balance distribution of the PCN. (b) One-shot optimization. (c) Long-term optimization.

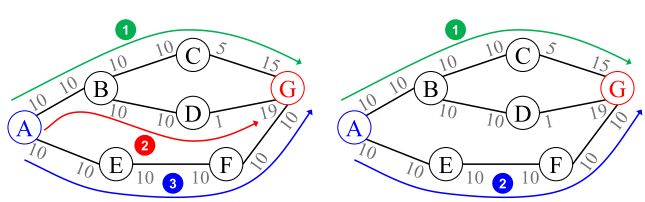
Fig. 2. An illustrative example of the problem with one-shot optimization, where the lines with arrows represent the transaction flows. In the one-shot optimization scheme, the channel balances of A → B, B → C and C → A are exhausted after sending transactions. By contrast, in the long-term optimization scheme, PCN remains perfectly balanced after sending transactions.

traditional communication networks, PCN routing is more sensitive to system dynamics. This is because forwarding transactions is a consumptive behavior and will change the distribution of channel balances, thereby affecting the settlement of future transactions. By contrast, the channel capacity can be immediately restored in traditional communication networks. Existing PCN routing algorithms nevertheless only consider *one-shot optimization* and the *instantaneous* throughput while failing to consider the long-term optimization of the throughput. The ignorance of the long-term optimization of the throughput can fasten the channel imbalance, thereby leading to the decreased throughput from a long-term perspective. In other words, despite the high instantaneous throughput, the overall throughput in the long run may be still low. Fig. 2 illustrates the problems caused by one-shot optimization. Assume that node A sends 5 tokens to node B, which then sends 5 tokens to node C. Node C next sends 5 tokens to node A. As shown in Fig. 2b, the traditional one-shot optimization tends to let each node choose the shortest path, consequently depleting channel capacity in the directions of $A \xrightarrow{①} B$, $B \xrightarrow{②} C$, and $C \xrightarrow{③} A$.

By contrast, Fig. 2c shows that the long-term strategy can balance each channel along the route $A \xrightarrow{①} B \xrightarrow{②} C \xrightarrow{③} B \xrightarrow{④} A$ while successfully executing transactions. In particular, PCN remains perfectly balanced after sending transactions. To achieve such a long-term throughput optimization, it requires the PCN to strategically route transactions from a long-term perspective, which is nevertheless non-trivial because the future payment demands are generally unavailable beforehand [19]. This motivates us to design PTRD based on DRL, which formulates the instantaneous throughput as reward and focuses on maximizing the cumulative rewards, thereby achieving high long-term throughput. Furthermore, DRL directly learns the system dynamics via the historical interactions, thereby endowing PTRD with the ability to predict future payment demands.

Challenge 2: how to make the trade-off between the transaction throughput and the privacy risks caused by channel probing? Cryptocurrency users have a practical concern on their privacy preservation. For this concern, PCNs use the source routing protocol for transaction sending, in which the sender is responsible for finding available paths having sufficient balances to reach the receiver. In addition, the instantaneous channel balances are not publicly announced in PCNs for privacy preservation. As a result, to determine the paths to route transactions, the senders need to iteratively probe the available channel balances of candidate paths. However, high-frequency channel probing is extremely privacy-invasive and also violates the design intent of PCNs though this issue has not been addressed in most existing studies. For instance, some routing algorithms, such as Flash [5] use breadth-first-search (BFS) to enumerate topologically-reachable paths and then to probe the channel balances along the paths. Fig. 3a shows a possible probing choice of these schemes, where node A wants to send 10 tokens to node G. It can be observed that the BFS strategy needs to probe eight channels to fulfill the transaction. This is because Path ① probes three channels, Path ② probes two channels (the common channel A-B is probed by Path ①), and Path ③ probes three channels. By contrast, Fig. 3b shows a better solution, in which only six channels are needed (i.e., only Path ① and Path ③ are needed). Therefore, there exists a trade-off between improving the transaction throughput and reducing privacy risks. In a privacy-preserving PCN, a privacy-aware routing algorithm should strive for the high throughput while maintaining the minimal need for channel probing.

Challenge 3: How to Defend Against DoS Attacks? Subject to complicated incentives, a participant may attack PCN instead of forwarding transactions to maximize its profits. In particular, a novel DoS attack based on route hijacking is emerging, where malicious nodes may seduce transaction senders to choose them as routing relays and compromise PCN by dropping transactions [13]. However, most dynamic routing schemes are vulnerable to these emerging DoS attacks since they highly rely on the probed informa-



(a) BFS probes eight channels. (b) A better solution probes six channels

Fig. 3. An illustrative example of the problem caused by state-of-the-art BFS-based routing schemes, where the lines with arrows and serial numbers indicate the sequential probing paths.

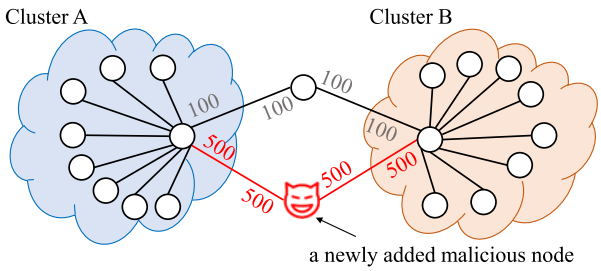


Fig. 4. An example illustrates DoS attacks, in which a single adversary can hijack routes of multiple nodes by creating channels with large balances. For clarity, the balances of other channels are omitted.

tion and tend to choose channels with high balances. Even worse, since PCN exhibits similar properties to the scale-free network [20], [21], [22], in which most transactions are relayed by a few nodes with high betweenness centrality [13], it further exacerbates the harm of DoS attacks. Consider Fig. 4 as an example, where cluster A and cluster B are connected through channels with a balance of 100 tokens. Then, a newly added malicious node deposits large funds and establishes channels with two nodes that have a high betweenness centrality. As a result, the newly added malicious node is more likely to be selected as routing relay.

In essence, this malicious attack occurs because existing routing algorithms are memoryless, where transactions are scheduled according to the current channel states. To maximize the throughput, most PCN routing schemes have a higher preference to a node with a higher balance at each channel for a higher successful rate of the transactions, thus the malicious node will have a higher chance to be selected than the normal nodes. Consequently, a single adversary can hijack routes of multiple nodes and even paralyse the entire PCN. Moreover, because of the implementation of multi-hop payments with HTLC, senders are unaware which node drops the transactions since they only have the knowledge of the failed transactions due to the timeout. This effect further increases the routing difficulty.

These challenges motivate us to design PTRD, a privacy-aware high-throughput routing algorithm with defence against DoS attacks. In particular, PTRD is based on DRL yet customized for PCN routing. We first present the system models in Section 3 and then give details of PTRD in Section 4.

3 Model Design

3.1 Network Model

In this work, we consider a bidirectional-channel PCN, which can be modeled as an undirected graph denoted by $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the vertex set of PCN nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set of opened payment channels. The number of nodes and the number of channels are denoted by $V = |\mathcal{V}|$ and $E = |\mathcal{E}|$, respectively. Due to the bidirectional channels, each edge $(u, v) \in \mathcal{E}$ has two weights representing the instantaneous balances in two directions. We denote the channel balance from u to v and that from v to u by b^{uv} and b^{vu}, respectively. In addition, for each channel $(u, v) \in \mathcal{E}$, we denote the initial capacity (i.e., the amount of cryptocurrencies when the channel (u, v) is established) by C^{uv}, where $C^{uv} = b^{uv} + b^{vu}$. If there is no direct channel connected two nodes, i.e., $\forall v_1, v_2 \in \mathcal{V}, (v_1, v_2) \notin \mathcal{E}$, the capacity $C^{v_1 v_2}$ equals to 0. The channel capacities and instantaneous balances of the whole PCN can be denoted by two matrices $\mathcal{C}$ and $\mathcal{B}$, respectively, each with size $V \times V$. For each node $u \in \mathcal{V}$, $N(u) = \{v | (u, v) \in \mathcal{E}\}$ is a set containing all the neighboring nodes of u and $\mathcal{A}(u)$ is a set containing all channels directly connected to node u. To preserve privacy, the instantaneous channel balances are not publicly announced in PCN, and each node only knows the balances of its connected channels.

3.2 Routing Model

To complete a payment request, the sender initiates one or more transactions. PCN conducts transactions with a source routing protocol, where the transaction sender is responsible for finding a path from the source to the destination and determining the amount of transactions. To maximize the long-term throughput, we propose a discrete-time model to describe the payment demands in PCN. In this model, a sequence of payment demands $\mathcal{D} = \{d_1, d_2, \ldots, d_T\}$ are generated to be conducted. Each payment demand d_t ($d_t \in \mathcal{D}$) is associated with four attributes: i) source $S(d_t)$, ii) destination $D(d_t)$, iii) amount of payments denoted by $A(d_t)$, and iv) timestamp $T(d_t)$. We assume that each payment is sequentially processed. When a payment d_t arrives, the source node needs to find all the available paths from $S(d_t)$ to $D(d_t)$. An available path p can be denoted by a list of edges: $p = [(u_1, u_2), (u_2, u_3), \ldots, (u_{m-1}, u_m)] \in \mathcal{E}^*$, where $u_1 = S(d_t)$ and $u_m = D(d_t)$. If the transaction is forwarded along path p, the channel balances are updated as: $\forall (u, v) \in p$, $b^{uv} = b^{uv} - A(d_t)$ and $b^{vu} = b^{vu} + A(d_t)$. For the transaction to be successfully delivered, the balances of channels along path p should be no less than the transaction amount $A(d_t)$. The consideration of balances along the path differentiates PCNs from traditional communication networks.

For high throughput, we adopt the same multi-path transport protocol as in [6], [23], which allows transactions to be split and sent across different paths. Since the instantaneous balances are not publicly announced for preserving privacy, a transaction sender needs to iteratively probe and find possible paths with sufficient balances. For each incoming transaction, the sender first generates a candidate path set $\mathcal{P}_C$ based on the topology. In this work, the candidate path set is generated by iteratively invoking the Dijkstra algorithm. In principle, the candidate path set should include the shortest paths and try to avoid selecting the same channel multiple times. Therefore, we first generate a weighted graph with the same topology as the PCN. The weight of each edge is 1. Then, the Dijkstra algorithm is iteratively invoked to find the path with the lowest sum of weights. For each path that is found, we increase the weight of each edge along the path by 10. This process iterates until the maximum number of candidate paths is reached. In practice, this process can be completed in advance since the generation of the candidate path set is purely topology dependent.

Given $\mathcal{P}_C$, the sender iteratively probes paths in path set $\mathcal{P}_C$ to get real balances until the payment demand is met or all paths are traversed. If such a solution exists, the transaction will be partitioned into multiple independent

transaction-units and be forwarded to the selected path set denoted by $\mathcal{P}_S$. Given $\mathcal{P}_S$ and the instantaneous balances along these paths, the amount of transaction-units sent to each path can be easily calculated with standard max-flow algorithms, such as the famous Edmonds-Karp algorithm.

3.3 Attack Model

Subject to complicated incentives, a participant may attack PCN instead of forwarding transactions to maximize its profits. In the attack model, we consider the route hijacking-based Dos attack, where malicious nodes may seduce transaction senders to choose them as routing relays and compromise PCN by dropping transactions. To this end, they may strategically establish a set of channels, which puts them in a topologically important location of the PCN graph model $G(\mathcal{V}, \mathcal{E})$. The sender does not know which nodes are malicious during routing. If there are malicious nodes along the select sending path set $\mathcal{P}_S$, the malicious nodes can launch DoS attack, resulting in transaction failure.

3.4 Goals

Given the PCN topology and payment demands, the sender needs to select a set of targeted paths from the candidate path set to route transactions. A well-designed routing algorithm should achieve the following goals:

1) *High long-term throughput.* To achieve economic feasibility and fulfill the ever-increasing payment demands, the aggregated throughput should be as high as possible in the long run. This goal also pushes senders to route transactions through paths with avoidance of malicious nodes (to defend against DoS attacks).
2) *Privacy-awareness.* Highly-frequent channel probing violates the inherent privacy-preserving design of PCNs. To address this issue, transaction-routing schemes should be carried out on the premise of meeting the privacy-protection requirements. The privacy constraint in PCN requires that the number of probed channels when a payment is processing is below a preset threshold.

In general, the design of PCN routing needs to strike a trade-off between the transaction throughput and privacy preservation. To this end, we develop an objective function subject to a privacy-preservation constraint

$$\begin{aligned} \max \quad & \sum_{t=1}^{T} I(d_t), \\ \text{subject to} \quad & \sum_{t=1}^{T} [\Pr(d_t) - \rho] \leq 0, \end{aligned} \tag{1}$$

where $I(d_t)$ is the indicator. In particular, $I(d_t) = 1$ if the receiver successfully receives the payment; $I(d_t) = 0$, otherwise. The number of probed channels is denoted by $\Pr(d_t)$ when routing d_t. The hyper-parameter ρ is used to regulate the routing behavior of PCN nodes, which represents the maximum number of probed channels that can be tolerated in the PCN. Importantly, equation (1) characterizes the system dynamic with a long-term objective, which aims at maximizing the long-term throughput over the period $\{1, 2, \ldots, T\}$. Directly solving equation (1) is impractical because $I(d_t)$ is unknown until the payments are conducted.

4 Algorithm Design

In this section, we present the detailed design of PTRD. We first formulate the constrained optimization problem as a CMDP. Next, to solve the CMDP, we present the network architecture of PTRD, which extends off-the-shelf DRL algorithms with an additional cost critic-network and an adaptive parameterized Lagrangian multiplier. In addition, we present a training process customized for this network architecture. Finally, considering that PCN is a distributed system and each user routes transactions independently, we develop a distributed training framework for training each distributed agent and accumulating the learned knowledge.

4.1 CMDP Formulation

As shown in equation (1), the problem of maximizing long-term throughput while meeting privacy constraints is a stochastic constrained optimization problem. This problem is typically intractable since future payment demands are unavailable beforehand. Fortunately, recent advances in DRL offer a promising solution to learn system dynamic and predict future payment demands. In particular, DRL is built without any prior knowledge of the system model. Inspired by this idea, we adopt DRL to set up an intelligent agent to interact with the PCN (i.e., the environment). To apply DRL, the interactions between DRL agents and the PCN is modelled as a sequential decision-making process. It is straightforward to divide the time period into multiple epochs according to the arrival of new payment demands. During each epoch t, the DRL agent deployed at the source node first observes the PCN state s_t, then makes a routing decision as action a_t, and next receives a reward r_t measuring the performance of the former action. Different from traditional DRL tasks, a cost value c_t is also generated to measure the extent, to which the former action violates the constraints. After the PCN executing the routing decision, it transfers to the next state s_{t+1} and then a new epoch starts. This process continues until a preset condition is met, such as no new payment demands are generated or the predefined maximum epoch T is reached.

Since PCN routing is a constrained optimization problem, we then formulate the problem as a CMDP with an additional cost function in contrast to the MDP model used in standard DRL tasks. Take node $u(u \in \mathcal{V})$ as an example, the corresponding CMDP components in the context of PCN routing are described as follows:

1) *State:* The state s_t at epoch t is the information observed by node u during routing. The state includes the PCN topology G, initial channel capacities $\mathcal{C}$, incoming payment demand d_t, candidate path set $\mathcal{P}_C$, the instantaneous balances of the channels connected to u, and the channels probed in the last epoch. Note that the instantaneous balances of other channels are unknown before probing.
2) *Action:* As mentioned in the routing model of Section 3.2, the key to sending transactions is to select a

set of paths $\mathcal{P}_S$ from the candidate path set $\mathcal{P}_C$ to probe their instantaneous balances. Given a candidate path set of size N, i.e., $\mathcal{P}_C = \{p_1, p_2, \ldots, p_N\}$, the action can be encoded as a vector of size N, i.e., $a_t = \{a_t^1, a_t^2, \ldots, a_t^N\}$, where $a_t^i \in [-1, 1]$. We have $a_t^i \geq 0$ if the sender u probes the balance of path p_i; otherwise, $a_t^i < 0$.

3) *Reward:* After probing the channel balances according to action a_t, the amount of transaction-units sent along each path can be easily solved using the maximum flow algorithm [5]. Then, depending on whether the receiver receives all transaction-units, a numerical reward $r(s_t, a_t)$ is calculated to measure the action performance. Since we focus on maximizing the throughput, $r(s_t, a_t)$ equals to the term $I(d_t)$ in equation (1).
4) *Cost:* Similar to the reward, the cost $c(s_t, a_t)$ is a numerical value given by executing an action. More importantly, the cost is used to describe the extent (or magnitude), to which an action violates the constraints. The cost $c(s_t, a_t)$ equals to the term $\Pr(d_t) - \rho$ in equation (1).
5) *State Transition Probabilities:* At the end of epoch t, the PCN (i.e., the environment) updates its channels and proceeds to a new state s_{t+1} according to the state transition probabilities. In general, the transition probabilities are unavailable in real-world PCNs. This is because the future payment demands are not explicitly given and the channel updates of the whole PCN are hidden due to privacy concerns.

Based on the above formulation, DRL solves equation (1) by learning a mapping (referred to as the policy) from each state to the optimal action. To be consistent with existing DRL literature, we transform the objective function in equation (1) to a general form in DRL settings. In particular, we have

$$\begin{aligned} \arg\max_{\pi_\theta} \quad & \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t)\right], \\ \text{subject to} \quad & \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t)\right] \leq d, \end{aligned} \tag{2}$$

where T is the number of epochs, $\gamma \in (0, 1]$ is the discount factor that measures the importance of future states, and π_θ is the parameterized policy, which is typically implemented by a deep neural network (DNN). The subscript θ denotes the parameters of DNN. The term $\mathbb{E}_{\pi_\theta}$ with subscript π_θ means that the state-action distribution follows the policy π_θ. Meanwhile, d is a hyper-parameter for the privacy constraint. For instance, we can set $d \leq 0$ to prevent the routing policy from violating the constraints, or we can set $d > 0$ to allow the policy to violate the constraint (only a few times).

4.2 Network Architecture

Compared with unconstrained MDPs in standard DRL tasks, it is more challenging to solve the optimal policy in the CMDPs of equation (2) [24]. A simple idea is to add a penalty term Θ to the original reward value if the number of probed channels reaches the preset threshold, i.e.,

$$r'_t = I(d_t) - \Theta. \tag{3}$$

Then, we can directly maximize this new reward r'_t. This approach is common in many multi-objective problems.

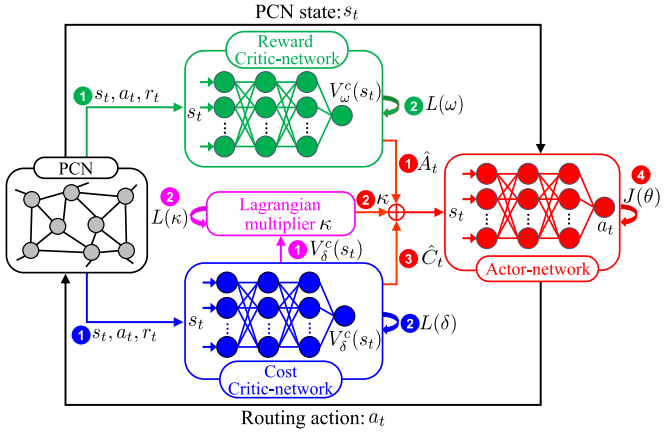


Fig. 5. Network architecture of PTRD. Black lines depict how PTRD routes transactions and other colored lines depict how PTRD is trained.

However, this setting will require designers to carefully choose an appropriate penalty value. For instance, if the penalty hyper-parameter Θ is too small, the DRL agent may deliberately violate constraints to achieve higher throughput. On the other hand, if Θ is too large, DRL may fail to learn anything[17], [25].

To solve the above problem, it is natural to consider introducing an adaptive penalty factor. Inspired by this idea and the Lagrangian-based method in [17], we propose PTRD, which extends a state-of-the-art DRL algorithm called Proximal Policy Optimization (PPO) [26] based on adaptive Lagrangian-multiplier methods. In general, Lagrangian-multiplier methods solve the constrained optimization problems by introducing an additional Lagrange-multiplier κ [27]. In line with this idea, we first convert equation (2) to an equivalent problem without constraints as follows:

$$\max_{\pi_\theta} \min_{\kappa \geq 0} \mathcal{L}(\pi_\theta, \kappa) = f(\pi_\theta) - \kappa g(\pi_\theta), \tag{4}$$

where

$$f(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t)\right],$$

and

$$g(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t)\right] - d.$$

To achieve adaptive control, the Lagrange-multiplier κ is implemented as learnable network parameters in PTRD. In this case, equation (4) can be alternatively solved by performing gradient ascent on θ and performing gradient descent on κ. Specifically, the gradient ascent applied to θ is to find the optimal policy π_θ under the given Lagrange-multiplier κ. Meanwhile, the gradient descent applied to κ is to enforce the constraint adaptively. If the policy violates the constraint frequently, taking the gradient descent on κ will make PTRD pay more attention to the constraints. Otherwise, PTRD will pay more attention on maximizing the throughput.

As shown in Fig. 5, our PTRD is based on the actor-critic architecture. This is because the action space is continuous and the critic enables efficient learning by providing more

useful feedback to the actor. PTRD contains four core neural networks:

1) *Actor-network* (aka the policy) π_θ determines the action to be taken. During each agent-environment interaction, the actor-network takes the observed PCN state s_t as the input and outputs the routing action a_t, as depicted by the black lines in Fig. 5. The goal of PTRD is to update π_θ toward the optimal policy.
2) *Reward critic-network* $V_\omega^r(\cdot)$ maps each state to the estimated long-term reward following the actor-network π_θ. Since the state transition probabilities are unavailable in real-world PCNs, the reward critic-network works as an approximation of the long-term reward function. It is used to evaluate the performance of the actor-network.
3) *Cost critic-network* $V_\delta^c(\cdot)$ maps each state to the estimated long-term cost following the actor-network π_θ. Similar to the reward critic-network, the cost critic-network is used to approximate the long-term cost function and to regularize the behaviors of the actor-network.
4) *Lagrangian multiplier* κ is implemented as a small neural network and can be adaptively updated during the training process of PTRD. For instance, if the actor-network violates the privacy constraint, i.e., $g(\pi_\theta) > 0$, we can enforce the constraint by increasing κ.

4.3 Training Process

We next present the training process of our PTRD. Given the above network design, the primary problem is then converted in how to train PTRD to solve equation (4). PTRD training proceeds in episodes and uses the collected experiences to optimize the policy. Specifically, each episode is composed of multiple epochs. During each epoch t, PTRD first observes the PCN state s_t, makes a routing action a_t, and then receives a reward $r_t = r(s_t, a_t)$ as well as a cost $c_t = c(s_t, a_t)$. After performing the action, PCN updates its channels and proceeds to a new state s_t, which ends the epoch. Hence, an experience collected from epoch t can be denoted as a tuple $(s_t, a_t, r_t, c_t, s_{t+1})$. This agent-environment interaction continues until reaching a terminal state that ends the episode. We next describe the training process in detail.

First, the reward critic-network $V_\omega^r(\cdot)$ approximates the long-term reward function with the network parameters ω. Accordingly, we update parameters ω by iteratively minimizing the Mean-Squared Error (MSE) between real long-term rewards and the estimated values. Let $\mathcal{G} = (s_1, a_1, r_1, c_1, s_2, \ldots, s_T, a_T, r_T, c_T)$ denote the sequence of experiences collected from the whole episode. Mathematically, the loss function for the reward critic-network can be represented as:

$$L(\omega) = \mathbb{E}_{\mathcal{G}}\left[\sum_{i=0}^{T-t} \gamma^i r_{t+i} - V_\omega^r(s_t)\right]^2, \tag{5}$$

where $\mathbb{E}_{\mathcal{G}}$ means the expectation with respect to the experience sequence $\mathcal{G}$ and γ is the discount factor that measures the importance of future states. Similarly, the cost critic-network $V_\delta^c(\cdot)$ is used to approximate the long-term cost function with the network parameters δ. Therefore, the parameters δ are updated by minimizing the MSE between real long-term costs and the estimated values, where the loss function is defined as follows:

$$L(\delta) = \mathbb{E}_{\mathcal{G}}\left[\sum_{i=0}^{T-t} \gamma^i c_{t+i} - V_\delta^c(s_t)\right]^2. \tag{6}$$

The updates of reward critic-network and cost critic-network are illustrated in the green lines and the blue lines in Fig. 5, respectively.

Then, the parameterized actor-network π_θ is updated by maximizing a clipped objective function similar to the PPO algorithm [26]. Specifically, in PPO, the objective function uses a specially designed surrogate objective to enforce constructive policy updates, i.e.,

$$J(\theta) = \mathbb{E}_{\mathcal{G}}\Big[\min(p_t(\theta)\hat{A}_t, p_t^{\text{clip}}(\theta)\hat{A}_t)\Big], \tag{7}$$

where $p_t(\theta)$ denotes the probability ratio between the behavior policy for collecting experiences and the policy to be updated. PPO denotes the behavior policy by π_{old}. Thus, the probability ratio $p_t(\theta)$ can be calculated as

$$p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}, \tag{8}$$

where $\pi_\theta(a_t|s_t)$ denotes the probability of taking action a_t in state s_t following the policy π_θ. Similarly, $\pi_{\text{old}}(a_t|s_t)$ denotes the action selection probability following the behavior policy π_{old}. The surrogate objective in equation (7) uses $p_t(\theta)$ to compensate the mismatch between the distribution of the training data collected by the behavior policy π_{old} and the true data distribution for the policy to be updated. This process can be explained as the application of importance sampling [28].

In addition, instead of directly using the reward function, PTRD uses a generalized advantage function $\hat{A}_t$ to reduce the variance during the gradient propagation, thus increasing the training stability. This is a common practice in deep reinforcement learning. The generalized advantage function $\hat{A}_t$ can be considered as a specific version of the long-term reward function with a lower variance by deducing a baseline. A popularized baseline to achieve the goal is the accumulated rewards for K steps forward [29]. Accordingly, the advantage function can be expressed as

$$\hat{A}_t = \sum_{j=0}^{K} \gamma^j \big[r_t + \gamma V_\omega^r(s_{t+1}) - V_\omega^r(s_t)\big], \tag{9}$$

where K is the number of steps that we look forward. In general, K is fixed and is much smaller than the length of the episode. The term r_t is the immediate reward and s_{t+1} is the next state of s_t. Intuitively, the advantage function measures the advantage of the overall rewards for K steps forward over the average case. Therefore, if a policy has a higher $\hat{A}_t$, it is more likely to select an action that brings a better-than-average reward.

Note that without restricting the ratio $p_t(\theta)$, directly maximizing $p_t(\theta)\hat{A}_t$ may lead to a drastic parameter update and

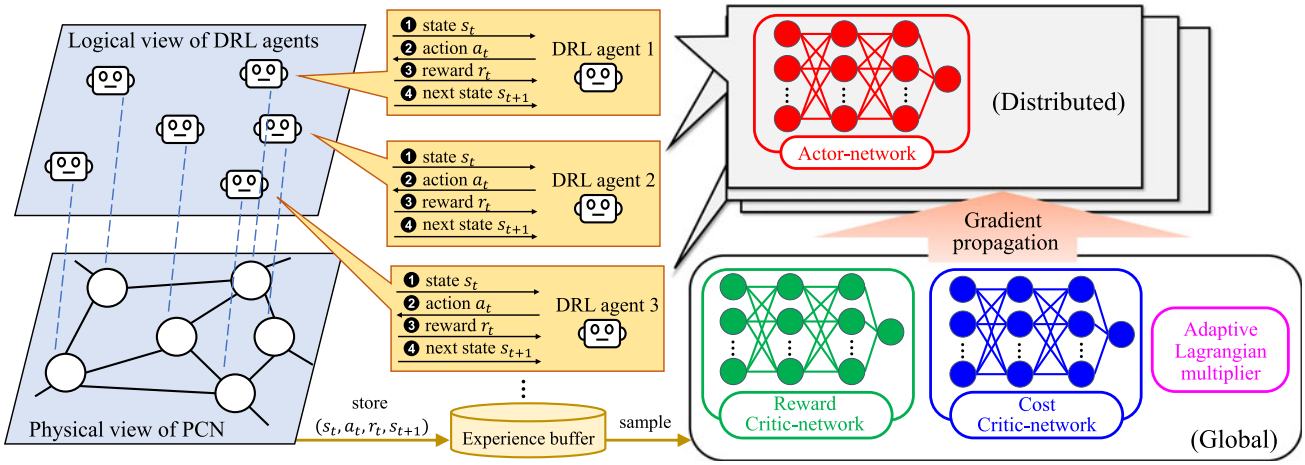


Fig. 6. Overview of the distributed training framework of PTRD.

instability since the probability ratio $p_t(\theta)$ may be extremely large. For instance, if the action selection probability $\pi_{\text{old}}(a_t|s_t)$ is a small value, it may lead to a large $p_t(\theta)$ since $\pi_{\text{old}}(a_t|s_t)$ is in the denominator. To address this issue, the objective function equation (7) imposes a constraint to enforce the ratio to stay within a small range around 1. This clipped ratio denoted by $p_t^{\text{clip}}(\theta)$ is defined as follows:

$$p_t^{\text{clip}}(\theta) = \text{clip}(p_t(\theta), 1-\epsilon, 1+\epsilon), \tag{10}$$

where ϵ is a hyper-parameter to move the ratio to interval $[1-\epsilon, 1+\epsilon]$. Finally, equation (7) selects the smaller value between $p_t(\theta)\hat{A}_t$ and $p_t^{\text{clip}}(\theta)\hat{A}_t$ as the surrogate objective and then maximize it, thereby leading to a bounded objective and a stable training process.

Note that the objective function of PPO, given in equation (7), only considers the rewards. To extend PPO from standard DRL tasks to CMDP, PTRD augments equation (7) with an adaptive cost term. Thus, the objective function can be rewritten as

$$J(\theta) = \mathbb{E}_{\mathcal{G}}\Big[\min(p_t(\theta)(\hat{A}_t - \kappa(\hat{C}_t - d)), \\ p_t^{\text{clip}}(\theta)(\hat{A}_t - \kappa(\hat{C}_t - d)))\Big], \tag{11}$$

where κ is the adaptive Lagrangian multiplier and $\hat{C}_t$ is the cost value. Similar to $\hat{A}_t$, $\hat{C}_t$ is the generalized advantage function of the long-term costs and uses the accumulated costs for K steps forward as the baseline

$$\hat{C}_t = \sum_{j=0}^{K} \gamma^j \big[c_t + \gamma V_\delta^c(s_{t+1}) - V_\delta^c(s_t)\big], \tag{12}$$

where c_t is the immediate cost and s_{t+1} is the next state of s_t. Obviously, performing gradient ascent on θ with the collected experiences increases the surrogate objective $J(\theta)$. This operation will update the PTRD toward maximizing the rewards and minimizing the costs, thereby maximizing the long-term throughput while minimizing the privacy violation rate.

In particular, to strike a trade-off between the reward and the cost, the Lagrangian multiplier κ in equation (11) is adaptive and implemented as a parameterized DNN, where κ is initialized to 0. During the training, κ is learned by minimizing the loss function $L(\kappa)$:

$$L(\kappa) = \mathbb{E}_{\mathcal{G}}\big[\kappa(d - V_\delta^c(s_t))\big]. \tag{13}$$

If the constraint is violated, i.e., $d - V_\delta^c(s_t) < 0$, minimizing $L(\kappa)$ will increase κ, consequently leading the actor-network to put more emphasis on the cost term in $J(\theta)$. Otherwise κ will be decreased to make the PTRD emphasize the reward, thereby achieving the adaptive control. Through periodically interacting with the environment and updating network parameters, PTRD steadily improves the performance of the actor network and eventually strikes a trade-off between improving PCN throughput and reducing privacy risks.

4.4 Distributed Training Framework

So far, we have presented how PTRD route each incoming transaction for PCN users. Considering that PCN is a distributed system and each user routes transactions independently, it is impractical to design a centralized DRL agent to make routing decisions for all PCN users. Also, since PCNs typically have tens of thousands of nodes, the concurrency of a centralized method is problematic [30].

A naive approach is to implement PTRD in a fully independent manner [31], where an independent PTRD agent is deployed at each PCN user. From the perspective of each node, it is basically equivalent to single-agent DRL. The independent PTRD agent consists of an actor-network, an adaptive Lagrangian multiplier, a reward critic-network, a cost critic-network, and a experience buffer. The training and interactions are fully independent and each user uses its own collected experiences to update its agent. However, it cannot be ignored that the process of agent-environment interaction before the convergence is often accompanied by trial-and-error costs. Moreover, to train a well-performed DRL agent, a large amount of highly diversified data is often required. As a result, in a fully independent scheme, each PCN user suffers from huge interaction costs caused by trial-and-error [32].

Therefore, we develop a novel distributed training framework that collects the knowledge from each distributed agent to reduce the trial-and-error costs. As shown in Section 4.3, the interaction and training processes can be

completely decoupled, where the actor-network is responsible for interaction and the reward critic-network, cost critic-network, and the adaptive Lagrangian multiplier are responsible to guide policy training. Fig. 6 shows an overview of our distributed training framework. Similar to the fully independent approach, we deploy a distributed DRL agent on each PCN user, except for the agent only containing an actor-network. During each decision epoch, using the actor-network alone can perform interactions. After interactions, PCN users store the collected experience (s_t, a_t, r_t, s_{t+1}) to a global experience buffer (according to steps ❶ to ❹). In addition, all the distributed actor-networks share a global reward critic-network, a global cost critic-network, and a global adaptive Lagrangian multiplier. Periodically, PTRD samples training data from the experience buffer and uses it to update the global networks according to the equations given in Section 4.3. For the distributed actor-networks, PTRD propagates the gradient required for the policy update to each user.

Algorithm 1. PTRD for PCN Routing

Input: Distributed actor-network π_θ, global reward critic-network V_ω^c, global cost critic-network V_δ^c, global Lagrangian multiplier κ, maximum training episode $\mathcal{J}$, maximum interaction epoch T, and training interval t_{train}.

1: **for** episode $j = 1, 2, \ldots, \mathcal{J}$ **do**
2: **for** epoch $t = 1, 2, \ldots, T$ **do**
3: Observe PCN state s_t;
4: Select action with actor-network $a_t \sim \pi_\theta(a_t|s_t)$;
5: Perform action a_t;
6: Receive a reward r_t and a cost c_t, and proceed to a new state s_{t+1};
7: Store experience $(s_t, a_t, r_t, c_t, s_{t+1})$ in the global experience buffer;
8: **if** t mod $t_{\text{train}} = 0$ **then**
9: Sample $(s_1, a_1, r_1, c_1, s_2, \ldots, s_T, a_T, r_T, c_T)$ as training data from the experience buffer;
10: Update the global reward critic-network and global cost critic-network by minimizing equations (5) and (6), respectively;
11: Update the Lagrangian multiplier by minimizing equation (13);
12: Compute the advantage estimations $\hat{A}_t$ and $\hat{C}_t$ with equations (9) and (12), respectively;
13: Compute the clipped surrogate gradient in equation (7);
14: Propagate the policy gradient to each distributed actor-network.
15: Update the actor-network with the received gradient.
16: **end if**
17: **end for**
18: **end for**

On the one hand, the actor networks are distributed because PCN is a distributed system and each user should route transactions independently. On the other hand, the critic network is shared globally to collect the knowledge of all PCN users and to amortize the costs of trial and error. The main idea behind this distributed training framework is that the interaction and training processes can be completely decoupled and each user shares its collected experiences to other users [33]. In this regard, the costs of trial-and-error can be spread over multiple users and the training speed can be greatly improved. Thus, from the perspective of an individual user, the cost is far lower than that under a fully independent training scheme. This entire procedure of PTRD is outlined in Algorithm 1. Lines 2–7 depict the agent-environment interaction. PTRD updates the agent every t_{train} epochs. The global networks are first updated with the sampled experience (lines 10–12) and then the distributed actor-networks are updated with the received policy gradient (lines 13–16).

5 Evaluation

5.1 Evaluation Setup

In this section, we evaluate the performance of PTRD based on the historical data of two real-world PCNs: Ripple and Lightning Network.

5.1.1 Parameters

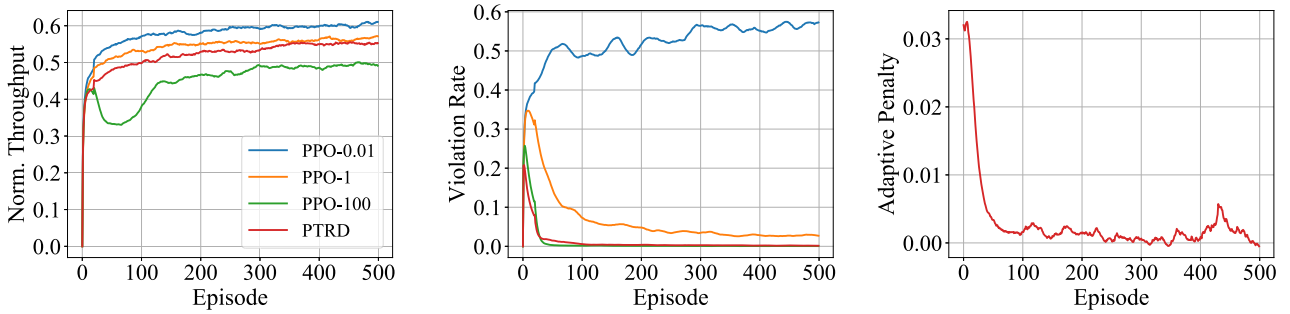
For network topology, we use the topology of Ripple on January 1, 2022. We remove channels without available funds in both directions of channels and remove nodes with only a single neighbor (i.e., they cannot fulfil routing requirements). Consequently, we obtain a PCN with 932 nodes and 43,708 channels. Similar to [5], [6], we consider that the initial PCN channels are perfectly balanced and are evenly assigned with the total funds at both directions. Therefore, the redistributed balances $\hat{b}^{uv} = \hat{b}^{vu} = (b^{vu} + b^{uv})/2$, where b^{vu} and b^{uv} are the instantaneous channel balances of the channel (u, v) in the dataset. To simulate the Ripple workload, we generate payments by randomly sampling from the dataset, which was also used in SpeedyMurmurs [7]. Each payment contains sender ID, receiver ID, and payment amount. Without specification, the number of generated payments in the evaluations is 1,000. To simulate DoS attacks, we randomly select 10 nodes from those nodes containing more than 10 neighbors as malicious nodes.

For the hyperparameters of neural networks, we implement the actor-network, reward critic-network, and cost critic-network, each with two fully-connected layers. Each layer has 256 neurons and the activation function of each layer is the tanh function. The Adam optimizer is used to update the networks. The learning rates for the policy network, two critic-networks, and the Lagrange-multiplier are 3×10^{-4}, 1×10^{-3}, and 5×10^{-2}, respectively. The surrogate objective of the actor-network is clipped with $\epsilon = 0.2$. The discount factor $\gamma = 0.99$. Lastly, we set $\rho = 10$, $d = 0$ for privacy constraints, $K = 50$ for the advantage functions of actor-network and critic-networks, and $t_{\text{train}} = 50$ for distributed training.

5.1.2 Baselines

For comparison, we use the following routing algorithms as baselines:

- *SpeedyMurmurs* [7] is a landmark routing algorithm used in earlier PCN systems. SpeedyMurmurs selects k topologically well-connected nodes as landmarks for routing. We set the number of landmarks to 8, which is the same as the number of candidate paths for PTRD.



(a) Normalized throughput. vs. Episode (b) Privacy violation rate. vs. Episode (c) Adaptive Penalty of PTRD. vs. Episode

Fig. 7. Convergence curves of the PTRD and standard PPO approaches.

- *Waterfilling* [6] is a PCN dynamic routing algorithm that sends transactions through paths with maximum available balances. For each transaction, Waterfilling uses four edge-disjoint paths per destination.
- *Flash* [5] is a state-of-the-art PCN routing algorithm that uses a modified Edmonds-Karp algorithm to probe and finds paths with sufficient balances.
- *PPO-*Θ [26] is a baseline that adds a penalty term Θ to the original reward value, as shown in equation (3), and uses the standard PPO algorithm to maximize this new reward. This method can be considered as a static version of PTRD.

5.1.3 Evaluation Metrics

Similar to prior studies [5], [6], [7], we use the ratio of the number of the completed payments to the total number of the generated payment demands over a given period as the *normalized throughput*. A payment is completed if it reaches the receiver. A higher throughput also means that the algorithm can better prevent DoS attacks. We also use the *privacy violation rate* as the evaluation metric, which is defined as the ratio of the number of payments that violate the privacy constraint during routing to the total number of payment demands.

5.2 Evaluation Results

5.2.1 Convergence Performance

To evaluate the effectiveness of the introduced adaptive Lagrangian-multiplier and the policy optimization method, we compare PTRD with standard PPO approaches in terms of the learning performance. We set the values of penalty term Θ of PPO schemes as $\{0.01, 1, 100\}$. The number of generated payments in the evaluation are 4,000. Fig. 7 plots the learning curves. It can been observed from Figs. 7a and 7b that our PTRD continuously improves the normalized throughput and reduces the privacy violation rate as the training process progresses. Compared with PPO-0.01 and PPO-1 (these two schemes are more concerned with the throughput), PTRD has a comparable throughput while maintaining a much lower privacy violation rate, which is only about 0.04%. By contrast, PPO-0.01 and PPO-1 have privacy violation rates as 53.7% and 2.51%, respectively. On the other hand, although PPO-100 (which is more concerned with privacy constraints) also has a low privacy violation rate, its normalized throughput is about 5% less than our PTRD. These observations are consistent with our observations in Section 4.2. In other words, naively applying standard DRL to PCN routing cannot achieve superior performance. Meanwhile, it also requires the algorithm designers to carefully choose an appropriate penalty value.

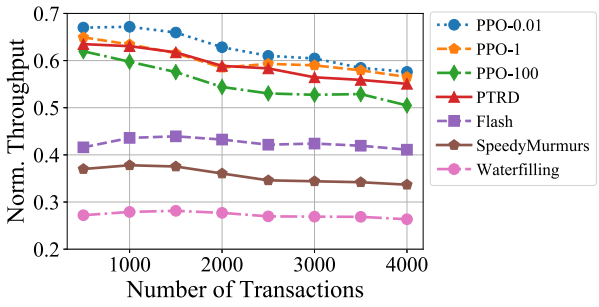
By contrast, we can observe from Fig. 7c that PTRD adaptively updates the penalty. The penalty value is high at the beginning of training. This is because PTRD is randomly initialized and may violate constraints frequently. A high penalty forces the agent to pay more attention to the constraints. When the training proceeds, the privacy violation rate gradually converges to zero. This is because PTRD gradually reduces the penalty value, allowing the agent to pay more attention to the throughput. Therefore, the Lagrangian-multiplier κ used in our PTRD can adaptively control the training process for a better trade-off between improving the throughput and preserving the privacy.

5.2.2 Comparison Under Different Transaction Loads

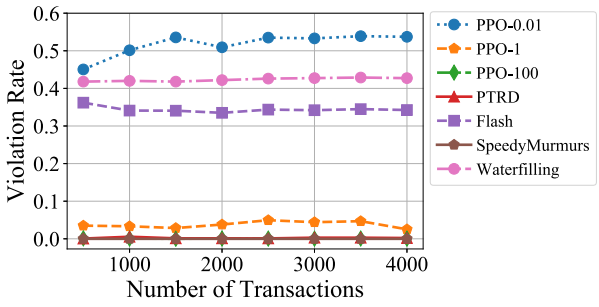
To evaluate the robustness of PTRD against different loads, we vary the number of generated transactions from 500 to 4,000. Fig. 8 shows both the normalized throughput and the privacy violation rate versus the number of transactions. Since SpeedyMurmurs is a static routing algorithm and sends transactions through topologically well-connected nodes, it has a violation rate of 0 (i.e., requiring no channel probing). Fig. 8(a) shows that the throughput of PTRD decreases as the number of transactions increases. This is because the increased number of transactions increases the training complexity. Even so, PTRD still consistently outperforms the baselines. The performance gains of the throughput of our PTRD over Flash, SpeedyMurmurs, and Waterfilling are up to 52.4%, 71.3%, and 133%, respectively. In addition, although the throughput of PPO-100 and PPO-1 is slightly higher than PTRD, their privacy violation rate is much higher, especially for PPO-100, which has an error rate up to 53%. In contrast, Fig. 8(b) shows that the privacy violation rate of PTRD is consistently close to 0 across the evaluation. This demonstrates that even with increased training complexity, the adaptive Lagrangian-multiplier κ of PTRD can still exert excellent effects in weighing transaction throughput and privacy preservation.

5.2.3 Comparison Under Different Privacy Constraints

We next investigate the effectiveness of the adaptive Lagrange-multiplier κ of PTRD under different conditions

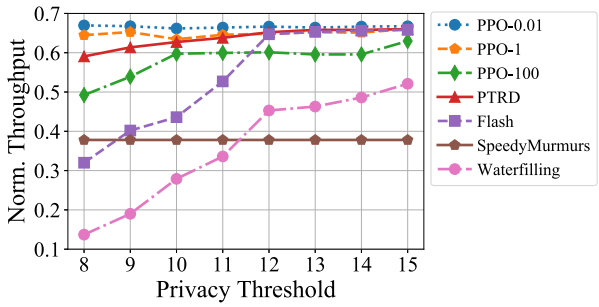


(a) Number of Transactions. vs. Normalized throughput.

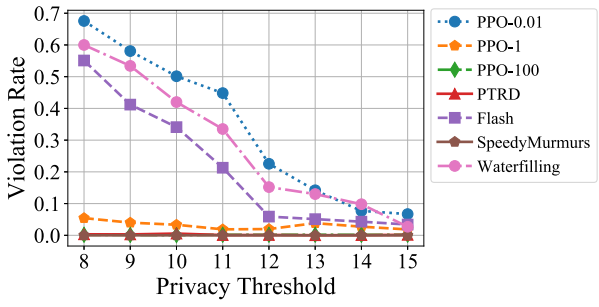


(b) Number of Transactions. vs. Violation Rate.

Fig. 8. Performance results under different transaction loads.



(a) Privacy Threshold. vs. Normalized Throughput



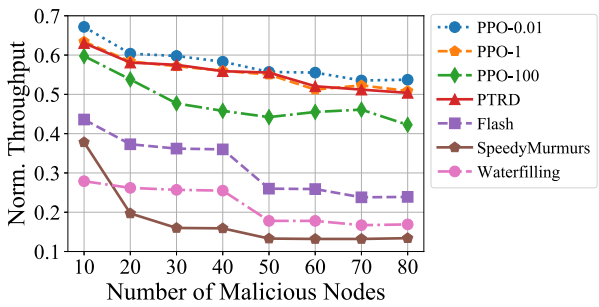
(b) Privacy Threshold. vs. Violation Rate

Fig. 9. Performance results under different privacy constraints.

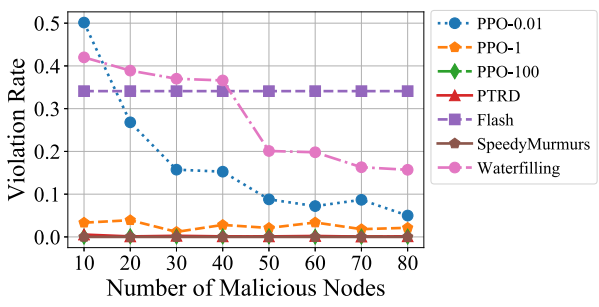
of privacy constraints. We vary the privacy threshold ρ from 8 to 15. A lower ρ means a more stringent constraint. As shown in Fig. 9(a), with the increment of ρ, the throughput of all routing algorithms except SpeedyMurmurs is rising. This is because: 1) SpeedyMurmurs is a topology-based static routing algorithm that does not require probing channel balances; 2) by probing more channels, other routing algorithms have a better chance of finding a viable path to send the transactions, leading to a high throughput. Significantly, our PTRD performs 84.3%, 56%, 330%, and 19.9% better than Flash, SpeedyMurmurs, Waterfilling, and PPO-100 on the throughput (at the minimum privacy threshold). When the threshold is larger than 13, PTRD achieves a throughput close to that of the PPO-0.01. This is close to the maximum throughput since PPO-0.01 pays much attention on throughput. As shown in Fig. 9(b), the violation rate of PTRD is kept near 0 throughout the evaluations, even when the privacy threshold is low. By contrast, the violation rate of PPO-0.01 significantly increases as the privacy threshold increases. This suggests that the adoption of the adaptive Lagrange-multiplier in our PTRD is effective in striking the trade-off between the throughput and privacy constraints.

5.2.4 *Comparison Under Different Numbers of Malicious Nodes*

We next show how the DoS attack impacts on the performance of routing algorithms. To simulate different levels of DoS attack, the number of malicious nodes is varied from 10 to 80. Note that the selected malicious nodes are topologically well-connected, so only a few nodes are needed to launch a serious attack on the PCN. Fig. 10a shows that the throughput of all routing algorithms decreases as the number of malicious nodes increases. This trend is inevitable because the PCN exhibits similar properties to the scale-free network [21] and most transactions are relayed by a few



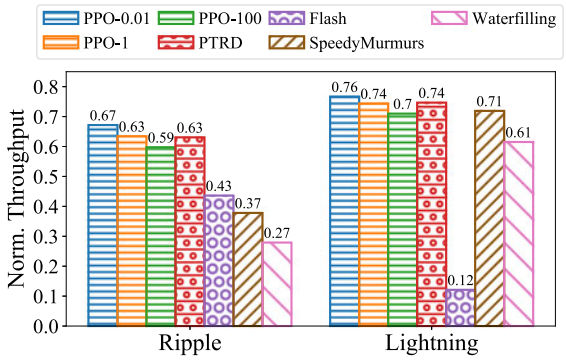
(a) Number of Malicious Nodes. vs. Normalized Throughput



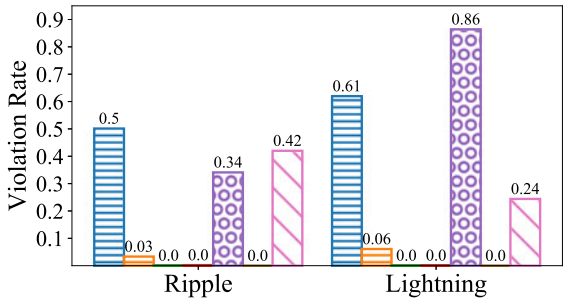
(b) Number of Malicious Nodes. vs. Violation Rate

Fig. 10. Performance results under different numbers of malicious nodes.

nodes with high betweenness centrality [10], [13]. As more and more nodes with high betweenness centrality are compromised, it will be more difficult to find paths with sufficient balances to send transactions. Nevertheless, PTRD still achieves a much higher throughput than PPO-100, Flash, SpeedyMurmurs, and Waterfilling and a much lower privacy violation rate than Flash, Waterfilling, PPO-0.01, and PPO-1, as shown in Fig. 10b. This result demonstrates the superiority of the adopted DRL in our PTRD, thereby



(a) Normalized throughput in Ripple and Lightning Network.



(b) Violation Rate in Ripple and Lightning Network.

Fig. 11. Performance results under different network topologies.

learning hidden information from historical experiences and preventing PTRD from selecting malicious nodes as routing relays.

5.2.5 Comparison Under Different Network Topologies

Lastly, we compare the performance of PTRD under different network topologies to evaluate its robustness. Specifically, we use the historical topologies of the Ripple and Lightning Network. The setting of Ripple is presented in Section 5.1.1. For the Lightning Network, we crawl its full topology on January 1, 2022, where the processed topology has 791 nodes and 3,845 channels. Similar to the Ripple, we remove channels without available funds in both directions of channels and remove nodes with only a single neighbor. Fig. 11 shows that in both Ripple and Lighting Network topologies, PTRD has achieved a comparable throughput performance to PPO-100 and PPO-1 while maintaining a much lower violation rate. Meanwhile, compared with SpeedyMurmurs, PTRD achieves about 66.6% and 4.2% throughput improvement in both Ripple and Lightning Network, respectively. These results confirm the generality of PTRD, which effectively improves the transaction throughput in various situations while ensuring compliance with privacy constraints.

6 Related Work

PCNs have recently emerged since 2016 while they are still in their early development. According to whether the routing algorithm considers the real-time PCN states when scheduling transactions, they can be divided into static scheduling and dynamic scheduling.

6.1 Static Scheduling

In static scheduling, PCN nodes schedule transactions according to pre-configured rules without probing instantaneous channel balances, where these rules are typically designed based on network topologies or node addresses. Given the same topology, the sending path obtained through path discovery for transactions with the same receiver is constant. Therefore, this scheduling strategy is called static scheduling. Similar ideas can be found in traditional communication networks. For instance, the Open Shortest Path First (OSPF) Protocol and Routing Information Protocol (RIP) have been widely used to compute the shortest paths or paths with the fewest hops in interior gateway routing.

Due to the implementation simplicity, the static scheduling algorithms have widely been used in early PCN systems. Prihodko et al. proposed Flare [8] for the Lightning Network. In Flare, each user maintains a local view of its k-neighborhood, i.e., at most k neighboring nodes within a hop distance. For transactions sending to the k-neighborhood of a user, Flare computes the shortest paths to the receivers based on its local view. In addition, for receivers that are not k-neighborhood, Flare asks nearby beacon nodes to send transactions; these beacon nodes are typically highly-connected. Considering the overhead of maintaining a local view of k-neighborhood, Malavolta et al. proposed the landmark-based routing algorithm called SlientWhispers [34], where each user maintains a set of landmark nodes and sends transactions through landmarks. Periodically, each user calls SlientWhispers to compute the shortest paths to the landmarks. Since all transactions have to pass through the landmarks, their processing capacities become the major bottleneck of the SlientWhispers. Roos et al. proposed a embedding-based routing algorithm called SpeedyMurmurs [7], where transactions are sent according to the IDs of the users. The performance of SpeedyMurmurs are severely affected by the embedding method. These static scheduling algorithms tend to send transactions according to the topology and focus on finding the shortest path while not considering the available channel balances along the sending paths. Consequently, this design severely limits the network throughput of PCNs.

6.2 Dynamic Scheduling

In dynamic scheduling, PCN nodes schedule transactions according to the real-time PCN states. As mentioned earlier, the premise of the successful delivery of transactions is to find paths with sufficient balances. Compared with static scheduling, dynamic scheduling algorithms tend to achieve the higher throughput. Many emerging PCN routing schemes fall into this category. For instance, Wang et al. proposed Flash [5], which divides transactions into mice transactions and elephant transactions based on their amount. For mice transactions, Flash directly looks up a routing table that stores the paths to the previous receivers. For the elephant transactions, Flash uses a modified Edmonds-Karp algorithm to probe the available channel balances and find the shortest paths with sufficient balances. Based on the idea of congestion control in data communication network, Sivaraman et al. proposed Spider [6], where each user

builds up the router-signalling queue to describe the real-time congestion magnitude of the channels. A channel is considered to be congested when there are one-way transactions passing through for a long time. In this case, the channel balances that eventually become depleted will fail to support further transactions toward that direction. When the congestion occurs, Spider sends the congestion signal to suppress transactions in that direction.

The acquisition of real-time channel balances is the prerequisite for dynamic scheduling. Because the instantaneous channel balances are not publicly announced, the sender is forced to iteratively probe the channel balances until finding a successful path (if any) to support transactions. How to make a balance between improving transaction throughput and decreasing the privacy risk caused by channel probing is a major challenge for dynamic scheduling algorithms. For instance, the modified Edmonds-Karp algorithm used in Flash only focuses on the shortest paths with sufficient balances. As a result, it may probe a large number of channels. This violates the design intent of PCNs. By contrast, PTRD achieves both privacy protection and high throughput with a novel DRL-based model and distributed training.

7 Conclusion

In this paper, we have studied the routing problem in PCNs. To make the trade-off between privacy constraints and the transaction throughput, we propose PTRD, a privacy-aware high-throughput routing algorithm with the defence against DoS attacks based on DRL. PTRD formulates the routing problem as a CMDP and extends off-the-shelf DRL algorithms by introducing an adaptive Lagrangian multiplier. In particular, since PCN is a distributed system, a distributed training framework is proposed by decoupling the interaction process from the training process. Compared with state-of-the-art PCN routing algorithms, PTRD can increase the long-term throughput by 2.7%–62.5%, thereby demonstrating the superiority of our PTRD in routing transactions from a long-term optimization perspective. We also demonstrate that PTRD can learn hidden information from historical experiences, consequently avoiding to select malicious nodes as routing relays. In sum, PTRD also has a better trade-off between improving the throughput and decreasing the privacy risks than traditional approaches.

References

[1] 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 DataCongr.*, 2017, pp. 557–564.

[2] 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.

[3] "Bitcoin," 2019. [Online]. Available: https://bitcoin.org/en/

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

[5] 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.*, 2019, pp. 370–381.

[6] V. Sivaraman et al., "High throughput cryptocurrency routing in payment channel networks," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 777–796.

[7] 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.*, 2018, pp. 1–15.

[8] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," White Paper, 2016. [Online]. Available: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf

[9] W. Tang, W. Wang, G. C. Fanti, and S. Oh, "Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 29:1–29:39, 2020.

[10] B. Weintraub, C. Nita-Rotaru, and S. Roos, "Exploiting centrality: Attacks in payment channel networks with local routing," 2020, *arXiv:2007.09047*.

[11] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, "Bdos: Blockchain denial-of-service," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Virtual Event*, 2020, pp. 601–619.

[12] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2021, pp. 170–188.

[13] S. Tochner, S. Schmid, and A. Zohar, "Hijacking routes in payment channel networks: A predictability tradeoff," 2019, *arXiv:1909.06890*.

[14] S. Tochner, A. Zohar, and S. Schmid, "Route Hijacking and DoS in Off-Chain Networks," in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, 2020, pp. 228–240.

[15] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, "A survey of state-of-the-art on blockchains: Theories, modelings, and tools," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 44:1–44:42, 2021.

[16] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, and Y. Zhang, "Deep reinforcement learning for internet of things: A comprehensive survey," *IEEE Commun. Surveys Tut.*, vol. 23, no. 3, pp. 1659–1692, Third Quarter 2021.

[17] A. Ray, J. Achiam, and D. Amodei, Benchmarking safe exploration in deep reinforcement learning, 2019. [Online]. Available: https://cdn.openai.com/safexp-short.pdf

[18] "Ripple," 2019. [Online]. Available: https://ripple.com/

[19] P. Li, T. Miyazaki, and W. Zhou, "Secure balance planning of off-blockchain payment channel networks," in *Proc. Conf. Comput. Commun.*, 2020, pp. 1728–1737.

[20] N. Vallarano, C. J. Tessone, and T. Squartini, "Bitcoin transaction networks: An overview of recent results," *Front. Phys.*, vol. 8, 2020, Art. no. 286.

[21] I. A. Seres, L. Gulyás, D. A. Nagy, and P. Burcsi, "Topological analysis of bitcoin's lightning network," in *Proc. 1st Int. Conf. Math. Res. Blockchain Economy*, 2019, pp. 1–12.

[22] S. Martinazzi and A. Flori, "The evolving topology of the lightning network: Centralization, efficiency, robustness, synchronization, and anonymity," *PLoS One*, vol. 15, no. 1, pp. 1–18, Jan. 2020.

[23] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 801–815.

[24] L. Costero, A. Iranfar, M. Zapater, F. D. Igual, K. Olcoz, and D. Atienza, "Resource management for power-constrained HEVC transcoding using reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2834–2850, 2020.

[25] B. Thananjeyan et al., "Recovery RL: Safe reinforcement learning with learned recovery zones," *IEEE Trans. Robot. Autom.*, vol. 6, no. 3, pp. 4915–4922, Jul. 2021.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, 2017, *arXiv:1707.06347*.

[27] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. New York, NY, USA: Academic Press, 1985.

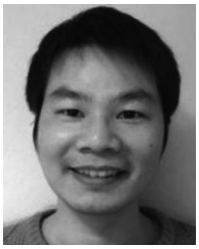
[28] S. T. Tokdar and R. E. Kass, "Importance sampling: A review," *Wiley Interdiscipl. Rev.: Comput. Statist.*, vol. 2, no. 1, pp. 54–60, 2010.

[29] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[30] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2021.

[31] 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. 2021.
[32] 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.
[33] 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.
[34] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *Proc. 24th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–15.

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.



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.

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, off-chain transactions and deep reinforcement learning.

Junyuan Liang received the BS degree from Sun Yat-sen University, Guangzhou, China, in 2020. He is currently working toward the MS degree with Sun Yat-sen University, Guangzhou, China. He is working on edge computing and blockchain system, in particular on storage, payment channel network and sharding in blockchain.

Hong-Ning Dai (Senior Member, IEEE) received the PhD degree in computer science and engineering from the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He is currently an associate professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His current research interests include the internet of things, big data, and blockchain technology. He is also a senior member of the Association for Computing Machinery (ACM). He has served as an associate editor/editor for the *IEEE Transactions on Industrial Informatics*, *IEEE Systems Journal*, *IEEE Access*, *Ad Hoc Networks*, and *Connection Science*.

Zibin Zheng (Senior Member, IEEE) is currently a professor and the deputy dean with the School of Software Engineering, Sun Yat-sen University, Guangzhou, China. He has 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 with ICSE2010, the Best Student Paper Award with ICWS2010.