RCS: A High-Success-Rate and Privacy-Preserving Payment Channel Network Routing Protocol

Wenjing Li, Chen Tian, Yuan Zhang, Sheng Zhong
Nanjing University, China
Email: liwenjing@smail.nju.edu.cn, tianchen@nju.edu.cn, zhangyuan@nju.edu.cn, zhongsheng@nju.edu.cn

***Abstract*—Payment channel networks (PCNs) offer a crucial solution to the scalability challenges of blockchain-based transaction systems. However, most existing PCN routing protocols employ a "guess-and-check" approach, which undermines their transaction success rate and efficiency. In this paper, we propose a routing protocol named RCS, based on a novel "Refined Confirm-and-Send" approach. Utilizing PCN topology statistics, RCS performs a refined probing of possible transaction paths and verifies whether a path has sufficient available balance before executing the transaction through it. This method effectively improves the transaction success rate while maintaining restrained overhead. Additionally, to address users' privacy concerns, we design a privacy-preserving version of RCS, named RCS+. RCS+ uses secure comparisons to identify paths with sufficient funds without disclosing channel balances or transaction amounts. Extensive simulations with real-world and synthetic datasets demonstrate that RCS and RCS+ outperform existing state-of-the-art protocols. RCS and RCS+ achieve a 10% higher transaction success rate compared to the Shortest Path approach, which serves as the core of Lightning Network's current routing mechanism. In terms of overhead, RCS maintains the lowest cost among all tested protocols, e.g., only 20% of the Flash protocol. While RCS+ incurs marginally higher overhead due to its enhanced privacy guarantees, its cost remains just 30% of Flash's overhead. Furthermore, RCS/RCS+ exhibits robust adaptability to dynamic changes in PCN topologies, ensuring scalability as the network evolves.**

***Index Terms*—Payment Channel Network, Routing Protocol, Secure Comparison, Privacy**

I. Introduction

The payment channel networks (PCN), e.g., the Lightning Network [1] for Bitcoin and the Raiden Network [2] for Ethereum, is an important solution to address the notorious scalability problem of blockchain-based cryptocurrency systems [3–5].

A PCN consists of a network of payment channels, each established by two users who deposit initial funds into a jointly managed address on the blockchain. Subsequent transactions between the two users are only "recorded" off the blockchain by locally updating their channel balances. On-chain operations are required only when closing the channel, at which point the final balances are globally broadcast and written to the blockchain. By avoiding recording every transaction on-chain, the PCN minimizes the costly on-chain operations, thereby can significantly reduce the burden on the blockchain and improve the blockchain transaction system's overall scalability and efficiency.

In PCNs, the maximum transaction amount between two users sharing a direct channel is constrained by the channel's current balance. For users without a direct channel, transactions must be routed through a multi-hop path of interconnected channels, each requiring sufficient liquidity to forward the payment. However, identifying such a valid path presents a significant challenge, as intermediate channel balances, which are dynamically updated and not publicly observable, are controlled by third-party nodes or users.

At present, most PCN routing protocols [6–13] employ a "guess-and-check" strategy. The sender (or paying user) uses the PCN's topology and channels' initial balances (which were broadcast and recorded on the blockchain during channel establishments) to estimate potential paths to the receiver (or payee user), and randomly selects one to attempt the transaction. However, because the balances of channels along the path constantly change, there is a substantial risk that the selected path could fail due to insufficient funds at the time of the transaction, resulting in a low transaction success rate. Moreover, when a transaction fails, the Hash Time-Locked Contract (HTLC) protocol's security mechanism temporarily locks the funds along the path until a timeout expires [14]. Although the sender can retry with alternative paths after funds are released, this process introduces additional delays, further degrading the overall efficiency of PCN transactions.

For instance, the LND routing protocol, currently deployed in the Lightning Network, adopts the "guess-and-check" strategy and attempts transactions along the shortest path [15]. If the balance of any channel along the shortest path is insufficient, the transaction fails. The sender then deletes these channels and recalculates the shortest path. This process is repeated until the transaction succeeds or a preset timeout event happens. Empirical studies show that LND has a low transaction success rate [11]. Each failed transaction triggers the HTLC protocol, which locks the associated funds along the path for an extended period, usually up to 6 hours or more [15].

A number of routing protocols [16–19] adopt a different strategy called "confirm-and-send", which determines the path that has sufficient funds before using it for transactions. This approach prevents transaction failures caused by insufficient channel balances, thereby avoiding fund lock-ups along the path and improving the transaction success rate. However, verifying fund availability on each potential path introduces substantial overhead, severely impacting efficiency. Achieving a high success rate with low overhead is the core challenge for routing protocols adopting the "confirm-and-send" strategy.

In this paper, we propose a novel confirm-and-send strategy called "refined confirm-and-send". Calculating the maximum transaction amount for all paths potential for each transaction is often impractical. "Refined confirm-and-send" also verifies balance sufficiency on the path before tempting the transaction, but does not run verifications for all potential paths. Instead, it selects paths based on network topology statistics. Based on this "refined confirm-and-send", we design the RCS routing protocol, which achieves a high transaction success rate in PCNs with acceptable overhead.

Specifically, RCS calculates the maximum transaction amount only for short paths whose length does not exceed k, where k is determined based on the PCN's topology and can be adaptively tuned. By restricting path length, RCS significantly reduces verification overhead while incurring only a marginal decrease in success rate. This trade-off is justified by the fact that longer paths, while offering more candidate routes, generally exhibit lower success rates due to the bucket principle—the well-known phenomenon where the maximum transaction amount along a path is constrained by the channel with the smallest balance. To further optimize efficiency, RCS lets each sender maintain a routing table. When a transaction is initiated, if the receiver does not exist in the routing table, the sender probes to find all paths with a length l ($l \leq k$). These paths are then recorded in the routing table to avoid re-searching them in future transactions between the same parties, thus saving time. If the paths between the sender and receiver exist in the routing table, the sender can randomly select one and collaborate with the users along that path to calculate the maximum transferable amount. This process continues until a path with sufficient funds is found to complete the transaction. In the rare event that no path exists between the sender and receiver with a length not exceeding k or there are insufficient funds in the all paths of length not exceeding k, RCS can temporarily adjust the value of k to find feasible paths.

When all intermediate nodes are willing to share plaintext comparisons of transaction amounts and channel balances, RCS can efficiently determines paths and improves success rates. However, RCS has potential privacy risks, such as repeated queries that may reveal balance information. To address this, we propose RCS+, a privacy-preserving version that protects user privacy while identifying valid paths.

In summary, we make the following contributions:

- We study the topology of real-world PCNs to understand the path topology of cryptocurrency transactions.
- We design RCS, a novel routing protocol based on the "refined confirm-and-send" strategy , which improves the transaction success rate and network throughput while keeping overhead small.
- We design a secure path funds verification protocol RCS+, a privacy-preserving version of RCS, effectively protecting transaction amount privacy and available channel privacy in PCNs.
- We extend the simulator and implement RCS and RCS+. Extensive simulations are conducted to evaluate their performance compared with state-of-the-art routing protocols. The results show that RCS and RCS+ achieve a transaction success rate 10% higher than that of the Shortest Path approach, the key algorithm in Lightning's currently deployed routing protocol. In terms of overhead, RCS consistently incurs the lowest cost, only 20% of Flash's overhead—the highest among all protocols. Although RCS+ has slightly higher overhead than RCS, it still maintains a low cost at only 30% of Flash's overhead.
- We simulate network evolution to evaluate the adaptability of RCS and RCS+, demonstrating their effectiveness in dynamic PCN environments.

The rest of the paper is organized as follows: Section II provides background on PCN; Section III describes the RCS routing protocol and RCS+ is presented in Section IV; Section V details the experiments and performance evaluation; Section VI reviews related work; and Section VII concludes this paper.

II. Background

A. Payment Channel Network

The operation of opening a channel is performed on-chain. To open a payment channel, two parties commit funds into a 2-of-2 multisignature address, holding it in custody for a fixed period [14]. The payment channel network is composed of multiple such channels.

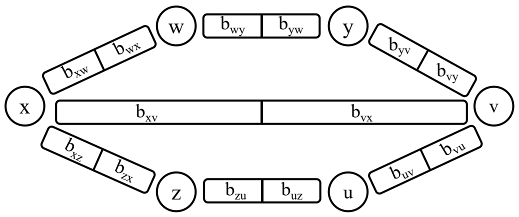


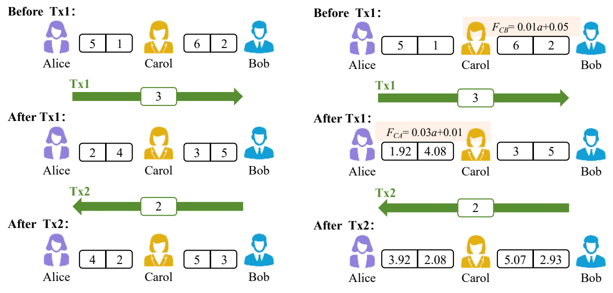
Fig. 1: Example diagram of payment channel network.

The payment channel network is represented as a directed weighted graph $G = (V, E)$ [20], where V represents the set of network users and E denotes the set of payment channels. Each edge is bidirectional, with each direction having a weight corresponding to the user's balance in this channel. These weights are updated dynamically as transactions occur, reflecting changes in the channel balances. For example, u and v are connected by the edge $e = (u, v)$. The weight b_{uv} denotes the balance of u in the channel C_{uv} (Fig. 1). Similarly, b_{vu} represents the balance of v in C_{uv}. The channel capacity Cap of each edge is the total balance in both directions of the channel, given by $Cap_{uv} = b_{uv} + b_{vu}$. The channel capacity is fixed when the channel is established and recorded on the blockchain, making it available to all network users.

Existing channels can also be closed through an on-chain process. Once both parties resolve any outstanding disputes and determine the allocation of funds, they can withdraw their funds from the channel.

B. Payment Path

When a transaction occurs between two users, a path with sufficient funds is required to complete it. It is impractical to establish a direct channel between every pair of users. Typically, users select an appropriate path that forwards payments through intermediate nodes. As the transaction progresses, the balance in the channels along the transaction path changes. For example, Alice pays 3 satoshis to Bob through the Alice-Carol-Bob path (Fig. 2(a)). To facilitate a clear analysis of how the balances in each channel change during the transaction, we temporarily disregard the fees charged by intermediate nodes (a detailed fee model will be discussed in Section II-C). In this case, two transactions occur: Alice to Carol and Carol to Bob. The HTLC protocol [14] ensures the atomicity and security of transactions: Carol receives funds from Alice if and only if Bob successfully receives the funds from Carol. Additionally, Bob can pay Alice by the Bob-Carol-Alice path.



(a) Without the fee model (b) With the fee model

Fig. 2: Transactions conduction with intermediate nodes.

C. Payment Fees

In PCN, the fee model for intermediate nodes involves two types of fees: the base fee F_{base} and the fee rate r [21]. The base fee is a fixed charge applied by intermediate nodes for facilitating a transaction. And the fee rate specifies the fee charged for each unit of currency transferred. The fee model is described as follows:

$$F = F_{base} + r \times a \tag{1}$$

where a represents the transaction amount flowing through the intermediate node.

Each payment channel has two distinct fee functions autonomously pre-configured by the two users of the channel. Fees are charged when funds flow into a channel from an intermediary node, and only one fee function is applied based on the transaction direction. As shown in Fig. 2(b), Alice and Bob transact via the Alice-Carol-Bob path. In transaction Tx1, Alice pays 3 satoshis to Bob, and the fee function $F_{CB} = 0.01a + 0.05$ of Carol in the Carol-Bob channel is applied, resulting in a fee of 0.08 satoshis. Alice pays this fee and sends 3.08 satoshis to complete the transaction. Similarly, in Tx2, Carol's fee function $F_{CA} = 0.03a+0.01$ in the Carol-Alice channel is applied.

It is crucial to highlight that the parameter a in the fee function refers to the amount ultimately received by the receiver, rather than the initial amount sent by the sender (which includes the fees of all intermediate nodes). Our protocol follows this design paradigm for fee calculation.

III. Routing Design

In this section, we describe the design of the RCS protocol in detail. First, we discuss the core foundation of the RCS protocol: the transaction path topology. Next, we explain the path finding and transaction path selection in RCS. Path finding consists of two stages: signal transmission and signal return. During signal transmission, the transaction sender uses Algorithm 1 to generate signals and send them to neighbors, which then update and forward the signal by Algorithm 2. During signal return, each node uses Algorithm 3 to process and forward the returned signal. Once the sender receives all signals and reconstructs the path, it collaborates with the intermediate nodes along the path to calculate the maximum transferable funds and finally select the transaction path.

A. Path Topology

We believe that the topological characteristics of PCNs can inspire the design of routing protocols, thereby improving the transaction success rate while reducing overhead.

The Lightning Network is a prominent application of PCNs, characterized by a small number of high-degree nodes that a large number of low-degree nodes tend to connect to. The study in [21] analyzes the topological characteristics of the current Lightning Network and finds that the network's diameter is 12 and the average distance between nodes is only 3.52, where the distance is defined as the shortest path between two nodes. Additionally, reference [21] captures snapshots of the Lightning Network over two years, exploring the evolution of the topological characteristics. Despite the expansion of the network, the average distance remains relatively stable. The study in [22] investigates transaction characteristics in the Lightning Network thoroughly by a simulator, finding that over 90% of successful transaction paths have a length of 4 or less. Similar phenomena are also observed in other PCNs, such as Ripple, where the lengths of successful transaction paths are mostly in single digits and are less than a fixed value.

Based on these topology studies, we observe that most transactions in PCNs are successfully completed through very short paths. In our routing protocol design, the length of paths found by RCS does not exceed k and k is determined based on the parameter l_{max}. In most cases $k = l_{max}$, and only in rare cases does it need to be temporarily adjusted. Our experiments show that when $l_{max} = 4$, RCS's performance surpasses that of other leading protocols (please refer to Section V-B for details). This path finding method not only reduces the number of paths to lower probing overhead but also ensures a high transaction success rate.

B. Path Finding

For the deployed PCN, information about recent successful transactions can be collected to determine l_{max}. When the RCS protocol is applied to a brand new PCN, it is necessary to first simulate transactions within the network, and the data from these simulations can then be used to determine l_{max}. The value of l_{max} is broadcast to all nodes, and each node in the network sets k to this value. In special cases, k for some transactions may be temporarily adjusted, and once the path finding for that transaction is completed, k will be reset to l_{max}. This rare special case is described in detail in III-B. The detailed protocol is as follows.

Each node maintains a routing table that records the paths between itself as the sender and different receivers, with the path length not exceeding k. If a receiver is not in the routing table, RCS uses its probing-based path finding algorithm to find paths and adds them to the routing table. Since payments between the same sender and receiver are frequent [10], the routing table can eliminate the frequent execution of the probe-based path finding algorithm, thus reducing overhead. The unique transaction path topology of PCNs ensures that k remains small, preventing the routing table of each node from becoming excessively large. Moreover, considering that changes occur in PCNs, such as nodes disappearing and new nodes joining, each node periodically clears its routing table to update the latest paths. But this update is infrequent.

The probing-based path finding algorithm consists of two phases: signal transmission and signal return.

- **Signal transmission:** the sender generates signals called Tosignal and transmits them to its neighbors. Then these neighbors update and forward the signal to find a path to the receiver with a length no greater than k.
- **Signal return:** this phase involves two types of signals: Backsignal and Nosignal. Backsignal is generated by the receiver based on the received Tosignal and is returned to the sender along the original path. Nosignal is used to inform the sender of a path finding failure.

Algorithm 1 Tosignal generation

```
Input: sender $s$, receiver $t$, s.neighbors $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$
1: $base$ = random($\mathbb{Z}_p$)
2: for $\sigma_i$ in s.neighbors do
3:     create a new Tosignal $\rho_i$
4:     $\rho_i.id = base + i$, $\rho_i.h = 0$, $\rho_i.s = s$, $\rho_i.t = t$
5:     for $j = 0$ to $k$ do
6:         $\rho_i.F[j].\xi$ = NULL, $\rho_i.F[j].b = 0$, $\rho_i.F[j].r = 0$
7:     end for
8:     $\rho_i.flag = 0$
9:     send $\rho_i$ to the neighbor $\sigma_i$
10: end for
```

Although the Tosignal, Backsignal and Nosignal are different names to distinguish signals between the transmission and return processes, their fundamental form is identical, which can be described as follows:

$$\rho = (id, h, s, t, F[], flag)$$

The variable id marks different paths, while h is the number of intermediate nodes traversed by the sender s to reach the receiver t. Although PCN requires nodes to disclose the fee functions of their channels, allowing senders to query them at any time, nodes may adjust the fee functions dynamically in practice [22]. This real-time modification of the fee functions may prevent the nodes from querying the correct fee functions of the intermediate nodes in a timely manner. Therefore, we use the array $F[]$ to store the fee functions of the intermediate nodes along the path. Each element F[i] corresponds to the fee function of the i-th intermediate node along the path probed by ρ, containing three parameters: the node name ξ, the base fee b and the fee rate r of its fee function. The value of $flag$ can be 0, 1 or -1, representing different signals. During the signal transmission, the $flag$ of Tosignal is 0. In the signal return, the $flag$ of Backsignal is 1 which indicates that a valid path has been found, and the $flag$ of Nosignal is -1, indicating no valid path in the probing direction.

Signal transmission: The sender executes Algorithm 1 to generate and initialize Tosignal. It first calculates the number of neighbors n based on its payment channels and records the neighbor set as $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$. Subsequently, the sender generates n Tosignal $\rho_1, \rho_2, \ldots \rho_n$, initializing each Tosignal ρ_i. Once the Tosignal ρ_i is successfully initialized, the sender dispatches ρ_i to the corresponding neighbor σ_i.

Algorithm 2 Tosignal update and forwarding

```
Input: Tosignal $\rho$, current node $\alpha$, $\alpha$.neighbors $\{\sigma_1, \ldots, \sigma_m\}$
1: if $\alpha == \rho.t$ then
2:     generate a Backsignal $\rho_{back} = \rho$, $\rho_{back}.flag = 1$
3:     return $\rho_{back}$
4: end if
5: $\rho.h$ ++, visitnodes = $\{\rho.s\}$
6: for $i = 0$ to $\rho.h - 1$ do
7:     visitnodes = visitnodes $\cup$ $\{\rho.F[i].\xi\}$
8: end for
9: $size$ = $|\alpha$.neighbors $-$ visitnodes$|$
10: if $\rho.h \geq k$ or $size == 0$ then
11:     generate a Nosignal $\rho_{no} = \rho$, $\rho_{no}.flag = -1$
12:     return $\rho_{no}$
13: else
14:     $\rho.F[\rho.h].\xi$ = the name of $\alpha$
15:     $\rho.F[\rho.h].b$ = the base fee of $\alpha$
16:     $\rho.F[\rho.h].r$ = the fee rate of $\alpha$
17:     for $\sigma_j$ in $\alpha$.neighbors $-$ visitnodes do
18:         $\rho_j = \rho$, $\rho_j.id = \rho.id$ + '/' + random($size$)
19:         send $\rho_j$ to the neighbor $\sigma_j$
20:     end for
21: end if
```

The node that receives Tosignal stores a tuple (ρ, ω), where ω is the name of the node that sends ρ. This tuple is used during the signal return. Storing large quantities of (ρ, ω) is neither practical nor necessary as they become obsolete once path finding is completed. In the RCS protocol, each tuple is deleted after a specified timeout. Then the node updates the Tosignal by Algorithm 2. The update depends on two conditions: whether the Tosignal has reached the receiver and whether the current path length is less than k. If the current node is the receiver, it generates a Backsignal ρ_{back} and initiates the signal return. Otherwise, the node checks whether the current path length is less than k. If the current path length is equal to or greater than k, further extension would exceed the maximum allowed length. In such a case, the current node immediately stops signal transmission and generates a Nosignal ρ_{no}. The $flag$ of ρ_{no} is set to -1, indicating that the sender cannot reach the receiver via the current path. ρ_{no} is then entered into the signal return.

Algorithm 3 Backsignal and Nosignal: update and forwarding

```
Input: Backsignal $\rho_{back}$, Nosignal $\rho_{no}$, current node $\alpha$
1: if $\rho_{back}.s$ == the name of $\alpha$ then
2:     The transaction sender received the Backsignal $\rho_{back}$
3:     return
4: end if
5: Count the number $sum$ of Backsignal received that have
   with the same sender and receiver as $\rho_{no}$
6: if $sum > 0$ then
7:     discard $\rho_{no}$
8:     Identify the position $q$ of the last '/' in $\rho_{back}.id$
9:     Remove all characters from the q-th position in $\rho_{back}.id$
10:    for Each tuple ($\rho$, $\omega$) stored by $\alpha$ do
11:        if $\rho_{back}.id == \rho.id$ and $\rho_{back}.s == \rho.s$ and $\rho_{back}.t$
           $== \rho.t$ then
12:            $nextnode$ = $\omega$
13:            break
14:        end if
15:    end for
16:    Send $\rho_{back}$ to $nextnode$
17: else if $sum == 0$ then
18:    Identify the position $q$ of the last '/' in $\rho_{no}.id$
19:    Remove all characters from the q-th position in $\rho_{no}.id$
20:    for Each tuple ($\rho$, $\omega$) stored by $\alpha$ do
21:        if $\rho_{no}.id == \rho.id$ and $\rho_{no}.s == \rho.s$ and $\rho_{no}.t ==$
           $\rho.t$ then
22:            $nextnode$ = $\omega$
23:            break
24:        end if
25:    end for
26:    Send $\rho_{no}$ to $nextnode$
27: end if
```

If the current node is neither the receiver nor has the path reached the length limit k, it acts as an intermediate node. To extend the path, the node updates ρ and forwards it to its neighbors, excluding those already visited by ρ. For example, if a Tosignal traverses Carol and Alice before reaching Bob, whose neighbors are Alice, Carol, David, and Evin, then according to Algorithm 2, Bob forwards the Tosignal only to David, Evin. Consequently, the effective forwarding set has a size of 2 instead of 4, meaning $size$ is 2. If the current node cannot continue forwarding Tosignal ρ (i.e., $size = 0$), a Nosignal ρ_{no} is generated to enter the signal return. Otherwise, the current node updates $\rho.F[]$, which records its name and fee function. The node generates $size$ Tosignal $\{\rho_1, \dots \rho_{size}\}$, where each ρ_j is sent to the corresponding neighbors σ_j.

As the Tosignal is continuously updated and transmitted, it may eventually reach the transaction receiver, forming a topological path from the sender to the receiver. Once Backsignal or Nosignal is generated, the signal return process begins.

Signal return: Signal return consists of two types of signals: Backsignal and Nosignal. During the return, each node matches the received Backsignal with the stored Tosignal to determine the correct channel for forwarding. Finally, the sender receives the Backsignal and reconstructs the path to the receiver based on its information. In contrast, Nosignal is usually discarded and does not need to be returned to the sender.

The nodes that receive Backsignal and Nosignal update and forward signals by Algorithm 3 during the signal return. Upon receiving Backsignal ρ_{back}, the node first check whether it is the transaction sender by comparing its name with $\rho.s$. If they match, the Backsignal has reached the sender. Otherwise, the node updates the Backsignal and continues the return process. The id of each channel is unique in each path for the same sender-receiver pair. The tuple (ρ, ω) stored in the signal transmission and id can help identify the correct node that Backsignal further returns. Generally, any Nosignal can be discarded upon receipt unless no Backsignal is received by the node, in which case the Nosignal must be returned.

Finally, the sender collects all returned reverse signals and uses the intermediate nodes recorded in each Backsignal to reconstruct the paths, which are then added to the sender's routing table. If the sender only receives Nosignal and no Backsignal, it means that the shortest path between the sender and receiver exceeds k. In this case, RCS can incrementally increase the k value by 1 and re-execute the probing-based path finding algorithm until paths are found. The network diameter is defined as the maximum distance between any two nodes. For a PCN network, the diameter is approximately 10 [21]. So the value of k will not be adjusted to a large value in the worst case. Once the path is found, the sender resets k to l_{max} for subsequent transaction path finding.

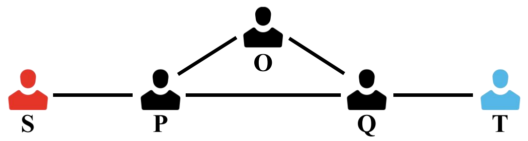


Fig. 3: Probe the paths between S and T with k=5.

Loop Avoidance. During the probing process, the signal may encounter a loop. The RCS protocol effectively prevents signal transmission within such loops. Fig. 3 illustrates how RCS achieves this. In this example, k=5, and there is a loop P-O-Q. When ρ reaches Q via S-P-O-Q, its current path length is 3. The updated ρ is then forwarded to T but not to O and P, because the signal transmission of RCS filters out the nodes that ρ has visited. Therefore, ρ does not loop within the P-O-Q during signal transmission. Similarly, during signal return, the signal follows the original path and also avoids loops.

C. Path Maximum Amount Calculation

After finding the path between the sender and the receiver in the routing table, the path is suitable for the transaction if the maximum transferable amount along it meets the sender's intended amount to send. In the following, we will explain how the RCS protocol determines whether sufficient funds are available along the path.

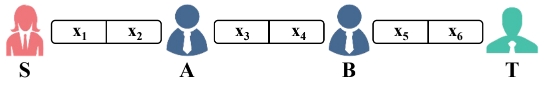


Fig. 4: Transaction path S-A-B-T.

Consider the path S-A-B-T in Fig. 4, where S represents the sender and T is the receiver of the transaction. Suppose that the fee function of A on C_{AB} is $F_{AB} = f_{base1} + a \times r_1$. Similarly, the fee function of B on C_{BT} is $F_{BT} = f_{base2} + a \times r_2$. If S completes the payment via this path, S must incur fees for both A and B. Let p_i represent the maximum transferable amount on each channel: p_1, p_2 and p_3 are the maximum amounts for the channels S-A, A-B, and B-T, respectively. The values of p_i must satisfy the following equations:

$$p_1 + f_{base1} + p_1 \times r_1 + f_{base2} + p_1 \times r_2 = x_1 \tag{2}$$

$$p_2 + f_{base2} + p_2 \times r_2 = x_3 \tag{3}$$

$$p_3 = x_5 \tag{4}$$

The values of p_i can be calculated from the above equations:

$$p_1 = (x_1 - f_{base1} - f_{base2})/(1 + r_1 + r_2) \tag{5}$$

$$p_2 = (x_3 - f_{base2})/(1 + r_2) \tag{6}$$

$$p_3 = x_5 \tag{7}$$

To successfully pay T an amount p along the path S-A-B-T, p must satisfy $p \leqslant min(p_1, p_2, p_3)$. During the signal transmission, the fee functions of the intermediate nodes are recorded in the signal, allowing each node on the path to know these fee functions based on the signal. Therefore, node A can calculate p_1 and p_2, and B can calculate p_2 and p_3. Thus, A knows that $p_A = min(p_1, p_2)$ and B knows that $p_B = min(p_2, p_3)$. To determine whether the path S-A-B-T can successfully execute the transaction, S, A, and B need only to jointly verify if $p \leqslant p_A$ and $p \leqslant p_B$.

D. *Transaction Execution*

For each transaction, the sender can randomly select a path from the routing table and collaborate with the intermediate nodes to determine whether the funds are sufficient. If the funds are sufficient, the path can be used for the transaction; otherwise, the process is repeated. If all paths in the routing table lack sufficient funds, it indicates that no path with a length not exceeding k is suitable for the transaction. In this case, k can be temporarily adjusted to find new paths as described in Section III-B.

Mitigating channel balance exhaustion. When a common criterion like "shortest-path-first" or "lowest-fee-first" is used by all senders to decide the final transaction path, a number of "high-quality" paths or channels would frequently be selected. This practice could quickly exhaust the channel balance of these paths, disrupting the connectivity of the entire PCN. To mitigate this issue, we propose a randomized selection approach that selects a path uniformly at random from the routing table. It ensures that different paths and channels have an equal probability of being selected, preventing overuse of some paths while others remain underused.

Handling concurrent requests. When multiple senders use RCS at the same time and execute their transactions on paths that share the same channels, a path collision occurs. The randomized path selection of RCS can effectively reduce the collision probability. In case collision still happens, RCS adopts the solution proposed in prior works [9, 10, 18]. Specifically, the time that a transaction starts is used to handle concurrent requests. The transaction that starts earlier is assigned a higher priority. When multiple transactions compete for the same channel, the transaction with higher priority gets to use the available balance first. If a transaction fails due to insufficient channel balance at run time, it immediately returns a failure result rather than waiting for the balance to increase, thereby avoiding deadlock. Upon failure, the sender selects a new transaction path. In case all paths fail, the sender can re-launch the path finding procedure.

IV. Privacy-preserving Design

This section presents the detailed design of the RCS+ protocol. RCS+ enhances RCS's path maximum amount calculation to preserve user privacy.

A. *Adversary Model*

Similar to existing privacy-preserving PCN designs [9, 17, 23], we assume a semi-honest adversary model, where adversaries adhere to the RCS protocol but attempt to infer private information about other user nodes from the data exchanged during protocol execution.

B. *Privacy Goals*

Similar to previous work [17, 23, 24], we consider the following privacy goals in RCS+.

Transaction amount privacy: No adversary can learn the total amount of transaction between uncompromised or honest users. In other words, all nodes other than the transaction sender and receiver cannot determine the transaction amount.

Available balance privacy: No adversary can determine the balance available in a payment channel between uncompromised users. Only the nodes at two ends of the payment channel know the current balance of their channel.

C. *Key Ideas and Detailed Description*

The path finding in RCS only relies on the public topology of the network and thus does not affect transaction or balance privacy. However, privacy leakage may occur when checking if balances along candidate paths are sufficient. As discussed in Section III-C, each intermediate node can independently compute the minimum value $p_{node.name}$ of the maximum transferable amounts of the two channels it participates in along the path, which does not compromise privacy. However, intermediate nodes inevitably learn the plaintext transaction amounts during the comparison process, leading to a leakage of transaction amount. Furthermore, if the sender repeatedly queries intermediate nodes along the same path with different transaction amounts (e.g., through binary search) before any updates to channel balances occur, it may infer the exact balance of a specific channel, compromising the privacy of available balances. To address this, we adapt the secure comparison protocol OT-CMP proposed by Deevashwer Rathee et al. [25] and construct the "L-bit-OT-CMP". Based on it, we design a secure path fund verification protocol RCS+.

Assume that the sender participating in the secure path funds verification is s, and the n intermediate nodes are $\{\pi_1, \pi_2, \ldots, \pi_n\}$. The sender s holds the transaction amount p_s, while each intermediate node π_i holds p_{π_i}. The goal is for s to securely verify that p_s does not exceed any p_{π_i} without revealing any additional information. To achieve this, the s and each π_i first run a secure comparison protocol and then combine all results to compute the final verification result. For comparisons, both s and π_i convert their values into ℓ-bit unsigned integers. They then securely compute a Boolean value $1\{p_{\pi_i} < p_s\}$ with the L-bit-OT-CMP protocol. If the comparison result is 0 (i.e., $p_{\pi_i} \geq p_s$) for all nodes, the path is confirmed to have sufficient funds. Conversely, if any intermediate nodes has $1\{p_{\pi_i} < p_s\}$ = 1, the path is considered to have insufficient funds.

In order to protect the comparison result a, the original OT-CMP lets one participant generate a random bit or mask b and the other participant holds $a \oplus b$. Therefore, the comparison result is secretly shared by the two participants and known to no one. To protect the individual comparison results between s and all π_i and enable their efficient combinations, we adapt the OT-CMP to make the comparison result secretly shared with two random L-bit binary numbers. Specifically, if the result is 0, the two participants would have two L-bit numbers whose bit-wise XOR equals 0^L; otherwise the bit-wise XOR equals a random L-bit number other than 0^L. The adaption can be efficiently implemented by changing the random mask b L-bit, and changing the comparison truth table's encoding as above. More details are presented in Algorithm 4.

As described in Algorithm 4, after executing L-bit-OT-CMP with every π_i, s obtains the masked comparison result lt_{π_i} and π_i gets the random mask $tempr_{\pi_i}$. Both lt_{π_i} and $tempr_{\pi_i}$ are L bits long. For secure combinations of the comparison results, starting from π_n, each π_i computes the XOR of its random mask and the received "combined mask", and sends the new combined mask to π_{i-1}. Finally, π_1 knows a combined mask that equals the XOR of all π_is' random masks, and send it to s. s computes the XOR of its all masked comparison results and the combined mask. If the result is L bits of zeros, the path has sufficient funds; otherwise, the funds are insufficient.

The L-bit-OT-CMP protocol in RCS+ allows s and each node π_i to obtain a random share of their comparison result. Since the protocol hides the selection choice, the transaction amount is hidden from intermediate nodes. Assuming nodes π_i are semi-honest and do not collude with sender s, it is easy to see s cannot learn the specific values of $1\{p_{\pi_i} < p_s\}$ ensuring that the available balance privacy is preserved, and verification has a false positive rate (i.e. the probability of outputting positive when funds are insufficient) of only $\frac{1}{2^L}$ as well as a false negative rate of 0.

Algorithm 4 Secure path funds verification

```
Input: $s$ holds $p_s$, $\{\pi_1, \ldots, \pi_n\}$ holds $\{p_{\pi_1}, \ldots, p_{\pi_n}\}$
Output: $s$ learns whether the funds on the path are sufficient.
 1: function EXTEND($z$, $R$)
 2:     if $z == 0$ then $tempz \longleftarrow 0^R$
 3:     else $tempz \xleftarrow{\$} \{0,1\}^R$
 4:     return $tempz$
 5: end function
 6: function $L$-bit-OT-CMP($P_0$, $P_1$, $x$, $y$)
 7:     $q = \ell/m$ and $M = 2^m$
 8:     $x = x_{q-1} \parallel \cdots \parallel x_0$, $y = y_{q-1} \parallel \cdots \parallel y_0$
 9:     for $j = \{0, 1, \ldots, q-1\}$ do
10:         $P_0$ samples $\langle lt_{0,j}\rangle_0^B$, $\langle eq_{0,j}\rangle_0^B \xleftarrow{\$} \{0,1\}^L$
11:         for $h = \{0, 1, \ldots, M-1\}$ do
12:             $ss_{j,h} = \langle lt_{0,j}\rangle_0^B \oplus EXTEND(1\{x_j < h\}, L)$
13:             $tt_{j,h} = \langle eq_{0,j}\rangle_0^B \oplus EXTEND(1\{x_j = h\}, L)$
14:         end for
15:         $P_0$ and $P_1$ invoke $\binom{M}{1} - OT_1$ with the inputs
            $\{ss_{j,h}\}_h$ from $P_0$ and $y_j$ from $P_1$. $P_1$ gets $\langle lt_{0,j}\rangle_1^B$
16:         $P_0$ and $P_1$ invoke $\binom{M}{1} - OT_1$ with the inputs
            $\{tt_{j,h}\}_h$ from $P_0$ and $y_j$ from $P_1$. $P_1$ gets $\langle eq_{0,j}\rangle_1^B$
17:     end for
18:     for $i = \{1, 2, \ldots, log\ q\}$ do
19:         for $j = \{0, 1, \ldots, (q/2^i) - 1\}$ do
20:             For $b \in \{0,1\}$, $P_b$ invokes $\mathcal{F}_{AND}$ with inputs
                $\langle lt_{i-1,2j}\rangle_b^B$ and $\langle eq_{i-1,2j+1}\rangle_b^B$ to get output $\langle temp\rangle_b^B$
21:             $P_b$ sets $\langle lt_{i,j}\rangle_b^B = \langle lt_{i-1,2j+1}\rangle_b^B \oplus \langle temp\rangle_b^B$
22:             For $b \in \{0,1\}$, $P_b$ invokes $\mathcal{F}_{AND}$ with inputs
                $\langle eq_{i-1,2j}\rangle_b^B$ and $\langle eq_{i-1,2j+1}\rangle_b^B$ to get output $\langle eq_{i,j}\rangle_b^B$
23:         end for
24:     end for
25:     return $P_0$ gets $L$-bit random value $\langle lt_{log\ q,0}\rangle_0^B$, $P_1$ gets
        $\langle lt_{log\ q,0}\rangle_1^B$ which has been randomized by $\langle lt_{log\ q,0}\rangle_0^B$
26: end function
27: procedure Main()
28:     for $\pi_i$ in $\{\pi_1, \ldots, \pi_n\}$ do
29:         call $L$-bit-OT-CMP($\pi_i$, $s$, $p_{\pi_i}$, $p_s$)
30:         $\pi_i$ gets $L$-bit random value $tempr_{\pi_i}$ and $s$ gets $lt_{\pi_i}$
        which has been randomized by $tempr_{\pi_i}$
31:     end for
32:     $tempr = tempr_{\pi_n}$
33:     for $\pi_i$ in $\{\pi_{n-1}, \ldots, \pi_1\}$ do
34:         $tempr = tempr \oplus tempr_{\pi_i}$
35:     end for
36:     $\pi_1$ sends $tempr$ to $s$
37:     $s$ calculates $lt_{\pi_1} \oplus lt_{\pi_2} \oplus \cdots \oplus lt_{\pi_n} \oplus tempr$
38: end procedure
```

D. Privacy Analysis

We provide a short analysis explaining that RCS+ achieves the privacy goals proposed in Section IV-B.

Transaction amount privacy. When determining whether the path funds are sufficient, only the transaction sender knows the transaction amount. The sender and the nodes use a secure comparison protocol to compare the transaction amount with the available funds. The security of secure comparison protocols guarantees that nodes learn nothing more than the comparison results they received by participating in the protocols. Since the results of L-bit-OT-CMP received by the nodes are random masks chosen by themselves, the nodes know nothing about the transaction amount. Thus, transaction amount privacy is ensured.

Available balance privacy. The intermediate nodes use their corresponding channel balances to calculate the maximum amount of fund that can pass through themselves. Same to the transaction amount, the calculated values are protected by the secure comparisons against the sender. Assuming the nodes are semi-honest and do not collude with the sender, the individual comparison result between any intermediate node and the sender cannot be extracted due to the fact that only the combined mask is revealed to sender, and only the random masks are revealed to neighboring nodes. This guarantees that 1) the sender only knows all channels have sufficient funds that are no less than the transaction amount if the verification passes, and that the available balance of at least one unknown channel is less than the transaction amount otherwise; 2) the nodes knows nothing about others. This effectively protects the available balance privacy of the nodes.

V. Experimental Evaluations

In this section, we comprehensively evaluate and compare the performance of RCS and RCS+ with other routing protocols. Specifically, our evaluation aims to investigate the following aspects:

- Their performance in real-world networks.
- The impact of PCN evolution on their performance.
- The comparison of them with other outstanding protocols on the above performance.

A. Experimental Setup

Simulator. We extend the simulator described in [10], implementing our routing protocol, RCS and RCS+, and simulate network by NetworkX [26] in Python. Focusing on routing performance, we simplify the simulation of PCN by omitting underlying security mechanisms, such as HTLC. In our simulation, the network condition (e.g., topology and available balances) changes dynamically as we simulate a series of transactions chronologically. This allows our simulations to more accurately reflect real-world conditions. While failed transactions can be retried after being locked for a period in PCNs, these inefficient retries are not considered in our experiments. To minimize experimental variability and randomness, we report the average results over 5 runs.

Topology. To evaluate the performance of routing protocols in real networks, we conduct experiments using the data from two real-world PCNs: Lightning and Ripple [27]. Their actual topologies are obtained from [28] and [29] respectively. Lightning consists of 9,867 nodes and 38,390 channels, while Ripple comprises 1,870 nodes and 17,416 channels. With the collected data, we reconstruct the Lightning and Ripple. Additionally, we simulate networks with varying numbers of nodes using the Watts Strogatz graph [30], which accurately represents PCN structures.

Workload. In PCNs, any node can act as the sender or receiver of a transaction. To simulate transactions in the experiment, two nodes are randomly selected to serve as the sender and receiver for each transaction. For Lightning and Ripple networks, the transaction amounts are randomly generated based on the ranges derived from their respective channel capacity data. In the simulated network, transaction amounts are randomly generated according to ranges determined by its total channel capacity.

Parameters. Based on the data collected from the Lightning and Ripple, we determine the parameter $l_{max} = 4$ for each network. According to the RCS protocol, the k-value for all nodes is initially set to l_{max}. In rare cases where no transaction path of length k or shorter exists or where such paths lack sufficient funds, RCS or RCS+ incrementally increases k by 1 until a feasible path is found. Once the issue is resolved, k is reset to $l_{\max}$ for subsequent transactions. Therefore, in most cases, all nodes maintain $k = l_{\max} = 4$.

Benchmarks. We evaluate and compare RCS and RCS+ with the following four notable routing protocols.

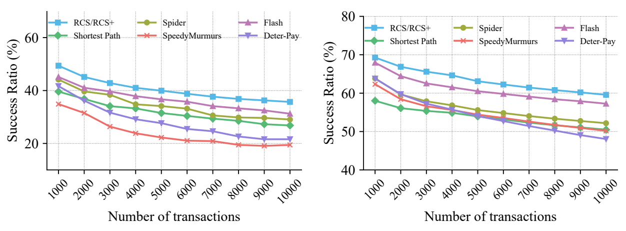
- *Shortest Path*: The sender selects the shortest path from itself to the receiver and uses it to execute the transaction. It is the core idea of the currently deployed routing protocol LND in the Lightning network.
- *SpeedyMurmurs* [9]: It adopts the landmark routing approach, where landmark nodes identify paths. Transaction amounts are then randomly distributed among these paths.
- *Spider* [11]: Each sender maintains k edge-disjoint widest paths to each receiver. Transactions are split into units for transmission. If a transaction-unit cannot be sent, it is placed in a per-destination queue at the sender that is served in LIFO order.
- *Flash* [10]: It categorizes transactions into mice and elephant payments based on their amounts. For mice payments, each node uses the paths in its routing table. For elephant payments, Flash designs a max-flow algorithm to find multiple paths, then splits the transaction and selects the lowest-fee path combination.
- *Deter-Pay* [19]: It probes only K paths among all feasible ones, and completes the transaction if the total passable amount across them exceeds the required payment.

Metrics. We use the following metrics to evaluate the performance of protocols:

- *Success ratio*: The percentage of successfully completed transactions among all generated transactions.
- *Success volume*: The total transaction amount of all successfully completed transactions.
- *Overhead*: Regardless of whether the protocol probes to find a path or checks for sufficient funds, users need to

communicate. We quantify this overhead by counting the number of messages transmitted.

The RCS+ protocol is a privacy-preserving extension of the RCS protocol. It maintains the same success ratio and success volume as RCS, but introduces additional overhead.

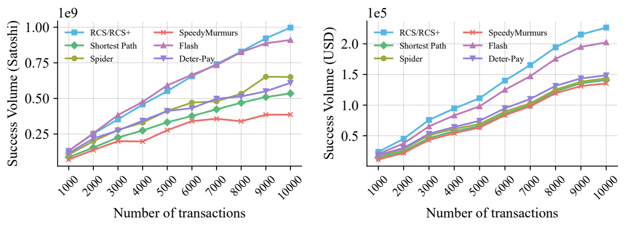


(a) Lightning (b) Ripple

Fig. 5: Success ratio in real-world networks.

B. *Performance in Real-world Networks*

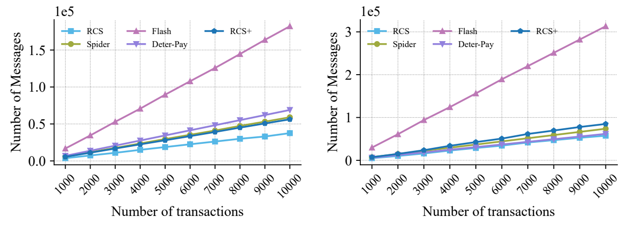
Success ratio. Fig. 5 illustrates the success ratio of each protocol with different numbers of transactions being executed in both Lightning and Ripple. **RCS and RCS+ outperform all baselines in both networks.** In Lightning, they achieve a success ratio **approximately 16% higher than SpeedyMurmurs and 12% higher than Deter-Pay**, and outperform Shortest Path, Spider, and Flash by about 10%, 6%, and 4%, respectively. In Ripple, their success ratio surpasses that of Shortest Path and Deter-Pay by around 10%, SpeedyMurmurs and Spider by 8%, and Flash by 3%. RCS and RCS+'s superior performance is due to its pre-transaction verification of path funds. As the number of transactions increases, the success ratio of all protocols decreases since channel balances are consumed and no new payment channels are added.



(a) Lightning (b) Ripple

Fig. 6: Success volume in real-world networks.

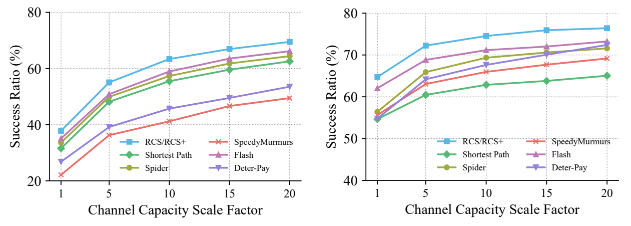
Success volume. Fig. 6 shows the changes in the success volume of each protocol in Lightning and Ripple as the number of transactions increases. Although the success ratio of each protocol declines with more transactions, the total number of completed transactions rises, resulting in an overall upward trend in success volume. **In both Lightning and Ripple, RCS and RCS+ outperform all baselines in the success volume**. In Lightning, RCS and RCS+ have a success volume that is 2.1x that of SpeedyMurmurs, 1.7x that of Shortest Path, and 1.4x that of Spider and Deter-Pay, and are comparable to Flash. The higher average channel capacity of Lightning allows for more effective support of elephant payments compared to Ripple. Flash employs "confirm-and-send" for large-amount payments and splits transactions, enhancing its performance in large-amount transactions and boosting its success volume. In Ripple, RCS and RCS+ similarly outperform Shortest Path, SpeedyMurmurs, Spider and Deter-Pay, achieving about 1.6x their success volumes, and surpass Flash by achieving 1.2x its success volume.



(a) Lightning (b) Ripple

Fig. 7: Overheads

Overhead. Among the evaluated protocols, only RCS, RCS+, Flash, Spider, and Deter-Pay require probing. We compare their communication overheads on the Lightning and Ripple networks. Fig. 7 shows that **RCS consistently incurs the lowest overhead on both networks, amounting to only 20% of Flash's overhead, which is the highest among the compared protocols**, as the number of transactions increases from 1,000 to 10,000. This efficiency stems from the fact that most transactions occur between recurring sender-receiver pairs, allowing RCS's routing table to reduce repeated path discovery. Although RCS+ incurs more communication due to secure comparisons, its overhead remains low—only 30% of Flash's overhead.



(a) Success ratio in Lightning (b) Success ratio in Ripple

Fig. 8: Performance with different channel capacities.

C. *Performance in Network Evolution*

Performance with different channel capacities. The overall channel capacity of PCNs is currently limited but is expected to grow as PCNs evolve [21]. Following existing work [9–11], we simulate this growth by expanding the current channel capacity from 1x to 20x. The number of transactions is fixed at 5,000. Fig. 8 shows the success ratio of each protocol as network channel capacity changes. As the payment channel balance increases, the success ratio of each protocol improves. **In both Lightning and Ripple, RCS and RCS+ consistently achieve the highest success ratio.** Particularly in the Lightning Network, their success ratio is approximately 20% higher than that of SpeedyMurmurs.

Performance with different number of nodes. As more users join PCNs, the number of PCN nodes expands. Based on the topology of the simulated network, we set l_{max} to 6 for RCS/RCS+ in our experiments. The number of transactions is fixed at 5000. Fig. 9 illustrates the success ratio of each protocol as the number of network nodes varies from 10000 to 20000. The success ratio of all protocols remains relatively stable, indicating their adaptability to an increasing number of network nodes. Notably, **RCS and RCS+ consistently achieve the highest success ratio as the number of nodes grows.**

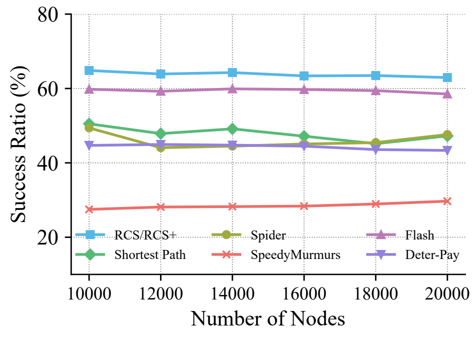


Fig. 9: Performance with different number of nodes.

VI. Related Work

The primary motivation of our work is to design a novel routing protocol that improves PCN performance while preserving user privacy. In this section, we discuss related works.

Application of probing in PCN routing protocols. Several probing-based protocols [10, 11, 18, 19], including ours, probe to find transaction paths. Spider [11] employs dynamic routing to select the optimal paths per transaction but incurs probing overhead due to the fact that channel balances change after each transaction, requiring probing paths for each transaction. Flash [10] employs dynamic routing for elephant payments to ensure high performance , while relying on pre-recorded paths from the sender's routing table for mice payments to reduce overhead. EPA-Route [18] cuts probing overhead by pruning the next hop of each probe at every hop. Deter-Pay [19] only probes a subset of paths. If the total balance available on these paths does not support the transaction, it continues to probe the remaining paths. RCS lowers probing overhead by utilizing routing tables and limiting the number of paths probed based on network topology. The experimental results show that RCS incurs lower overhead than other protocols.

Improvements in PCN routing performance. In current PCNs, users in LND [15] attempt to pay along the shortest path. This blind attempt reduces the transaction success rate. Several protocols have been proposed to address the issue. CoinExpress [16] employs the Ford-Fulkerson max-flow algorithm for finding paths. FSTR [7] selects paths that minimize fund inclination. cRoute [8] guides routing based on network congestion gradients without computing specific paths. Flash [10] adjusts the path-finding strategy according to the transaction amount, improving the success rate and controlling overhead. Spider [11] selects the k edge-disjoint widest paths for transactions. Additionally, protocols like Flash, Spider, Auto-tune [13] and SpeedyMurmurs [9] split transactions to enhance performance but increase transaction fees. Other works [31–33] focus on reducing payment fees while maintaining routing performance.

However, most existing routing protocols primarily employ the "guess-and-check" strategy, meaning they cannot verify the sufficiency of funds on the selected path before a transaction. Verifying the availability of sufficient funds beforehand would improve the transaction success rate, which called "confirm-and-send". But it requires checking all potential paths, which is time-consuming. We optimize it to minimize the number of paths that need to be verified for sufficient funds, calling it the "refined confirm-and-send". Based on this optimized strategy, we proposed RCS, which maintains a reasonable path finding overhead while achieving an excellent transaction success rate.

Privacy-preserving implementation in PCN routing protocols. Several proposals for privacy-preserving payments utilize techniques such as zero-knowledge proofs [34], decentralized mixing [35] and secure multi-party computation (SMPC) [36]. PrivPay [23] introduces the concepts of value privacy and sender/receiver privacy. It enhances privacy by employing trusted hardware on the central server, but its scalability remains poor. SilentWhispers [17] utilizes SMPC technology to protect privacy, but nodes must send messages to each landmark node, resulting in large overall time overhead. SpeedyMurmurs [9] and Sprite [37] surpass SilentWhispers in terms of transaction efficiency but exhibit lower transaction success rates compared to non-privacy-preserving protocols. WebFlow [38] strikes a balance between low overhead and a high transaction success rate but does not preserve the available balance privacy. To address users' privacy needs, we introduce a secure comparison scheme and implement RCS+ that protects both transaction amount and available balance privacy while maintaining excellent routing performance.

VII. Conclusion

In this work, we design RCS, a routing protocol that employs the "refined confirm-and-send" strategy. The required transaction paths are short based on the characteristics of transaction paths in PCNs, and each user maintains their own routing table to avoid repeated path-finding between the same sender and receiver, thereby reducing overhead. Furthermore, to address users' strong privacy concerns, we implement RCS+ using a secure comparison protocol. Through simulations with real-world and synthetic datasets, we demonstrate that RCS and RCS+ outperform other routing protocols, especially in terms of transaction success rate, while maintaining a reasonable overhead. Additionally, the experimental results show that RCS and RCS+ adapt well to the evolution of PCNs.

Acknowledgement

We sincerely thank the anonymous reviewers for their valuable and constructive comments. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB2702800, the Natural Science Foundation on Frontier Leading Technology Basic Research Project of Jiangsu under Grant BK20222001, NSFC Grants No.62272215, No.62272222, and the Fundamental Research Funds for the Central Universities (No. 2024300401); Yuan Zhang and Sheng Zhong are the corresponding authors.

References

[1] "The Lightning Network," https://lightning.network/.

[2] "The Raiden Network," https://raiden.network/.

[3] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[4] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 17–30.

[5] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Deconstructing the blockchain to approach physical limits," *Cryptology ePrint Archive*, 2018.

[6] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, vol. 144, 2016.

[7] S. Lin, J. Zhang, and W. Wu, "Fstr: Funds skewness aware transaction routing for payment channel networks," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 464–475.

[8] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," *arXiv preprint arXiv:1810.00037*, 2018.

[9] A. K. Stefanie Roos, Pedro Moreno-Sanchez and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *25th Annual Network and Distributed System Security Symposium (NDSS 2018)*, 2018.

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

[11] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 777–796.

[12] L. Eckey, S. Faust, K. Hostáková, and S. Roos, "Splitting payments locally while routing interdimensionally," *Cryptology ePrint Archive*, 2020.

[13] H.-J. Hong, S.-Y. Chang, and X. Zhou, "Auto-tune: An efficient autonomous multi-path payment routing algorithm for payment channel networks," *Computer Networks*, vol. 225, p. 109659, 2023.

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

[15] "Lightning Network Daemon (LND)," https://github.com/ lightningnetwork/lnd.

[16] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks," in *2018 27th international conference on computer communication and networks (ICCCN)*. IEEE, 2018, pp. 1–9.

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

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

[19] Q. Cai, J. Chen, D. Luo, G. Sun, H. Yu, and M. Guizani, "Deter-pay: A deterministic routing protocol in concurrent payment channel network," *IEEE Internet of Things Journal*, 2024.

[20] D. Karlan, M. Mobius, T. Rosenblat, and A. Szeidl, "Trust and social collateral," *The Quarterly Journal of Economics*, vol. 124, no. 3, pp. 1307–1361, 2009.

[21] Y. Y. Chen YJ, Zhu XT and C. ZY, "Empirical analysis of lightning network: topology, evolution, and fees," *Journal of Software*, vol. 33, no. 10, pp. 3858–3873, 2021.

[22] F. Béres, I. A. Seres, and A. A. Benczúr, "A cryptoeconomic traffic analysis of bitcoin's lightning network," *Cryptoeconomic Systems*, 2020.

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

[24] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 455–471.

[25] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[26] "NetworkX," https://networkx.github.io/.

[27] "Ripple," https://ripple.com/.

[28] "Real-Time Lightning Network," https://1ml.com.

[29] "Ripple Transaction," https://crysp.uwaterloo.ca/software/speedymurmurs/.

[30] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[31] Y. Zhang, D. Yang, and G. Xue, "Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks," in *ICC 2019-2019 ieee international conference on communications (icc)*. IEEE, 2019, pp. 1–6.

[32] Z. Avarikioti, L. Heimbach, Y. Wang, and R. Wattenhofer, "Ride the lightning: The game theory of payment channels," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 264–283.

[33] S. Tochner, A. Zohar, and S. Schmid, "Hijacking routes in payment channel networks: A predictability tradeoff," in *2nd ACM Conference on Advances in Financial Technologies (AFT)*, 2020.

[34] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 397–411.

[35] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II 19*. Springer, 2014, pp. 345–364.

[36] K. El Defrawy and J. Lampkins, "Founding digital currency on secure computation," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1–14.

[37] G. Panwar, R. Vishwanathan, G. Torres, and S. Misra, "Sprite: Secure and private routing in payment channel networks," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 1878–1894.

[38] X. Zhang, S. Shi, and C. Qian, "Low-overhead routing for offchain networks with high resource utilization," in *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2023, pp. 198–208.