Optimal Hub Placement and Deadlock-Free Routing for Payment Channel Network Scalability

Lingxiao Yang*, Xuewen Dong*,✉, Sheng Gao†, Qiang Qu‡, Xiaodong Zhang*, Wensheng Tian§, Yulong Shen*
*School of Computer Science and Technology, Xidian University, Xi'an, China,
lxyang@stu.xidian.edu.cn, {xwdong, zhangxiaodong}@xidian.edu.cn, ylshen@mail.xidian.edu.cn
†School of Information, Central University of Finance and Economics, Beijing, China. sgao@cufe.edu.cn
‡Shenzhen Institute of Advanced Techbology, Chinese Academy of Sciences;
Huawei Blockchain Lab, Huawei Cloud Tech Co., Ltd, Shenzhen, China. quqiang4@huawei.com
§Nanhu Lab, Jiaxing, China. tws@nanhulab.ac.cn

***Abstract*—As a promising implementation model of payment channel network (PCN), payment channel hub (PCH) could achieve high throughput by providing stable off-chain transactions through powerful hubs. However, existing PCH schemes assume hubs are preplaced in advance, not considering payment requests' distribution and may affect network scalability, especially network load balancing. In addition, current source routing protocols with PCH allow each sender to make routing decision on his/her own request, which may have a bad effect on performance scalability (e.g., deadlock) for not considering other senders' requests. This paper proposes a novel multi-PCHs solution with high scalability. First, we are the first to study the PCH placement problem and propose optimal/approximation solutions with load balancing for small-scale and large-scale scenarios, by trading off communication costs among participants and turning the original NP-hard problem into a mixed-integer linear programming (MILP) problem solving by supermodular techniques. Then, on global network states and local directly connected clients' requests, a routing protocol is designed for each PCH with a dynamic adjustment strategy on request processing rates, enabling high-performance deadlock-free routing. Extensive experiments show that our work can effectively balance the network load, and improve the performance on throughput by 29.3% on average compared with state-of-the-arts.**

***Index Terms*—Payment Channel Network, Placement, Routing, Scalability.**

I. Introduction

Cryptocurrencies are gaining popularity in the financial ecosystem. However, the scalability issues of their underlying blockchain technology are still challenging. Since each transaction needs to be confirmed by the consensus mechanism, this can take several minutes to hours. Instead of continually improving the design of the consensus mechanism, a leading layer-2 proposal for addressing the scalability challenge relies on off-chain payment channels [1], [2]. The core idea is to move mass transactions submitted on-chain to off-chain and execute them securely using a locking mechanism. Only the key steps (e.g., resolving disputes, opening/closing channels) are put on-chain for confirmation.

Multiple payment channels among nodes constitute a *payment channel network* (PCN). Two nodes without the direct payment channel can conduct off-chain transactions through intermediaries routing. While appealing, PCNs raise the issue of finding paths by the senders and maintaining the network topology. Furthermore, the collateral deposited in a channel cannot be used anywhere else in a bounded time. The above reasons prompted the design of TumblerBit [3], which first proposes the *payment channel hubs*.

Payment channel hubs (PCHs) are the untrusted intermediaries that allow participants to make fast, anonymous, off-chain payments [3]. The basic idea is that each participant opens a channel with a PCH. The PCH mediates payments between senders and recipients and gains a fee. Although this scheme sacrifices the fully decentralized nature of blockchain, it significantly improves the performance on the premise of verifiable security [4]. Blockchain has been compromising decentralization in the increasingly urgent need for high availability. For example, EOS [5] has compromised from decentralization to multi-centralization.

Motivation. With the increase in usage frequency, the overall load of the PCN rises rapidly, and load imbalance phenomena happen gradually. When the blockchain community[1] decides to upgrade or design a new PCN for large-scale usage scenarios, the overall load of the PCN system should be considered. The inappropriate placement of PCHs in PCNs can easily lead to an unbalanced network communication load. However, as shown in Table I, the existing scalable schemes (e.g., [1], [2], [6]–[10]) mainly study the improvement of routing strategies to improve performance. The placement problem has never been discussed, which is the *first flaw*. The current source routing they adopted requires each sender to maintain a complete PCN topology and independently compute routes according to his/her own requests [9], [10]. In large-scale networks, the senders' performance is severely challenged, and without coordinating other requests can easily cause deadlocks. Existing PCHs (e.g., [3], [4], [11], [12]) inherit the above source routing flaw and design complex cryptographic primitives to provide unlinkability [3], [4] for off-chain payments to obfuscate the relationship between transaction parties, with limited scalability improvements [11], [12], which is the *second flaw*.

✉ Xuewen Dong is the corresponding author.

[1]Like all public blockchains, our solution is also community-autonomous. This means that all members have equal rights in decision-making, which requires a 67% majority approval.

2575-8411/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDCS57875.2023.00087

TABLE I
STATE-OF-THE-ART PCN SCALABLE SCHEMES

Properties \ Literatures	Lighting [1], Raidon [2] Flare [6], Sprites (FC '19) [7]	REVIVE (CCS '17) [8]	Spider (NSDI '20) [9]	Flash (CoNEXT '19) [10]	TumbleBit (NDSS '17) [3] A^{2}L (S&P '21) [4]	Perun (S&P '19) [11] Commit-Chains [12]	Splicer (This work)
Improving throughput	●	●	●	●	●	●	●
Support large transactions	○	○	●	●	○	○	●
Payment channel balance	○	●	●	○	○	○	●
Deadlock-free routing	○	◐	◐	○	○	○	●
Transaction unlinkability	○	○	○	○	●	○	●
Optimal hub placement	○	○	○	○	○	○	●

This paper presents *Splicer*, a novel multi-PCHs solution with high scalability while inheriting unlinkability. In particular, we propose the PCHs called the *smooth nodes* for routing computations. The scalability of Splicer is two-fold: (i) Our network scale is scalable, allowing more clients to access the system through a variable number of PCHs. Meanwhile, we study the PCH placement problem to achieve optimal network communication load balancing. (ii) Splicer provides performance scalability. We design a rate-based routing protocol to perform multi-path payments by splitting transactions, achieving a high *transaction success ratio* and *throughput*. It proves that the network funds flow smoothly and the network is nearly deadlock-free. The ***insight*** of Splicer is to seek a new tradeoff between decentralization and scalability for PCNs[2].

Challenges. Splicer addresses several challenges: (i) *PCH placement modeling and solving.* Since the clients in PCNs are scattered geographically, the desired properties are challenging to define formally. In addition, the method of solving the model may change as the scale of PCNs increases. (ii) *A scalable routing protocol for PCHs.* Naive approaches (e.g., shortest-path routing) may lead to underutilizing funds or deadlock in some channels because the transaction always flows on the shortest paths. Additionally, previous instant and atomic routing [1], [2] would cause the payment value to be limited by channel funds.

Contributions. We make the following contributions:

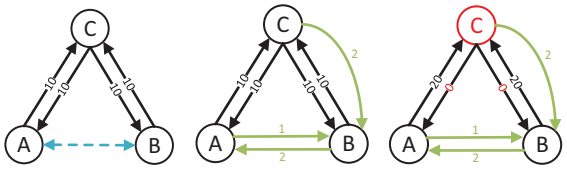
- **Globally optimal PCH placement.** Multiple PCHs make distributed routing computations for client payment requests to balance the network load. We model the PCH placement problem to minimize the communication delay and overhead. Besides, we propose two solutions for the placement optimization problem in small-scale and large-scale networks.
- **Deadlock-free routing protocol.** We design a scalable rate-based routing mechanism for the multiple PCHs, allowing large-value transactions to be completed on low-capacity channels in multi-path and enabling high-throughput without disrupting the balance of channel funds. Additionally, we consider congestion control to adjust the transaction flow rates in the PCNs further.
- **Evaluation.** We implement Splicer using Lightning Network Daemon (LND) testnet [13]. We simulate the PCHs placement model and conduct extensive experiments to evaluate the performance of Splicer. The results show that Splicer can effectively balance the network load, and improve the performance on transaction success ratio by 42% and throughput by 29.3% on average compared with the state-of-the-arts.

[2]Our system name Splicer is derived from this insight, which rationally connects the clients in PCNs with multiple PCHs.

II. PRELIMINARIES

A. Payment Channel Networks

Fig. 1(a) shows an example of payment channel networks in which (A, C) and (C, B) have established a bidirectional payment channel, respectively. Each direction of the bidirectional payment channels has deposited ten tokens. So a virtual payment channel [11] is formed between A and B. C relays the payment if A wants to send five tokens to B. Thus two transactions occur, A to C and C to B. As an incentive to participate, C receives a forwarding fee. A key challenge is ensuring that C forwards the correct amount of tokens. Thus, the cryptographic *hash time lock contract* (HTLC) [1] is proposed to guarantee that C can get the tokens paid by A on the channel (A, C) only after C has successfully paid to B within a bounded time in the channel (C, B).



(a) A simple PCN. (b) The initial state. (c) A deadlock at *C*.

Fig. 1. Examples illustrating the PCNs.

B. Local Deadlocks in PCNs

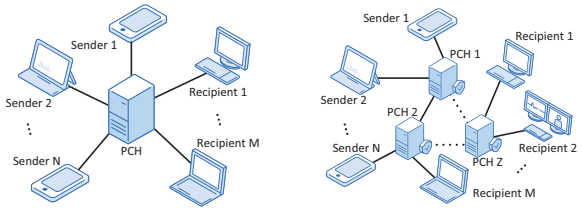
To illustrate the local deadlocks in PCNs, we consider an initial state in Fig. 1(b) that A and C transfer funds to B at a rate of 1 and 2 tokens/sec, respectively. B transfers funds to A at a rate of 2 tokens/sec. It is noted that the payment rates we specify are not balanced, which causes net funds to flow out of C and into A and B. Payment channels are balanced by ten tokens in each direction. We suppose that the transactions only flow between A and B. The system can achieve a total throughput of 2 tokens/sec only by allowing A and B to transfer to the opposite at a rate of 1 token/sec. But once the payments are executed as described in the initial state, the system throughput ends up at zero. This is because C sends B funds faster than its funds being replenished by A, reducing its transferable funds to zero, as shown in Fig. 1(c). However, C needs positive funds to route transactions between A and B. The transactions between A and B cannot be processed, even though they have sufficient funds. Thus the network goes into a *deadlock* state.

III. SPLICER: PROBLEM STATEMENT

A. System Model

Entities. There are two types of entities in Splicer:

- **Clients** are the end-users in PCNs who can send or receive a payment. We expect the clients to be lightweight, allowing mobile or IoT devices to make payments. Clients outsource the payment routing computation to smooth nodes. Each client interacts with a unique smooth node.



(a) Star-like topology. (b) Multi-star-like topology.

Fig. 2. PCH topologies.

- **Smooth nodes** process payment requests from clients by running routing protocol. Each smooth node makes path decisions for the current payment requests of its directly linked clients. Besides, multiple smooth nodes form a key management group (KMG) to create or retrieve keys with any distributed key generate protocol [14].

Topology. Fig. 2(a) shows the star-like topology in state-of-the-art PCHs [3], [4], [11]. A PCH opens payment channels with multiple clients separately. Thus, clients need one-hop routing for mutual payments through the intermediate PCH [12]. As shown in Fig. 2(b), in Splicer we model the PCN with a *multi-star-like topology*, in which the clients evenly connect to the smooth nodes (PCHs). Fig. 2(b) shows an example where the client consists of N senders and M recipients, and PCHs include Z smooth nodes. We define the multi-star-like PCN topology as follows.

Definition 1. *(Multi-star-like PCN topology). In a PCN, there are multiple PCHs connected directly or indirectly. The clients trade with each other through the PCHs that are directly connected to them.*

Notice that the payment hub model has been widely adopted in PCNs (e.g., [3], [4], [11], [12]). However, we are the first to propose a multi-star-like PCN with multiple PCHs.

Workflow. A PCN can be modeled as a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ in which $\mathbb{V}$ denotes the set of nodes, and $\mathbb{E}$ denotes the set of payment channels between them. $\mathbb{V}_{\text{CLI}} \subseteq \mathbb{V}$ denotes the clients, and $\mathbb{V}_{\text{SN}} \subseteq \mathbb{V}$ denotes the smooth nodes, where $\mathbb{V}_{\text{CLI}} = \mathbb{V} - \mathbb{V}_{\text{SN}}$. As shown in Fig. 3, there is a payment from client P_s to client P_r, S_i and S_j are the smooth nodes to which they are connected, for $P_s, P_r \in \mathbb{V}_{\text{CLI}}$ and $S_i, S_j \in \mathbb{V}_{\text{SN}}$. KMG contains ι smooth nodes, where ι is a system parameter. We now sketch the payment preparation and execution workflow.

a) Payment preparation: For simplicity, we omit the detailed process of creating payment channels, and the channels' initial deposits are sufficient, similar to Ref. [15], [16]. Now P_s establishes a secure communication via transport layer security (TLS) protocol with S_i, so do P_r and S_j. Then a payment channel is created between P_s and S_i, so do P_r and S_j. Next, P_s initiates a payment request $\mathsf{pay}_{\text{req}}$ to S_i via the secure channel so that S_i knows the client has a new transaction to execute. Then S_i starts the *payment initialization*. S_i creates a fresh transaction id tid and obtains fresh ($\mathsf{pk}_{\mathsf{tid}}$, $\mathsf{sk}_{\mathsf{tid}}$) pair from the KMG. S_i sends tid and the corresponding public key $\mathsf{pk}_{\mathsf{tid}}$ to P_s, and S_i keeps $\mathsf{sk}_{\mathsf{tid}}$ private. Then S_i generates the initial state $\mathsf{state}_{\mathsf{tid}} := (\mathsf{tid}, \theta_{\mathsf{tid}})$, a tuple containing tid and a boolean θ_{tid} indicating whether the transaction is completed.

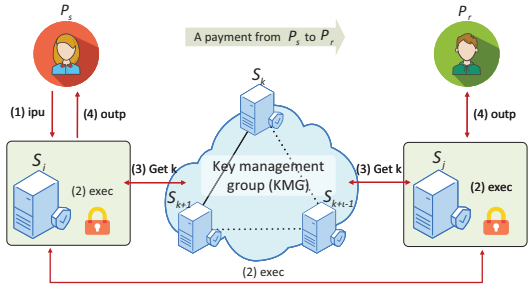


Fig. 3. Workflow in multi-hop working model.

b) Payment execution: We illustrate the steps of payment execution in Fig. 3 as follows:

(1) To start the process of executing a transaction tid with input inp which contains D_{tid}. Let $D_{\mathsf{tid}} = (P_s, P_r, \mathsf{val}_{\mathsf{tid}})$ denote the payment demand of P_s, where $\mathsf{val}_{\mathsf{tid}}$ represents the payment amount. P_s first computes $\mathsf{inp} = \mathsf{Enc}(\mathsf{pk}_{\mathsf{tid}}, D_{\mathsf{tid}})$, then sends to S_i a message $(\mathsf{tid}, \mathsf{inp})$ and the payment funds.

(2-3) S_i decrypts inp with $\mathsf{sk}_{\mathsf{tid}}$ to obtain $D_{\mathsf{tid}} = \mathsf{Dec}(\mathsf{sk}_{\mathsf{tid}}, \mathsf{inp})$. Then the routing protocol splits the D_{tid} into K transaction-units (TUs) D_{tuid} with fresh id tuid, generates the corresponding states $\mathsf{state}^i_{\mathsf{tuid}} = (\mathsf{tuid}, \theta^i_{\mathsf{tuid}})$, for $2 \le i \le K$, and $\theta_{\mathsf{tid}} = \bigwedge_{2 \le i \le K} \theta^i_{\mathsf{tuid}}$. From the KMG, S_j obtains $(\mathsf{pk}_{\mathsf{tuid}}, \mathsf{sk}_{\mathsf{tuid}})$ pair. S_i sends $\mathsf{Enc}(\mathsf{pk}_{\mathsf{tuid}}, D_{\mathsf{tuid}})$ to S_j, then S_j decrypts it with $\mathsf{sk}_{\mathsf{tuid}}$. Once S_j receives the funds corresponding to D_{tuid}, it returns to S_i an acknowledgment $\mathsf{ACK}_{\mathsf{tuid}}$ via secure channel, with which S_i updates the $\mathsf{state}^i_{\mathsf{tuid}}$ as $\theta^i_{\mathsf{tuid}} = \mathsf{true}$. When all acknowledgments $\mathsf{ACK}_{\mathsf{tuid}}$ are received, S_i updates the $\mathsf{state}_{\mathsf{tid}}$ as $\theta_{\mathsf{tid}} = \mathsf{true}$.

(4) Finally, S_j receives all TUs of D_{tid}, and sends the payment funds to P_r at one time. P_r generates a successful receipt acknowledgment $\mathsf{ACK}_{\mathsf{tid}}$, which is finally forwarded to P_s by the smooth nodes.

In fact, in the above workflow, any transaction initiator needs to pay an additional forwarding fee to the intermediaries on the routing path, which is used as a forwarding incentive for smooth nodes, see §IV-D.

B. Trust, Communication and Threat Models

Trust model. Splicer is community-autonomous, and there is a certain trust transference between entities. In Fig. 4, Splicer runs a multiwinner voting algorithm (e.g., [17]) in the smart contract that effectively allows all entities to fairly select a *smooth node candidate list* in a long period. It can take into account the two properties: (i) *Excellence* means the selected candidates are "better" for outsourcing routing tasks (e.g., have more client connections, transaction funds, and lower operational overhead). (ii) *Diversity* means that the candidate positions are as diverse as possible. We leave the optimal design of multiwinner voting for future work.

The first selected candidate list of smooth nodes temporarily performs payment routing as actual PCHs. When the network state is stable, the candidate smooth nodes run a smart contract containing a placement optimization algorithm to determine the actual PCHs (long term running). Notice that after the distribution of transaction requests is stabilized in the network, the overall distribution information of requests obtained by

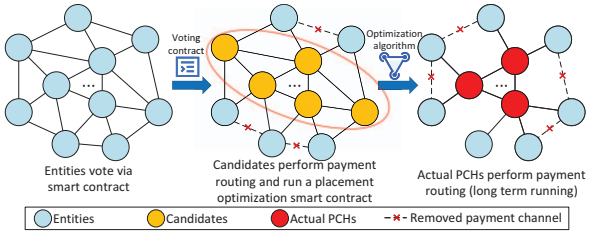


Fig. 4. System trust transference model.

each candidate PCH is consistent, and the actual PCHs finally decided are consistent.

Fig. 4 shows that this process is accompanied by the removal of redundant payment channels, thus reducing the complexity of the network. Actual PCHs require pledging funds to a public pool for access, and their behavior checks and balances each other; if some PCHs appear malicious or colluding, they will be identified by other PCHs. Splicer also provides the client with a reporting and arbitration mechanism. The malicious PCHs will be removed, and their deposit will be confiscated as a punishment (the loss is greater than the profit). Then new PCHs will be selected from the new candidate list to supplement, so rational PCHs will not choose corruption. We emphasize that sensitive information (e.g., node identity, transaction content, routing data) is transmitted in ciphertext, coupled with unlinkability, so the client does not need to worry about privacy leakage.

Communication model. As shown in Fig. 5, we sketch the communication process. Splicer runs under the bounded synchronous communication setting. At the beginning of epoch e+1, PCHs obtain and synchronize the final global information of the last epoch, including clients' states and network data (e.g., topology, channel state, payment flow rate, etc). Meanwhile, after receiving directly connected clients' local payment requests, each PCH makes distributed routing decisions based on the network data (final global information of epoch e) and its clients' new requests (local information of epoch e+1). Finally, recipients generate the payment acknowledgments, which PCHs forward to senders. Splicer loops the above process.

Threat model. Each PCH is rational and potentially malicious, deviating from the protocol to obtain benefits. An adversary may compromise a target PCH's operating system and network stack, which can arbitrarily drop, delay, and replay messages. Since the adversary would not profit from corrupting the PCH placement process, the adversary's attack may only cause the payment routing of some transactions to fail. However, the failed transactions would be withdrawn by PCH without causing losses to the client or affecting the system's stability.

C. PCH Placement and Routing Problems

Placement problem. The core of placement problem is to select the actual PCHs (fixed long-period running placement optimization smart contract) in the smooth node candidate list and make them install and run the PCH program for payment routing. The placement optimization smart contract determines the number of actual PCHs and their location in the network. We emphasize that this is a community-autonomous process and not a centralized decision. In long-term stable operation, the actual PCHs do not change, unless the network is not in an

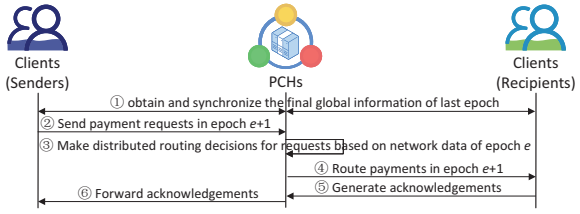


Fig. 5. System communication process in epoch e+1.

optimal operating state after long-term operation (the result of the placement optimization smart contract output changes) or a malicious PCH is removed. In practice, the community weighs the costs and benefits to determine when a new placement problem should be addressed.

Since there are many geographically dispersed nodes in the real PCNs, PCHs may be far from some clients, leading to unstable connections or high communication delay and overhead between them. We aim to place the PCHs in proximity to the clients evenly, and all the clients have the lowest average payment hops forwarded by them. PCHs are physically distributed but logically polycentric and cooperative in managing payment routing. While such a placement strategy makes the distance between nodes shorter, it also considers the costs of PCHs collecting statistics from the clients and synchronizing between the PCHs. This creates a *network load tradeoff*: (i) the PCHs should be near their routed clients to reduce the communication delay and route management overhead (**management cost**). (ii) they should be close to each other to reduce the delay and overhead of synchronizing states (**synchronization cost**). Thus the PCHs should be appropriately placed in a PCN, leading to a placement problem. We are the first to discuss the PCH placement problem in the PCNs, and we further describe the details of the placement problem in §IV-B and the solutions in §IV-C.

Routing problem. The existing routing solutions try to: (i) reduce routing costs to improve throughput and (ii) rebalance channel funds to improve routing performance. However, source routing requires each sender to compute the routing paths, which is limited in large-scale scenarios (Until April 11, 2023, the total number of nodes in the Lightning Network is 16,427. This paper defines a local network with more than 3,000 nodes as a large-scale network). We need to design a distributed routing decision protocol over multiple PCHs. Thus we propose a rate-based routing mechanism inspired by the ideas of packet-switching technology. Transactions are split into multiple independently routed TUs by each PCH. Each TU can transfer a few funds bounded by a *Min-TU* and a *Max-TU* value at different rates. We emphasize that this multi-path payment routing approach has proven to be feasible in Spider [9]. Notice that it does not affect the confidentiality of payments because each TU is encrypted with an independent public key from the KMG. In addition, the intermediate nodes of each TU routing path may be different, which confuses the relationship between two-party of multiple original transactions in PCNs. Thus, Splicer inherits the unlinkability of state-of-the-art PCHs. It is more difficult for intermediaries to identify sensitive information ciphertext. We elaborate on the rate-based routing protocol in §IV-D.

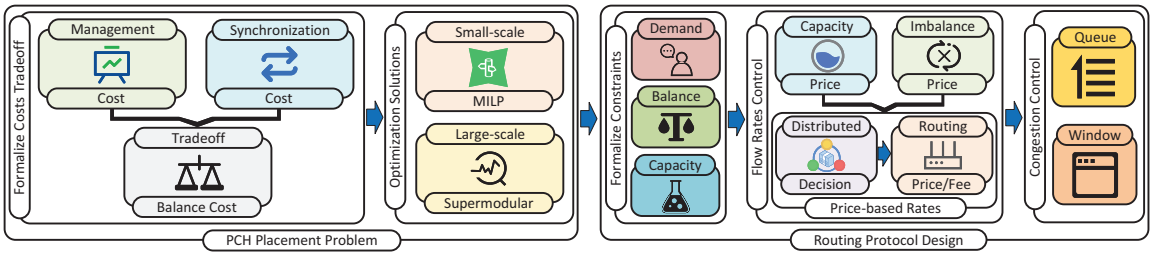


Fig. 6. The overall structure of our design.

IV. System Design

A. Overview

The overall structure of our design is shown in Fig. 6. We first consider the placement of smooth nodes. The placement problem is to seek a tradeoff between routing management and synchronization costs. Now we study the details of the smooth nodes placement problem, formulating it as an optimization problem with two costs tradeoff and translating it into minimizing the *balance cost*. Then, we provide solutions in two PCN scales for the transformed optimization problem. In a small-scale network, we transform the placement problem into a *mixed-integer linear programming* (MILP) problem to find the optimal solution. We use supermodular function techniques to find the approximate solution in a large-scale network.

Next, we study the details of routing protocol design for smooth nodes. We first give the formal constraints of the routing problem, including *demand*, *capacity*, and *balance* constraints. Then we consider the transaction flow rate control based on the routing price. We define the *capacity and imbalance prices*, derive the *routing price and fee* through distributed decisions, and obtain the price-based flow rates. Finally, we consider congestion control during routing and design the waiting queue and window to alleviate congestion.

B. Formalize the Placement Problem

We describe the long periodic election of smooth node candidate list in the trust model of §III-B. We briefly present the PCH placement problem in §III-C. Next, we further model how to determine the actual PCHs in the candidate list.

To formally describe the network load tradeoff, we let two binary variables $x_n, y_{mn} \in \{0,1\}$ denote whether a candidate node $n \in \mathbb{V}_{\text{SNC}}$ ($\mathbb{V}_{\text{SNC}}$ denotes the set of candidate smooth nodes) can be placed as a smooth node and whether a client $m \in \mathbb{V}_{\text{CLI}}$ is assigned to the smooth node n, respectively. Thus, the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ show the placement and assignment plans, respectively:

$$\boldsymbol{x} = (x_n \in \{0,1\} : n \in \mathbb{V}_{\text{SNC}}), \tag{1}$$
$$\boldsymbol{y} = (y_{mn} \in \{0,1\} : m \in \mathbb{V}_{\text{CLI}}, n \in \mathbb{V}_{\text{SNC}}). \tag{2}$$

A node n is not capable enough to be placed as a smooth node ($x_n = 0, \forall n \notin \mathbb{V}_{\text{SNC}}$). Each client needs to be assigned to a smooth node, and we require $\sum_{n\in\mathbb{V}_{\text{SNC}}} y_{mn} = 1, \forall m \in \mathbb{V}_{\text{CLI}}$. A node n must be placed as a smooth node so that client m can be assigned ($y_{mn} \le x_n, \forall m \in \mathbb{V}_{\text{CLI}}, n \in \mathbb{V}_{\text{SNC}}$).

Let ζ_{mn} and δ_{nl} denote the management cost of assigning a client $m \in \mathbb{V}_{\text{CLI}}$ to a smooth node $n \in \mathbb{V}_{\text{SNC}}$, and the synchronization cost between two smooth nodes $n, l \in \mathbb{V}_{\text{SNC}}$, respectively. Notice that ζ_{mn} and δ_{nl} are local or edge-wise parameters probed by candidate smooth nodes at the last long period. Then the total management cost and synchronization cost in the network can be expressed as

$$\mathcal{C}_M(\boldsymbol{y}) = \sum_{m\in\mathbb{V}_{\text{CLI}}} \sum_{n\in\mathbb{V}_{\text{SNC}}} \zeta_{mn} y_{mn}, \tag{3}$$
$$\mathcal{C}_S(\boldsymbol{x},\boldsymbol{y}) = \sum_{n\in\mathbb{V}_{\text{SNC}}} \sum_{l\in\mathbb{V}_{\text{SNC}}} x_n x_l (\delta_{nl} \sum_{m\in\mathbb{V}_{\text{CLI}}} y_{mn} + \epsilon_{nl}), \tag{4}$$

where ϵ_{nl} denotes the constant cost in synchronization.

The tradeoff is transformed into a balance between the costs shown in equations (3)-(4). Let $\omega \ge 0$ denote the weight value between the two costs, and the balance cost can be stated as

$$\mathcal{C}_B(\boldsymbol{x},\boldsymbol{y}) = \mathcal{C}_M(\boldsymbol{y}) + \omega \mathcal{C}_S(\boldsymbol{x},\boldsymbol{y}). \tag{5}$$

The PCH placement problem is shown as $\min \mathcal{C}_B(\boldsymbol{x},\boldsymbol{y})$, where the constraints are formulas (1)-(2). The problem is complex in that it contains discrete variables and a nonlinear objective function (4) with cubic and quadratic terms, and is a typical NP-hard problem [18].

C. Optimization Placement Problem Solutions

Small-scale optimal solution. We convert the placement problem to a MILP problem to find the small-scale optimal solution. The conversion is vital since after turning into a problem with a linear objective function with constraints, it can be solved easily by existing various commercial solvers.

We use standard linearization techniques to achieve this conversion process. First, we introduce two vectors $\boldsymbol{\vartheta}$ and $\boldsymbol{\varphi}$ as the additional optimization variables

$$\boldsymbol{\vartheta} = (\vartheta_{nl} \in \{0,1\} : n, l \in \mathbb{V}_{\text{SNC}}), \tag{6}$$
$$\boldsymbol{\varphi} = (\varphi_{nlm} \in \{0,1\} : n, l \in \mathbb{V}_{\text{SNC}}, m \in \mathbb{V}_{\text{CLI}}). \tag{7}$$

Second, the linear constraints for ϑ and φ are as follows

$$\vartheta_{nl} \le x_n, \quad \vartheta_{nl} \le x_l, \quad \vartheta_{nl} \ge x_n + x_l - 1, \quad n, l \in \mathbb{V}_{\text{SNC}}, \tag{8}$$
$$\varphi_{nlm} \le \vartheta_{nl}, \quad \varphi_{nlm} \le y_{mn}, \quad \varphi_{nlm} \ge \vartheta_{nl} + y_{mn} - 1, \quad n, l \in \mathbb{V}_{\text{SNC}}, m \in \mathbb{V}_{\text{CLI}}. \tag{9}$$

Where the constraints in (8) mean that if at least one x_n and x_l are 0, ϑ_{nl} is 0; otherwise, it is 1. Similarly, the constraints in (9) work on the same principle.

Third, we linearize the cost function (4) using the new variables, and it can be converted to

$$\widehat{\mathcal{C}}_S(\boldsymbol{\vartheta}, \boldsymbol{\varphi}) = \sum_{n \in \mathbb{V}_{\mathrm{SNC}}} \sum_{l \in \mathbb{V}_{\mathrm{SNC}}} \Big(\sum_{m \in \mathbb{V}_{\mathrm{CLI}}} \delta_{nl} \varphi_{nlm} + \epsilon_{nl} \vartheta_{nl} \Big). \quad (10)$$

Finally, the MILP can be stated as $\min \mathcal{C}_M(\boldsymbol{y}) + \omega \widehat{\mathcal{C}}_S(\boldsymbol{\vartheta}, \boldsymbol{\varphi})$, where the constraints are formulas (1)-(2) and (8)-(9).

Therefore, the PCH placement problem has been converted to a MILP problem and can be directly solved by existing commercial solvers. The various solvers usually apply a combination of the branch and bound method and the cutting-plane method, which can solve the MILP problem quite fast for the small-scale problem. However, our model involves the payments of mobile or IoT devices, and the scale of the PCNs may be enormous, leading to an extremely large MILP problem, creating a bottleneck in the solvers' computational performance. Hence we overcome this problem by proposing an approximation solution to solve the large-scale problem.

Large-scale approximation solution. Firstly, we introduce a lemma that reveals the relationship between the placement plan $\boldsymbol{x}$ and the assignment plan $\boldsymbol{y}$.

Lemma 1. *Given a placement plan $\boldsymbol{x}$, for each $m \in \mathbb{V}_{\mathrm{CLI}}, n \in \mathbb{V}_{\mathrm{SNC}}$, the optimal assignment plan $\boldsymbol{y}$ can be expressed as*

$$y_{mn} = \begin{cases} 1, \ if \ n = \underset{n' \in \mathbb{V}_{\mathrm{SNC}}: x_{n'}=1}{\arg\min} (\omega \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n'l} + \zeta_{mn'}), \\ 0, \ otherwise. \end{cases} \quad (11)$$

Proof. Assuming that there is an optimal assignment plan $\boldsymbol{y^o}$, in which the client m^o is assigned to the smooth node n^o. Then there is another node $n^h \neq n^o$ and $n^h = 1$, let

$$\omega \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^h l} + \zeta_{mn^h} < \omega \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^o l} + \zeta_{mn^o}, \quad (12)$$

which shows that if the client m^o is reassigned to the smooth node n^h, the management cost reduces $\zeta_{mn^o} - \zeta_{mn^h}$, and the synchronization cost reduces $\sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^o l} - \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^h l}$. Thus the value of objective function $\mathcal{C}_B$ reduces $\omega \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^o l} + \zeta_{mn^o} - \omega \sum_{l \in \mathbb{V}_{\mathrm{SNC}}: x_l = 1} \delta_{n^o l} - \zeta_{mn^o} > 0$. However, this contradicts our assumption that $\boldsymbol{y^o}$ is an optimal assignment plan. □

Lemma 1 indicates that it is easy to find the assignment plan $\boldsymbol{y}$ for a given placement plan $\boldsymbol{x}$, thus we concentrate on optimizing the placement plan. Let X_n represent the placement of a smooth node n (i.e., $x_n = 1$), and the set of all possible placements of the smooth node is shown as

$$\mathcal{S} = (X_n : n \in \mathbb{V}_{\mathrm{SNC}}), \quad (13)$$

which means if and only if $X_n \in \mathcal{X}$, a subset $\mathcal{X} \subseteq \mathcal{S}$ shows a placement plan $\boldsymbol{x}$ that $x_n = 1$. Let $\boldsymbol{x}_{\mathcal{X}}$ denote the binary representation of $\mathcal{X}$, thus the balance cost objective function $\mathcal{C}_B$ can be denoted as a set of function $f : 2^{\mathcal{S}} \to \mathbb{R}$

$$f(\mathcal{X}) = \mathcal{C}_B(\boldsymbol{x}_{\mathcal{X}}, y(\boldsymbol{x}_{\mathcal{X}})), \quad (14)$$

where $y(\boldsymbol{x}_{\mathcal{X}})$ indicates the optimal assignment plan given the smooth node placement plan $\boldsymbol{x}_{\mathcal{X}}$ based on equation (11).

Secondly, we consider a well-researched class of set functions known as supermodular [19].

Definition 2. *Given a finite set $\mathcal{S}$, a set function $f : 2^{\mathcal{S}} \to \mathbb{R}$ is called supermodular if for all subsets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{S}$ with $\mathcal{A} \subseteq \mathcal{B}$ and every element $i \in \mathcal{S} \setminus \mathcal{B}$ it holds that*

$$f(\mathcal{A} \cup \{i\}) - f(\mathcal{A}) \leq f(\mathcal{B} \cup \{i\}) - f(\mathcal{B}), \quad (15)$$

which states that when an element i is added to a set, the marginal value rises with the expansion of the respective set.

Lemma 2. *The set function $f(\mathcal{X})$ is supermodular for the case of uniform costs $\delta_{nl} = \delta_{n'l'} = \delta$, $\forall n, l, n', l' \in \mathbb{V}_{\mathrm{SNC}}$.*

The lemma 2 has been proved in [18]. Based on that, the placement problem can be molded as minimizing a supermodular function f.

Algorithm 1: Placement Approximation Algorithm

Input: Two initially solutions X_0^s, Y_0^s, element u_i
Output: Final solution X_z^s (or equivalently Y_z^s)

1 **for** $i = 1$ ***to*** z **do**
  // Maintain the two solutions until they coincide
2 $a_i \leftarrow f(X_{i-1}^s \cup \{u_i\}) - f(X_{i-1}^s)$
3 $b_i \leftarrow f(Y_{i-1}^s \setminus \{u_i\}) - f(Y_{i-1}^s)$
4 $a_i' \leftarrow \max\{a_i, 0\}$, $b_i' \leftarrow \max\{b_i, 0\}$
5 **if** $a_i'/(a_i' + b_i')^\star$ **then**
6 $X_i^s \leftarrow X_{i-1}^s \cup \{u_i\}$, $Y_i^s \leftarrow Y_{i-1}^s$
7 **else**
8 $X_i^s \leftarrow X_{i-1}^s$, $Y_i^s \leftarrow Y_{i-1}^s \setminus \{u_i\}$
9 **return** X_z^s (or equivalently Y_z^s)
10 * If $a_i' = b_i' = 0$, then $a_i'/(a_i' + b_i') = 1$

Thirdly, solving this kind of problem is equivalent to addressing their submodular function maximization version. Let f^{ub} denote an upper limit of the highest possible value of $f(\mathcal{X})$, the submodular function is $\widehat{f}(\mathcal{X}) = f^{ub} - f(\mathcal{X})$.

There are various approximation algorithms (e.g., [20], [21]) to maximize $\widehat{f}(\mathcal{X})$, and an approximation bound ψ indicates the ratio of the value of the approximate solution over the optimal solution value is always at least ψ. The algorithm in [21] provides the best approximation bound, which $\psi = \frac{1}{2}$. As is outlined in Alg. 1, it yields in $z = |\mathbb{V}_{\mathrm{SNC}}|$ iterations, and $u_i (1 \leq i \leq z)$ is an arbitrary element of set $\mathcal{S}$. Two solutions X^s and Y^s initially set as $X_0^s \leftarrow \varnothing$ and $Y_0^s \leftarrow \mathcal{S}$. **Lines 1-9** show as follows. At the i^{th} iteration, the algorithm adds u_i to X_{i-1}^s or removes u_i from Y_{i-1}^s randomly and greedily based on the marginal gain of each of the two options. Thus the algorithm generates two random solutions X_i^s and Y_i^s. After z iterations, both solutions coincide (i.e., $X_z^s = Y_z^s$), and it is returned in line 9. Line 10 handles a special case in line 5

where $a'_i = b'_i = 0$. Lastly, based on the above, we can get the approximate solution of the large-scale network instances.

D. Rate-Based Routing Protocol Design

The formal constraints. For a path p, let r_p represent the payment rate sent on p from the start to the end. We assume that TUs are sent through a payment channel of capacity $c_{a,b}$ from the smooth node a to another smooth node b at a rate $r_{a,b}$. Once payment is forwarded, it takes Δ time on average to receive the TUs acknowledgment from the end, thus $r_{a,b}\Delta$ funds are locked in the channel. The *capacity constraint* on the channel means that the average rate cannot exceed $c_{a,b}/\Delta$. In addition, in order to ensure the channel fund balance, there is a *balance constraint* that the one direction payment rate $r_{a,b}$ needs to match the rate $r_{b,a}$ in the reverse direction. Otherwise, the funds move to one end of the channel, and eventually, all converge at one end, creating a local deadlock (see §II-B).

To ensure the full utilization of funds in channels, we consider a common model of *utility* for making payments. The logarithm of the total rate at which payments are sent from a source represents the utility of the source [22]. Therefore, we seek to maximize the total utility of whole source-destination pairs subject to the above constraints as follows:

$$\max \quad \sum_{s,e\in\mathbb{V}} log(\sum_{p\in\mathbb{P}_{s,e}} r_p) \tag{16}$$

$$s.t. \quad \sum_{p\in\mathbb{P}_{s,e}} r_p\Delta \le d_{s,e} \quad \forall s,e\in\mathbb{V} \tag{17}$$

$$r_{a,b} + r_{b,a} \le \frac{c_{a,b}}{\Delta} \quad \forall (a,b)\in\mathbb{E} \tag{18}$$

$$|r_{a,b} - r_{b,a}| \le \epsilon \quad \forall (a,b)\in\mathbb{E} \tag{19}$$

$$r_p \ge 0 \quad \forall p\in\mathbb{P}, \tag{20}$$

where s denotes the start and e denotes the end, $\mathbb{P}_{s,e}$ is the set of all paths from s to e, $d_{s,e}$ is the demand from s to e. $c_{a,b}$ denotes the capacity of the channel (a,b), and $\mathbb{P}$ denotes the set of all paths. Formula (17) indicates the *demand constraint*, ensuring the total flow of all paths is no more than the total demand. Formula (18) and (19) are *capacity* and *balance constraints*, respectively. The balance constraint is harsh in the ideal case (i.e., the system parameter $\epsilon = 0$), but we intend that the flow rates in both channel directions tend towards equilibrium in practice (i.e., ϵ is small enough).

Distributed routing decisions. Each PCH makes distributed routing (incremental) decisions for payments based on the network data of the last epoch and its clients' requests in the current epoch. Based on primal-dual decomposition techniques [23], we consider the optimization problem for a generic utility function $U(\sum_{p\in\mathbb{P}_{s,e}} r_p)$. Lagrangian decomposition can naturally decompose this linear programming problem into separate subproblems [24]. A solution is to compute the flow rates that should be maintained on each path. We set the *routing price* in both directions of each channel, and the PCHs adjust the prices to control the flow rates of TUs. Meanwhile, the routing price is used as the *forwarding fee* to incentivize PCHs.

The routing protocol is shown in Alg. 2. **(Lines 1-9)** There is decrypted a payment demand $D_{s,e}$, the smooth node splits it into k packets of TUs d_i (we limit *Min-TU* $\le |d_i| \le$ *Max-TU* to control the number of split TUs), $|D_{s,e}| = \sum_{i=1}^{k} |d_i|$, and there are k paths $\{p_i\}_{1\le i\le k} \in \mathbb{P}_{s,e}$ (see §V-D for a discussion of choosing different paths). For brevity, we only consider a channel (a,b) in path p_i, $(a,b)\in p_i$. Let $\lambda_{a,b}$ denote the *capacity price* that indicates the total rate of arrival transactions exceeds the capacity, and let $\mu_{a,b}$ and $\mu_{b,a}$ denote the *imbalance price* that represents the imbalance of rate in the two directions, respectively. These three prices are updated every τ seconds to keep the capacity and balance constraints from being violated. Let n_a, n_b denote the funds required to maintain the flow rates at a and b. The capacity price $\lambda_{a,b}$ is updated as

$$\lambda_{a,b}(t+1) = \lambda_{a,b}(t) + \kappa(n_a(t) + n_b(t) - c_{a,b}), \tag{21}$$

where κ is a system parameter used to control the rate of price change. Any required funds excessing the capacity $c_{a,b}$ cause the capacity price $\lambda_{a,b}$ to rise, which indicates that the rates via a,b need to be reduced and vice-versa.

Let m_a, m_b represent the TUs arriving at a and b in the last period, respectively. The imbalance price $\mu_{b,a}$ is updated as

$$\mu_{a,b}(t+1) = \mu_{a,b}(t) + \eta(m_a(t) - m_b(t)), \tag{22}$$

where η is a system parameter. Any funds arriving in (a,b) direction more than (b,a) direction cause the imbalance price $\mu_{a,b}$ to increase and $\mu_{b,a}$ to decrease, which means that the rates routing along (a,b) need to be throttled and vice-versa.

Based on observations of routing prices and node feedbacks, the smooth nodes run a multi-path routing protocol to control the rates at which payments are transferred. Probes [10] are sent periodically every τ seconds on each path to measure the above two prices. The routing price of the channel (a,b) is

$$\xi_{a,b} = 2\lambda_{a,b} + \mu_{a,b} - \mu_{b,a}, \tag{23}$$

the forwarding fee that a needs to pay to b is

$$\mathsf{fee}_{a,b} = T_{\mathsf{fee}}(2\lambda_{a,b} + \mu_{a,b} - \mu_{b,a}), \tag{24}$$

where $T_{\mathsf{fee}} (0 < T_{\mathsf{fee}} < 1)$ is a systematic threshold parameter.

Thus, the total routing price of a path p is

$$\varrho_p = (1 + T_{\mathsf{fee}}) \sum_{(a,b)\in p} \xi_{a,b}, \tag{25}$$

which indicates the total amount of excess and imbalance demands. Then the smooth node sends a probe on path p, which sums the price $\xi_{a,b}$ of each channel (a,b) on p. Based on the routing price ϱ_p from the most recently received probe, the rate r_p is updated as

$$r_p(t+1) = r_p(t) + \alpha(U'(r) - \varrho_p(t)), \tag{26}$$

where α is a system parameter. Therefore, the sending rate on a path is adjusted reasonably according to the routing price.

Algorithm 2: Distributed Routing Decision Protocol

Input: Decrypted demand $D_{s,e}$, rate r_{p_i}, required funds n_a, n_b, arrived TUs m_a, m_b
Output: Routing rates r_{p_i} $(1 \le i \le k)$

1 Split the demand $D_{s,e}$ into d_i on path p_i $(1 \le i \le k)$.
2 **for** $i = 1$ ***to*** k **do**
3 **for** $\forall$ *payment channel* (a, b) *on path* p_i **do**
  // Update the routing rates
4 $\lambda_{a,b} \leftarrow \lambda_{a,b} + \kappa(n_a + n_b - c_{a,b})$
5 $\mu_{a,b} \leftarrow \mu_{a,b} + \eta(m_a - m_b)$
6 $\xi_{a,b} \leftarrow 2\lambda_{a,b} + \mu_{a,b} - \mu_{b,a}$
  // Forwarding fee
7 $\mathsf{fee}_{a,b} \leftarrow T_{\mathsf{fee}}(2\lambda_{a,b} + \mu_{a,b} - \mu_{b,a})$
  // Routing price of the path p_i
8 $\varrho_{p_i} \leftarrow (1 + T_{\mathsf{fee}}) \sum_{(a,b) \in p_i} \xi_{a,b}$
9 $r_{p_i} \leftarrow r_{p_i} + \alpha(U'(r) - \varrho_{p_i})$
  // Congestion control
10 **if** $r_{p_i} > r_{a,b}^{\text{process}}$ ***or*** $F_{a,b} < |d_i|$ **then**
11 $q_{a,b}^{\text{amount}} \leftarrow d_i$
12 $t_{p_i}^{\text{delay}} \leftarrow$ Smooth nodes monitor
13 **if** $t_{p_i}^{\text{delay}} > T$ **then**
14 $d_i^* \leftarrow d_i$
15 **if** d_i^* *is aborted* **then**
16 $w_{p_i} \leftarrow w_{p_i} - \beta$
17 **if** $q_{a,b}^{\text{amount}} < w_{p_i}$ ***and*** d_i *is transmitted* **then**
18 $w_{p_i} \leftarrow w_{p_i} + \frac{\gamma}{\sum_{p_i' \in \mathbb{P}_{s,e}} w_{p_i'}}$
19 **return** r_{p_i}

Congestion control (Lines 10-18). We consider that this rate-based approach may cause congestion of TUs, so we use the waiting queue and window to control congestion. Whenever congestion occurs, intermediate hubs in the path queue up the TUs, representing a capacity or balance constraint violation. So the smooth nodes need to use a congestion control protocol to detect capacity and imbalance violations to control queues by adjusting the sending rates in channels.

The congestion controller has two basic properties to achieve both efficiency and balanced rates. (i) It should try to keep the queue not empty, which indicates that the channel capacity is being used efficiently. (ii) It should keep the queues bounded, which means that the flow rate of each path can not exceed capacity or be imbalanced. There are some congestion control algorithms [25] that satisfy the two properties and can be adapted for PCNs. Now we describe the protocol briefly:

If the rate r_{p_i} exceeds the upper limited rate $r_{a,b}^{\text{process}}$ that the channel (a, b) can process, or the demand $|d_i|$ exceeds the current funds in $(a \to b)$ direction $F_{a,b}$, then the protocol goes to the congestion control part. ***(i)*** Let $q_{a,b}^{\text{amount}}$ denote the amount of TUs pending in queue $q_{a,b}$. The queuing delay t_p^{delay} on path p is monitored by smooth nodes, and if it exceeds the pre-determined threshold T, the packet d_i is marked as d_i^*. Once a TU is already marked, hubs do not process the packet and merely forward it. When the recipient sends back an acknowledgment with the marked field appropriately set, hubs forward it back to the sender. ***(ii)*** Based on the observations of congestion in the network, smooth nodes control the payments rates transferred in the channels and choose a set of k paths to route TUs from s to e. ***(iii)*** The window size w_p represents the maximum of unfinished TUs on path p. The smooth nodes maintain the window size for every candidate path to a destination, which indirectly controls the flow rate of TUs on the path. The smooth nodes keep track of unserved or aborted TUs on the paths. New TUs can be transmitted on path p only if the total number of TUs to be processed does not exceed w_p. On a path p from s to e, the window is adjusted as

$$w_p\,(t+1) = w_p\,(t) - \beta, \tag{27}$$

$$w_p\,(t+1) = w_p\,(t) + \frac{\gamma}{\sum_{p' \in \mathbb{P}_{s,e}} w_{p'}}, \tag{28}$$

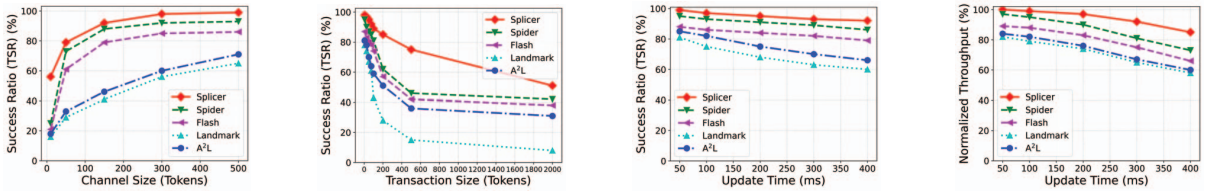
where equation (27) means the marked packets fail to complete the payment within the deadline, and the senders choose to cancel the payment, and equation (28) means the unmarked packets are transmitted. The positive constants β and γ denote the factors that the window size decreases and increases.

Notice that although Spider [9] uses a similar multi-path payment model, the main *differences* of Splicer are: (i) Splicer considers forwarding costs, though the fee model is different from that of the Lightning Network in Spider. (ii) Besides congestion control, Splicer also provides rate control as to minimize the capacity and imbalance violations in the network. (iii) Splicer's route computation is outsourced to PCHs instead of being processed by end-users. In the next section, we evaluate the performance of such an optimized Splicer vs. Spider solution.

V. Performance Evaluation

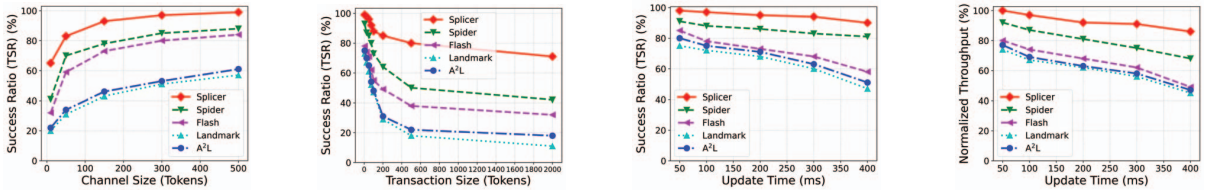
A. Experiment Setup

Our evaluation consists of a simulation using MATLAB and full implementation of the Lightning Network Daemon (LND) testnet. We model the PCN in two scales, a small-scale network (100 nodes) and a large-scale network (3000 nodes). Our modified LND is deployed on the machine with a six-core i7-9750H processor working at 2.6 GHz, 32 GB of RAM, 500 GB of SSD, and a 10 Gbps network interface. Referring to Spider's evaluation benchmark, the channel connections between nodes are generated by ROLL [26] based on the Watts-Strogatz small-world model. Following the heavy-tailed distribution of the real-world dataset on the lightning channel size [27], funds are set on each side of the channels. The directional distribution of each transaction is generated on our processed Lightning Network real-world dataset, and the transaction value is generated in the same credit card dataset [28] adopted by Spider. Notice that we have confirmed that these transactions are guaranteed to cause some local deadlocks and contain large-value transactions that the Lightning Network cannot handle.



(a) Influence of the channel size (b) Influence of the transaction size (c) Influence of the update time (d) Normalized throughput

Fig. 7. The comparison between Splicer and other schemes under different metrics in small-scale networks.



(a) Influence of the channel size (b) Influence of the transaction size (c) Influence of the update time (d) Normalized throughput

Fig. 8. The comparison between Splicer and other schemes under different metrics in large-scale networks.

Parameter settings. The minimum, average, and median channel sizes are 10, 403, and 152 tokens. The transaction timeout is 3 seconds, the *Min-TU* is 1 token, and the *Max-TU* is 4 token. The number k of multiple paths is 5. We set management cost $\zeta_{mn} = 0.02 \cdot hops_{mn}$, and the synchronization cost $\delta_{nl} = 0.01 \cdot hops_{nl}$, $\epsilon_{nl} = 0.05 \cdot hops_{nl}$, where $hops$ represents the number of hops in the communication path between nodes. In congestion control, we set the queue size of each channel as 8000 tokens. The factors of the window size β and γ are 10 and 0.1, respectively. The update time $\tau = 200$ ms. The threshold T of delay in the queue is 400 ms.

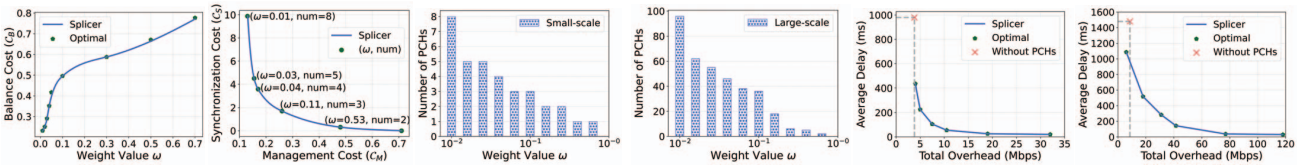
B. Performance of Splicer

We study the performance of Splicer under different metrics compared with different schemes. As shown in Fig. 7 and Fig. 8, Splicer consistently outperforms other schemes in small and large network scales. **Spider** [9] is a multi-path source routing scheme in which each sender decides the routes. **Flash** [10] is also based on source routing, using a modified max-flow algorithm to find paths for large payments, and routing small payments randomly through precomputed paths. **Landmark** routing is adopted in many prior PCN routing schemes [6], [29], [30]. Each sender computes the shortest path to the well-connected landmark nodes, and then the landmark nodes route to the destination in k distinct shortest paths. The **A^{2}L** [4] is the state-of-the-art PCH that focuses on providing unlinkability. The results are as follows:

Transaction success ratio (TSR) means the number of completed transactions over the number of generated transactions. A high value of TSR indicates the stability of the model, i.e., the ability to handle transaction deadlocks and balance the network load. Fig. 7(a) and 8(a) show the TSR of Splicer is an average of 53.4% higher than the other four schemes. Combining the Fig. 7(b) and 8(b), there is also a considerable increase (49.1%) in the TSR as the transaction size varies. These results demonstrate that the PCHs distributed routing decision protocol can improve the TSR. We note that the improvement of Spider is more obvious under the large-scale networks. This feature benefits hubs' deployment because hubs perform many routes, have larger capital, and thus may have a larger channel size. Fig. 7(c) and 8(c) show the TSR under the influence of update time τ in different schemes. The results show that the TSR of Splicer is stable above 90% with the increase of update time, which is slightly higher (5% and 10.5%, respectively) than Spider. Because Spider also adopts a multi-path routing strategy, reducing the possibility of deadlocks, the TSR is high when the channel size is appropriate. In contrast, the TSR of A^{2}L decreased significantly, and Splicer increased by 26% and 39%, respectively. However, Spider handles source routing computations at the end-users, which is limited by the performance of a single machine, so the TSR is lower than Splicer, especially with large-scale networks. Because A^{2}L's complex cryptographic primitives reduce scalability, the overall average improvement in TSR for Splicer is 42%, which proves that the network funds flow smoothly, almost without deadlock.

Normalized throughput is the total value of payments completed over the total value generated, normalized by the maximum throughput. A high throughput demonstrates the model's ability for massive concurrent transactions and further corroborates TSR to prove the stability of the model. Fig. 7(d) and 8(d) show the normalized throughput of Splicer is an average of 29.3% higher than the other four schemes. Splicer's throughput improvement is more significant in large-scale networks (average 37.7%). Compared to Spider, the normalized throughput of Splicer is 8.5% and 15.6% higher on average, respectively. Compared with A^{2}L, the improvement of Splicer is more prominent, which are 28.2% and 48.4%, respectively. As the increased update time, more and more transactions are getting closer to the deadline, so the probability of transaction failure is higher. Because A^{2}L lacks a scalable routing strategy design, it is more affected by this factor. Therefore, considering the TSR and throughput, we choose the median of 200 ms as the update time for Splicer.

The above results show that Splicer can significantly improve performance scalability compared to state-of-the-arts. In large-scale networks, the performance improvement effect of Splicer is more significant. Additionally, Splicer's placement optimization makes communication costs smaller (see §V-C). Thus in



(a) Balance cost (b) Trade-off in costs (c) Small-scale networks (d) Large-scale networks (e) Small-scale costs (f) Large-scale costs

Fig. 9. Evaluation of smooth node placement.

TABLE II
THE INFLUENCE OF THE DIFFERENT CHOICES IN ROUTING FOR SPLICER.

Scale	Path Type				Path Number				Scheduling Algorithm			
	KSP	Heuristic	EDW	EDS	1	3	5	7	FIFO	LIFO	SPF	EDF
Small	65.20%	77.02%	85.29%	83.26%	32.31%	70.74%	86.13%	82.88%	53.81%	90.35%	76.18%	70.44%
Large	58.85%	76.19%	90.07%	88.31%	35.53%	65.45%	90.40%	87.72%	61.19%	93.23%	82.21%	78.26%

large-scale low-power scenarios, we recommend Splicer.

C. Evaluation of Smooth Node Placement

We evaluate the placement of smooth nodes as in Fig. 9.

Efficiency tradeoff. Fig. 9(a) and 9(b) show the influence of weight value on the costs in the small-scale network. Fig. 9(a) shows that running the PCHs' average balance cost varies with the weight value of ω proposed in §IV-B. Overall, the performance of our model is close to the optimal for almost all values of ω. This indicates that our model successfully simulates the relationship between the two communication costs of the network. Fig. 9(b) further shows the tradeoff between the two costs. The annotation for the nodes in the figure is the corresponding weight ω and the *number* of smooth nodes (e.g. 4 smooth nodes for $\omega = 0.04$). Management costs are incurred between smooth nodes and clients, and synchronization costs are only incurred between smooth nodes. PCNs have different affordability for these two costs. For example, because of the strong computational capability of the PCHs, PCNs can bear a high cost of synchronization. Clients may be IoT nodes so that PCNs can carry less management cost. Therefore, Splicer can adjust both costs by increasing or decreasing the number of smooth nodes in the voting smart contract suitably based on the results. In addition, the influence curves of weight value on costs in the large-scale network are similar to those in the small-scale network. The difference is that large-scale networks require more smooth nodes than small-scale networks. Fig. 9(c) and 9(d) show the number of smooth nodes for different weight values ω in small and large network scales. When management cost is preferred, Splicer deploys more smooth nodes, reducing the communication overhead and latency of the smooth nodes managing the clients and vice-versa.

PCH placement effectiveness. Fig. 9(e) and 9(f) show the average transaction delay and total traffic overhead with and without PCHs (i.e., comparing distributed routing and source routing decisions) to demonstrate the effectiveness of smooth nodes. We depict the delay-overhead curves by iterating the weight values in small and large-scale networks. If without smooth nodes, the average delay and traffic overhead are fixed. With similar total overhead, the average delay of Splicer is significantly lower than without smooth nodes. Overall, Splicer achieves 80.9% lower latency than schemes without PCHs (e.g., Spider). Appropriate placement of some PCHs can reduce the total overhead of the network. Besides, Splicer can tolerate more traffic overhead to reduce transaction latency further.

D. Routing Choices in Splicer

In addition, we study the influence of the different choices in routing on the TSR, as shown in Table II.

Path type. We evaluate the performance by choosing different types of routing paths for each source-destination pair in two network scales. **KSP** means the k-shortest paths. **Heuristic** method picks 5 feasible paths with the highest channel funds. **EDW** represents the edge-disjoint widest paths. **EDS** means the edge-disjoint shortest paths. The results show that EDW outperforms other approaches in two network scales. Due to the channel size following the heavy-tailed distribution, the widest paths can better utilize the network's capacity.

Path number. We evaluate the performance in the different numbers of EDW paths. The TSR increases as the number of paths increases, suggesting that more paths utilize the network's capacity better. It is noted that the TSR declines slightly as the number of paths increases to 7, due to the high computational complexity that causes the performance bottleneck. Therefore, we choose 5 routing paths in Splicer.

Scheduling algorithm. We change the scheduling methods for the waiting queue. There are four kinds of methods: first in first out (**FIFO**); last in first out (**LIFO**); smallest payments first (**SPF**); and earliest deadline first (**EDF**). The results show that LIFO represents 10-40% higher than other methods since it first processes the transactions far from the deadline. FIFO and EDF process transactions closest to their deadlines, resulting in poor transaction performance due to more failures. Though SPF shows the second well performance, the large transactions pile up and take up a lot of channel funds, leading to a lower transaction success ratio.

VI. RELATED WORK

Payment channel hubs: TumbleBit [3] presents a cryptographic protocol for the PCHs, which maintains multi-channels to reduce the routing complexity and make the transactions *unlinkable* (i.e., the hubs do not know the two parties of transactions). But TumbleBit relies on scripting-based functionality, and the communication complexity increases linearly with the security parameter. A^2L [4] proposes a novel cryptographic primitive to improve TumbleBit, which provides better backward compatibility and efficiency. Commit-chains

[12] process off-chain transactions in a pattern similar to PCHs: a centralized operator maintains a service for multiple users. It provides a tradeoff between channel establishment cost, user churn, collateral management, and decentralization. Perun [11] proposes a smart contract-based method to build virtual PCHs to reduce communication complexity but at the cost of losing unlinkability. However, they do not consider the placement of the PCH. The PCH placement problem can significantly affect the transaction latency and communication overhead of PCNs.

Layer-2 source routing: Flare [6] describes a hybrid routing algorithm, which seeks to optimize the average time of finding a payment route. Revive [8] proposes the first rebalancing scheme for PCNs, which rebalances the channels where node funds are unbalanced. But the routing algorithm proposed by Revive relies on a trusted third party, which is vulnerable to the single point of attack. Sprites [7] tries to reduce the worst-case "collateral cost" of an off-chain linked transaction. Spider [9] presents a multi-path routing scheme based on "packetization", which can achieve high-throughput routing in the PCNs. However, the sender computes the routing path to the recipient. In a large-scale PCN, the performance requirements of the sender can be quite demanding. In addition, there are other new layer-2 scaling solutions, such as Rollups [31] proposed on Ethereum, that increase throughput by batching transactions at the expense of high latency. However, it is outside the scope of our discussion on PCN architecture.

VII. Conclusion

We propose a distributed routing mechanism with high scalability based on multiple PCHs, to seek a new tradeoff between decentralization and scalability for PCNs. PCHs route transaction flows in PCNs in an optimal deadlock-free manner. We formulate the PCH placement problem for network scalability and propose two solutions in small and large-scale networks. To improve performance scalability, we design the rate-based routing and congestion control protocol on PCHs. Extensive experimental results show that Splicer outperforms the state-of-the-arts.

Acknowledgments

This work was supported in part by National Key R&D Program of China (No. 2020YFB1005500); the National Natural Science Foundation of China (No. 61972310, 61972017, 62072487, 61941114); the Beijing Natural Science Foundation (No. M21036); the Populus Euphratica Found, China (No. CCFHuaweiBC2021008); the Innovation Fund of Xidian University, China (No. YJSJ23005).

References

[1] The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. [Online]. Available: https://lightning.network/lightning-network-paper.pdf
[2] Raiden network. [Online]. Available: https://raiden.network
[3] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub." in *NDSS*, 2017.
[4] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1834–1851.
[5] EOSIO Blockchain. [Online]. Available: https://eos.io/
[6] M. S. A. O. Pavel Prihodko, Slava Zhigulin and O. Osuntokun, "Flare: An approach to routing in lightning network," 2016.
[7] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning." in *Financial Cryptography*, vol. 11598, 2019, pp. 508–526.
[8] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017, p. 439–453.
[9] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *NSDI*, 2020.
[10] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks." in *CoNEXT*. ACM, 2019, pp. 370–381.
[11] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 106–123.
[12] R. Khalil, A. Zamyatin, G. Felley, P. Moreno-Sanchez, and A. Gervais, "Commit-chains: Secure, scalable off-chain payments," Cryptology ePrint Archive, Report 2018/642, 2018.
[13] Lightning Network Daemon. [Online]. Available: https://github.com/lightningnetwork/lnd
[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems." in *EUROCRYPT*, vol. 1592, 1999, pp. 295–310.
[15] P. Li, T. Miyazaki, and W. Zhou, "Secure balance planning of off-blockchain payment channel networks." in *INFOCOM*. IEEE, 2020, pp. 1728–1737.
[16] Z.-L. Ge, Y. Zhang, Y. Long, and D. Gu, "Shaduf: Non-cycle payment channel rebalancing," in *NDSS*, 2022.
[17] L. E. Celis, L. Huang, and N. K. Vishnoi, "Multiwinner voting with fairness constraints." in *IJCAI*, 2018, pp. 144–151.
[18] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN Controller Placement at the Edge: Optimizing Delay and Overheads." in *INFOCOM*. IEEE, 2018, pp. 684–692.
[19] V. P. Il'ev, "An approximation guarantee of the greedy descent algorithm for minimizing a supermodular set function." *Discret. Appl. Math.*, vol. 114, no. 1-3, pp. 131–146, 2001.
[20] M. Feldman, J. Naor, and R. Schwartz, "Nonmonotone submodular maximization via a structural continuous greedy algorithm - (extended abstract)." in *ICALP (1)*, vol. 6755, 2011, pp. 342–353.
[21] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, "A tight linear time (1/2)-approximation for unconstrained submodular maximization." *SIAM J. Comput.*, vol. 44, no. 5, pp. 1384–1402, 2015.
[22] F. P. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control." *Computer Communication Review*, vol. 35, no. 2, pp. 5–12, 2005.
[23] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, 2005.
[24] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.
[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.
[26] A. Hadian, S. Nobari, B. Minaei-Bidgoli, and Q. Qu, "ROLL: Fast In-Memory Generation of Gigantic Scale-free Networks." in *SIGMOD Conference*, 2016, pp. 1829–1842.
[27] S. Tikhomirov, P. Moreno-Sanchez, and M. Maffei, "A quantitative analysis of security, anonymity and scalability for the lightning network." in *EuroS&P Workshops*, 2020, pp. 387–396.
[28] Credit Card Fraud Detection. [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud
[29] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *NDSS*, 2017.
[30] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions." in *NDSS*, 2018.
[31] Optimistic Rollups. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/