RobustPay$^+$: Robust Payment Routing with Approximation Guarantee in Blockchain-based Payment Channel Networks

Yuhui Zhang, *Student Member, IEEE,* and Dejun Yang, *Senior Member, IEEE,*

Abstract—The past decade has witnessed an explosive growth in cryptocurrencies, but the blockchain-based cryptocurrencies have also raised many concerns, among which a crucial one is the scalability issue. Suffering from the large overhead of global consensus and security assurance, even the leading cryptocurrencies can only handle up to tens of transactions per second, which largely limits their applications in real-world scenarios. Among many proposals to improve the cryptocurrency scalability, one of the most promising and mature solutions is the payment channel network (PCN), which offers the off-chain settlement of transactions with minimal involvement of expensive blockchain operations. However, transaction failures may occur due to external attacks or unexpected conditions, e.g., an uncooperative user becoming unresponsive. In this paper, we present a distributed robust payment routing protocol RobustPay$^+$ to resist transaction failures, which achieves robustness, efficiency, distributedness and approximate optimization. Specifically, we investigate the problem of robust routing in PCNs from an optimization perspective, which is to find a pair of payment paths for a payment request, while minimizing the worst-case transaction fee, subject to the timeliness and feasibility constraints. We present a distributed 2-approximation algorithm for this problem. Moreover, we modify the original Hashed Time-lock Contract (HTLC) protocol and adapt it to the robust payment routing protocol to achieve robustness and efficiency. Extensive simulations demonstrate that RobustPay$^+$ significantly outperforms baseline algorithms in terms of the success ratio and the average accepted value.

Index Terms—Cryptocurrency, payment channel network, routing, blockchain

1 Introduction

Over the past decade, the blockchain-based cryptocurrencies have risen to more than $80 billion in peak capital, including Bitcoin [13], Ethereum [18], and Ripple [20]. These altcoins make use of the blockchain technology to achieve real-time total settlement [19] of different currencies and assets [29, 30], which is much cheaper than the current central bank system. Nevertheless, when attempting to scale up blockchains like Bitcoin and Ethereum, several concerns emerge. Firstly, every participant needs to know every single transaction to ensure a unique and synchronized global status. This leads to high overhead and demand for local storage. For instance, it could cost a Bitcoin user almost 20 GB (can increase to 60 GB after three years) of additional storage each year [9]. Further, blocks can only be added at a certain maximum rate determined by the necessary proof-of-work computations that need to be carried out by generating a number whose hash value that starts with a pre-decided number of zeros. Taking the Bitcoin blockchain as an example, the maximum number of transactions per second (tps) is only 7 [8], which is not comparable to over 47,000 peak tps processed by Visa [26].

The payment channel network (PCN) was proposed to tackle the scalability issues [16]. A simple illustration of PCN is shown in Fig. 1. PCNs can process instant and less valuable payments without involving blockchain transactions which are slow and expensive. Only the initial and final balances of each channel are required to be registered. Section 3 provides an in-depth discussion of the PCN working mechanism. PCNs have been employed to develop Bitcoin's Lightning Network [16] and Ethereum's Raiden Network [14]. The uniqueness of PCNs is that PCNs can significantly lower the transaction fees than in blockchain. It is achieved by allowing a payment sent to the recipient through multiple hops between payment channels. In the payment transferring process, hosts of the channels on the route can charge fees accordingly. Therefore, the key motivation is to optimize the routing in PCNs and guarantee the success of a payment.

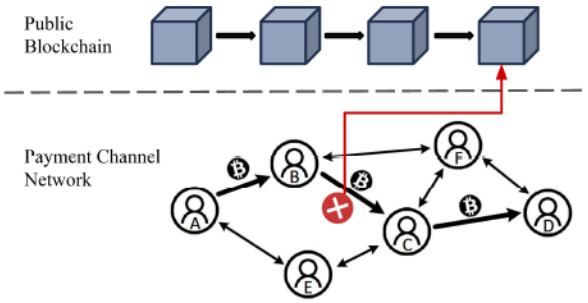


Fig. 1. Payment channel network (PCN) with a payment transaction from A to D. The transactions are off-chain. When two parties disagree with the transaction history, the transaction history will be published to the blockchain for verification. The dishonest party will be penalized.

Several reported research works have investigated payment routing in PCNs [12, 17, 21–23, 31]. However, these efforts either emphasize on privacy [12, 22], or underestimate the importance of key realistic constraints such as the

This paper is an extended and enhanced version of [32]. Zhang and Yang are affiliated with Colorado School of Mines, Golden, CO 80401. Email:{yuhzhang, djyang}@mines.edu. *This research was supported in part by NSF grants 1717315 and 2008935. The information reported here does not reflect the position or the policy of the federal government.*

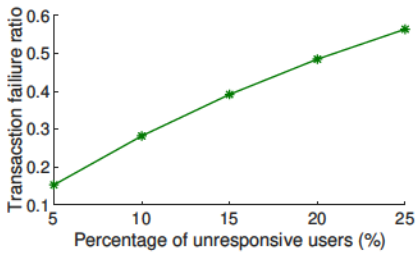


Fig. 2. Impact of unresponsive users on the transactions

transaction fees [12, 17, 21–23, 31]. One misconception of the routing in PCNs is that PCNs may be treated as a conventional computer ad-hoc network. In fact, the complexity of routing in PCNs is increased by two distinct features, which are not found in conventional ad-hoc networks. The first distinct feature is the **feasibility constraint**, which results from the fact that additional fees paid to the users along the route have to be sent together with the payment to the recipient. The second feature is referred to as the **timeliness constraint**, which is because the tolerance on the cooperating time in payment forwarding process of each PCN user is different. Due to these differences between PCNs and computer ad hoc networks, PCNs cannot apply the algorithms for conventional routing problems directly.

Recently, Zhang *et al*. [33] designed an optimal algorithm CheaPay to generate a single payment path that minimizes the transaction fee in PCNs and satisfies both the feasibility and timeliness constraints. However, CheaPay cannot resist transaction failures due to unexpected conditions or external attacks along the path. For example, some uncooperative intermediate users on the path may become unresponsive intentionally or unintentionally. Being unresponsive means that the channel funds may be on hold until the HTLC expires. A channel participant keeps funds in a channel either to make a payment for goods or service (which now is delayed), or to earn fees by forwarding a transaction. Since the value of cryptocurrencies fluctuates dramatically, the time value of money (TVM) concept in financial management also applies to cryptocurrencies, or might contribute more according to the historic volatility. Thus, unresponsive users are a liability to channel participants. It is similar to a denial of service attack, which ties up cryptocurrencies rather than bandwidth. In order to evaluate the impact of the unresponsive users on the transactions, we conducted simulations on a real-world dataset from the Bitcoin Lightning Network [5]. The simulation result in Fig. 2 shows that the transactions failure ratio increases by 15%, when 5% users are unresponsive. This indicates that unresponsive users are indeed an issue and need to be taken into consideration during payment routing. Therefore, PCNs are expected to provide better **robustness** against transaction failures due to unresponsive users, *i.e.*, a payment routing protocol satisfies robustness, if it constructs two or more node-disjoint payment paths, where each payment path can fulfill the payment request. A payment is transferred on these payment paths simultaneously. If one path fulfills the payment first, the other path(s) will be invalidated.

In this paper, we investigate the robust payment routing by constructing two node-disjoint payment paths for a payment request in PCNs. A robust payment routing has a number of distinct characteristics, thus it is expected to satisfy a set of desired properties. First, a robust payment routing protocol is expected to minimize the worst-case transaction fee, which is referred to as **optimization**, since it constructs more than one node-disjoint payment paths to resist transaction failures. As for NP-hard problems, we can only seek for **approximate optimization**. Secondly, a robust payment routing protocol should satisfy **efficiency**, i.e., to minimize the routing and payment latency incurred by transmitting a payment through multiple payment paths simultaneously. Finally, a robust payment routing should satisfy **distributedness**, as no central administrative operator exists in PCNs. Even if such an operator exists, it would not be trusted.

In face of these challenges, we propose RobustPay$^+$, a robust payment routing protocol that satisfies robustness, approximate optimization, efficiency and distributedness. Specifically, we investigate the problem of robust payment routing in PCNs from an optimization perspective. This problem is referred to as the **Maximum Transaction Fee Minimization (MTFM)** problem: minimizing the maximum transaction fee of a pair of node-disjoint paths to transfer a payment from the sender to the recipient, while guaranteeing that both the timeliness and feasibility constraints are satisfied for each involved payment channel on this pair of node-disjoint paths. Meanwhile, we design an HTLC mechanism providing more flexible choices and security to users and adapt it to the robust payment routing protocol. **The main contributions of this paper are**:

- To the best of our knowledge, we are the first to consider the robust payment routing protocol, which provides resistance to transaction failures in PCNs.
- We investigate important design goals of payment routing in PCNs, which are robustness, efficiency, distributedness and approximate optimization.
- We propose RobustPay$^+$, a distributed **Robust Pay**ment routing protocol against transaction failures in PCNs. RobustPay$^+$ consists of three stages: Payment Path Construction, HTLC Establishment and Payment Forwarding.
- We enhance the robustness for payment routing in PCNs by constructing two node-disjoint paths for a payment request and achieve approximate optimization by designing a distributed 2-approximation algorithm to minimize the worst-case transaction fee.
- We also modify the original HTLC protocol and adapt it to RobustPay$^+$ to guarantee efficiency.
- Extensive simulations demonstrate that RobustPay$^+$ not only minimizes the worst-case transaction fee, but also achieves superior success ratio and average accepted payment value over baseline algorithms.

The remainder of the paper is organized as follows. In Section 2, we provide a brief literature review of related work. In Section 3, we present the background and system overview of PCNs. In Section 4, we formally describe the system model, outline the design goals and give the problem formulation. In Section 5, we illustrate the robust payment routing protocol RobustPay$^+$, demonstrate the design of the routing algorithm and analyze the properties. In Section 6,

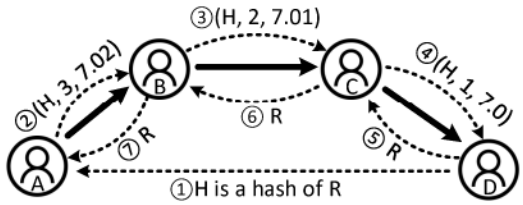


Fig. 3. Hashed time-lock contract (HTLC). The sender A sends a payment of 7 to the recipient D via B and C with an HTLC tolerance of 3. Assume that the transaction fee charged by each user is 0.01. Circled numbers represent the sequence of the operations.

we test and validate the performance of RobustPay$^+$ by comparing it to baseline algorithms. We summarize this paper in Section 7.

2 Related Work

Up to now, there are only limited efforts on studying the routing problems in PCNs. Three years ago, one of the pioneering decentralized routing algorithms for PCNs, known as Flare [17]. In this algorithm, there is a routing table on each node, formed by the adjacent nodes that are close in hop distance and paths to a list of beacon nodes. The privacy-reserving routing problem was studied by Malavolta [12]. He developed SilenWhisper, which is a routing scheme based on Landmark Routing [27]. In SilenWhisper, the landmark that is passed by all paths may result in unnecessarily long paths. Furthermore, this approach can violate decentralization, which is the only intention of using the blockchain system. SpeedyMurmurs [22] used an improved algorithm based on embedding-based path discovery [15]. In this way, the weakness of SilentWhisper could be overcome and improved in terms of success ratio, delay of payment, overhead, length of path, and stabilization. Rohrer *et al.* [21] sketched the payment flow as multiple paths adding up together to make use of the available capacities in the network in an efficient way. Following Rohrer's path, Yu *et al.* [31] outlined a scattered algorithm, which improved the success ratio of payment and reduced the system overhead. However, all the previous studies either concentrate on the privacy [12, 22] or simplifying the problem without taking into account the hop-dependent constraints [12, 17, 21, 22, 31].

Recently, Bagaria *et al.* [6] devised a technique that constructs redundant payment paths free of counterparty risk. However, this solution was designed for multi-path routing schemes without considering optimal routing. Zhang *et al.* [33] proposed CheaPay to minimize the transaction fee of a payment path, while considering the timeliness and feasibility constraints. But CheaPay did not provide robustness to payment routing in PCNs, including responses to transaction failures due to unexpected conditions, e.g., an uncooperative user becoming unresponsive.

3 Background and System Overview

In this section, we provide the necessary background on permissionless blockchains and present an overview of our payment channel network system.

3.1 Decentralized Ledger

Cryptocurrencies like Bitcoin [13], Ethereum [18], and Ripple [20] are based on the blockchain technology, which is an append-only decentralized ledger of transactions shared among mutually distrusted entities. However, the consensus algorithm (e.g. proof-of-work in Bitcoin) that guarantees the unique global state requires large local storage, due to the high levels of data replication and computational power for adding a block containing transactions to the blockchain.

Scalability Issue. The main concern of decentralized blockchains is that every peer needs to be aware of all transaction of all other peers to not be vulnerable to double-spending. Bitcoin currently only supports up to 7 transactions per second [8] which is not comparable to over $47,000$ peak tps processed by Visa [26]. Therefore, the blockchain-based cryptocurrencies cannot scale for wide-spread use.

3.2 Payment Channel

To overcome the scalability issue, off-chain approaches have been proposed to eliminate the need to commit each individual transaction to the blockchain. The use of payment channels is one way to realize the off-chain approach. Two users establish a payment channel by each depositing a certain amount into a joint account and adding this transaction to the blockchain.

Now a transaction between them is essentially a channel balance update agreed upon by them. A channel is protected by multi-signature smart contracts, which ensure validity, nonequivocality and non-repudiation of the ongoing transactions. When one party publishes an obsolete balance history to reverse settled transactions or to double-spend, the contract guarantees that the dishonest party is punished by granting all its remaining channel balance to the other party. This economically prevents an adversary from utility gain via dishonest behaviors. When the channel closes because either it is not needed anymore or the deposit is depleted, a closing transaction will be broadcast to the blockchain and will send deposited amount to each user according to the most recent balance.

3.3 Payment Channel Network

Unfortunately, payment channel alone cannot solve the scalability issue. Requiring everyone to create a payment channel with everyone else results in a large amount of on-chain transactions broadcast to the blockchain. In order to enable payments between any two users, payments can be routed through multiple hops of channels in the payment channel network (PCN) formed by users connected by payment channels. This, however, can lead to issues that a user denies performing payment transfer after receiving a preceding one, or the recipient denies receiving the payment.

3.4 Hashed Time-Lock Contract (HTLC).

To address these issues, the Hashed Time-Lock Contract (HTLC) mechanism has been introduced [16], as shown in Fig. 3. The recipient first generates a random value R and sends its hash H to the sender. The sender, as well as any intermediate user, includes H in the transaction contract, such that the transferred payment can be claimed

by the transferee only when the secret R is provided to the transferor. In addition, each transaction is restricted by an HTLC tolerance, such that if the transferor does not receive R within the HTLC tolerance, the transferred fund will be refunded to the transferor after the HTLC expires. The unit of the HTLC tolerance, denoted by δ, is the worse-case bound on time for one on-chain transaction. Every user in the payment path sets a tolerance, which is a smaller HTLC tolerance in the outgoing payment channel than that in the incoming payment channel. For example, in Lightning Network, the tolerance is set as the number of hops until the recipient [16]. As an example, the HTLC $(H, 2, 7.01)$ from B to C in Fig. 3 means that C can receive a payment of 7.01 from B if C can provide the preimage of H within 2δ. This mechanism ensures that a user can pull the payment from its predecessor after its payment has been pulled by its successor. Note that the HTLC tolerance time is not the time of payment routing, which is fast when users are cooperative and responsive. In addition to the payment to the recipient, an HTLC also includes the transaction fees charged by the intermediate nodes for transferring the sender's payment. The fees are significantly lower than blockchain transaction fees largely due to the time-value of locking up funds in the channel, as well as paying for the chance of channel close on the blockchain.

3.5 Challenges

The main challenge of the routing in PCNs is that PCNs cannot be treated as a conventional computer ad-hoc network. In fact, the complexity of routing in PCNs is increased by two distinct features, which are not found in conventional ad-hoc networks. The first distinct feature is the **feasibility constraint**, which means different balance requirements on different channels. It is resulted from that fees paid to the users along the route have to be sent together with the payment to the targeted recipient, during the payment transferring process. The second feature is referred to as the **timeliness constraint**, which cannot be found in computer ad hoc networks. The reason is that the tolerance on the cooperating time in payment forwarding process of each PCN user is different. The tolerance is judged by the number of hops to the recipient. Due to these differences between PCNs and computer ad hoc networks, PCNs cannot apply the algorithms for conventional routing problems directly.

4 System Model and Problem Formulation

In this section, we describe the network model and the payment model, outline the desired design goals and give a precise problem formulation.

4.1 Network Model

A PCN can be represented as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. Each node $v_i \in \mathcal{V}$ represents a user, who has a cryptocurrency account and establishes at least one payment channel with a peer user. Each edge $e = (v_i, v_j) \in \mathcal{E}$ represents a payment channel, where v_i is the *transferor* and v_j is the *transferee*. Each edge is associated with several attributes. First, each edge $(v_i, v_j) \in \mathcal{E}$ has a transaction fee $f_{i,j}$, denoting the amount of value charged by v_i for transferring a payment to v_j. For notational convenience, we let $f_{i,i} = 0$. Second, each edge $(v_i, v_j) \in \mathcal{E}$ has a channel balance $b_{i,j}$, denoting the amount of the remaining balance that v_i can transfer to v_j. Third, each edge $(v_i, v_j) \in \mathcal{E}$ also has an HTLC tolerance $\tau_{i,j}$, denoting the maximum time v_i would wait for the secret R from v_j. Note that we omit the transmission time in PCNs, because it is negligible compared to the transaction time on blockchain. For simplicity, we assume the set $\mathcal{E}$ only contains edges with positive balances at any time. An edge with zero balance is removed from the graph. In addition, we define an *(i, j) path* as a simple path from v_i to v_j.

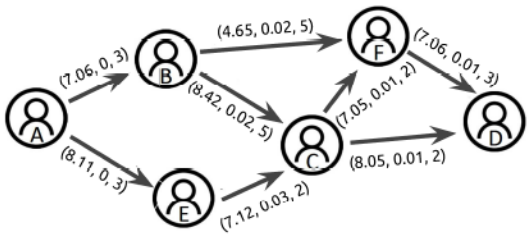


Fig. 4. Example of PCN. Each payment channel is associated with 3 attributes $(channel\ balance, transaction\ fee, HTLC\ tolerance)$.

We assume that each user only has local knowledge on all its incoming and outgoing edges, including their transaction fees, balances and HTLC tolerances. In general, each user cannot know the transaction fee, balance or HTLC tolerance of any remote edge, due to network asynchrony and dynamics.

4.2 Payment Model

A *payment request* is denoted by $R = (v_s, v_t, a)$, where v_s and v_t are the sender and recipient respectively, and a is the amount of the payment to be transferred. A payment request R is performed via a number of transactions through different channels, organized as an (s, t) path p denoted by a sequence $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$, where $v_0 = v_s$ and $v_L = v_t$. Here we abuse the notation $(v_i, v_{i+1}) \in p$ to represent that a channel (v_i, v_{i+1}) is involved in a payment path p. We use a transaction fee function $F_p(l, m)$ to denote the total transaction fee from v_l to v_m on a payment path p, where $0 \leq l < m \leq L$:

$$F_p(l,m) = \begin{cases} \sum_{i=l+1}^{m-1} f_{i,i+1}, & v_l = v_s, \\ \sum_{i=l}^{m-1} f_{i,i+1}, & v_l \neq v_s. \end{cases} \quad (1)$$

For a payment path, all the payment channels are called *involved channels* (ICs), and all the users except the sender and the recipient along the payment path are called *intermediate users* (IUs).

4.3 Design Goals

Now we propose the following desirable design goals that a payment routing protocol should satisfy.

- **Robustness**: A payment routing protocol satisfies robustness, if it generates two or more node-disjoint payment paths, where each of them can fulfill the payment request individually. In PCNs, a node may

become unresponsive due to external attacks, unexpected conditions or uncooperative behaviors, which leads to transaction failures. If the routing protocol generates only a single path for a payment request, it fails when a node on this path becomes unresponsive. In order to resist such transaction failures, it is necessary to generate more than one node-disjoint payment paths to transfer a payment, where no common intermediate user is shared on both paths. The payment is forwarded on these payment paths simultaneously. If one path fulfills the payment first, the other path(s) will be invalidated.

- **Approximate Optimization**: A payment routing protocol satisfies optimization, if it minimizes the worst-case transaction fee for a payment request. Since a robust payment routing protocol constructs two or more payment paths, where each payment path can fulfill the payment request, it is necessary to minimize the maximum transaction fee of these node-disjoint payment paths. However, this optimization problem could be NP-hard due to the distinct characteristics in PCNs. Sometimes we can only strive to achieve approximate optimization. We will discuss the NP-hardness of this problem in Section 4.4.
- **Efficiency**: A payment routing protocol satisfies efficiency, if it minimizes the routing and payment latency incurred by transmitting a payment through more than one path simultaneously. In the robust payment routing, only one payment path will be used to fulfill the payment request, and the other payment path(s) will be invalidated. Thus, it is necessary to guarantee that this payment path introduces the minimum latency.
- **Distributedness**: A payment routing protocol satisfies distributedness, if it dose not rely on a centralized trusted party. Centralized routing is subject to a single point of failures upon external attacks and hence cannot be trusted by users. Instead, users need to communicate with each other and conduct local computations to find routes for payments.

4.4 MTFM Problem Formulation

Following the desirable design goals that are outlined above, we consider the **Maximum Transaction Fee Minimization (MTFM)** problem for routing in PCNs. To formally formulate our studied problem, we introduce the following necessary concepts.

Timeliness Constraint: A payment path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ satisfies timeliness constraint, if the payment request can be successfully fulfilled within the HTLC tolerance of each IC, *i.e.*, $\tau_{i,i+1} \geq L-i, \forall i \in [0, L-1]$. Timeliness guarantees the commitment of honest processing at any IC.

Feasibility Constraint: A payment path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ satisfies feasibility constraint, if the payment request can be successfully transferred through each IC. Specifically, the balance $b_{i,i+1}$ of $e = (v_i, v_{i+1})$ should be at least the payment amount a plus the accumulation of transaction fees paid to the IUs that follow v_i on the path, *i.e.*, $b_{i,i+1} \geq a + F_p(i+1, L), \forall i \in [0, L-1]$.

Apparently, a single payment path cannot guarantee *robustness*, since some uncooperative IUs on the path may decide to become unresponsive or suffer from external attacks. In order to avoid such a transaction failure, we can establish a pair of node-disjoint payment paths, where either payment path can fulfill the given payment request. Since either payment path can be used to transfer a payment, it is necessary to guarantee *optimization*, which is to minimize the maximum transaction fee of the pair of node-disjoint payment paths, while still guaranteeing timeliness and feasibility constraints. Towards this goal, we consider the following optimization problem in PCNs:

Maximum Transaction Fee Minimization (MTFM): Given a payment request $R = (v_s, v_t, a)$, find a pair of timely and feasible node-disjoint payment paths, either of which can fulfill R, such that the maximum transaction fee of these two payment paths is minimized.

The MTFM problem can be proved NP-hard by reduction from the Min-Max 2-path problem, which has been proved to be NP-complete in [11].

Theorem 1. *The MTFM problem is NP-hard.*

Thus, we can only strive to achieve an approximation solution for the MTFM problem.

5 A Distributed Robust Payment Routing Protocol in PCNs

In this section, we present the design of RobustPay$^+$. We first provide the high-level overview and intuition behind RobustPay$^+$ and then follow the design goals that are outlined in Section 4.3 to design RobustPay$^+$ in detail.

5.1 Design Rationale and Challenges

A payment *transaction failure* occurs, if there are external attacks or unexpected conditions along the payment path. Thus, we provide robustness in payment routing by constructing a pair of node-disjoint payment paths, such that if a transaction failure occurs on one payment path, the other payment path can still fulfill the given payment request. Specifically, we design a distributed algorithm derived from the Suurballe's Algorithm [24] for the min-sum 2-path problem. To apply the Suurballe's Algorithm that was originally designed for centralized systems, we need to address several challenges. The first one is to transform the Suurballe's Algorithm into a distributed algorithm, where each node only has local knowledge. Second, the timeliness and feasibility constraints need to be taken into consideration.

In order to support robust payment routing, the original HTLC [16] needs modification, since it was originally designed for routing on a single payment path. This may cause a potential problem that the recipient claims the payment on both paths. To adapt the HTLC mechanism to robust payment routing, we need to modify the current HTLC protocol carefully to satisfy efficiency as well as off-chain security.

We address these challenges in the following subsections and design the RobustPay$^+$ protocol that users are expected to follow.

TABLE 1
Main notations

Notation	Meaning
$f_{i,j}$	amount of transaction fee of channel (v_i, v_j)
$b_{i,j}$	amount of remaining balance of channel (v_i, v_j)
$\tau_{i,j}$	HTLC tolerance of channel (v_i, v_j)
$\mathcal{IN}_i$	set of v_i's transferees
$\mathcal{OUT}_i$	set of v_i's transferors
$\Gamma^h(i,j)$	minimum transaction fee of the h-(i,j) TFM path
$\mathcal{N}_h(i,j)$	v_i's outgoing transferees of $\Gamma^h(i,j)$
$v_{i'}$	auxiliary node of v_i

TABLE 2
Example of Routing Table before Node Splitting

i	j				
A	B	C	D	E	F
$\Gamma^h(i,j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i,j)$	A	-	-	A	-
number of hops	1	-	-	1	-
feasible	✓	-	-	✓	-

5.2 Payment Path Construction

The first stage is to construct two payment paths for a payment request, such that either payment path can fulfill the payment request, and there is no *intermediate user* (IU) shared on both payment paths. Such two payment paths are referred to as a pair of node-disjoint payment paths. If a transaction failure occurs on a payment path due to unexpected conditions, the other payment path can still work to fulfill the transaction. In this section, we design and analyze RobustPay$^+$, a distributed approximation algorithm that determines a pair of node-disjoint payment paths.

5.2.1 Design Rationale

Before we formally describe the design of RobustPay$^+$, we introduce the main notations in Table 1 and necessary definitions in the following.

(i,j)-TFM path [33]: A payment path is an (i,j)-TFM path, if it satisfies the following conditions: 1) this payment path satisfies both the timeliness and feasibility constraints; 2) this payment path has the minimum transaction fee among all payment paths from v_i to v_j.

h-(i,j) TFM path: [33] A payment path is an h-(i,j) TFM path, if it satisfies the following conditions: 1) the length of this payment path is no more than h; 2) this payment path satisfies both the timeliness and feasibility constraints; 3) this payment path has the minimum transaction fee among all payment paths from v_i to v_j.

(i,j)-MTFM path pair: A pair of payment paths is an (i,j)-MTFM path pair, if it satisfies the following conditions:

1) these two payment paths satisfy both the timeliness and feasibility constraints;
2) these two payment paths do not share any common intermediate user (IU);
3) the maximum transaction fee of these two payment paths is the minimum among all pairs of payment paths from v_i to v_j.

Our algorithm RobustPay$^+$ is based on the distributed Suurballe's Algorithm [24], which was originally designed to minimize the total cost of two disjoint paths in a centralized system. Specifically, we design RobustPay$^+$ by implementing CheaPay [33], which generate a single (i,j)-TFM path in PCNs. RobustPay$^+$ algorithm consists of five stages: initialization, first routing, graph transformation, second routing and pair generation. In the initialization stage, the algorithm splits a node v_i into two nodes v_i and $v_{i'}$ by creating an auxiliary edge $(v_i, v_{i'})$ and reassigning all outgoing edges on v_i as outgoing edges on $v_{i'}$, because the Suurballe's Algorithm [24] was originally designed for finding a pair of edge-disjoint payment paths. In the first routing stage, the algorithm gives an (s,t)-TFM path by revoking CheaPay [33], which is an optimal distributed algorithm that minimizes the transaction fee of a payment path in PCNs while considering the timeliness and feasibility constraints. Note that this path is not one the two payment paths for final payment transaction. In the graph transformation stage, the algorithm transforms G to a residual graph. In the second routing stage, the algorithm outputs an (s,t)-TFM path in the residual graph, similarly to the first routing stage. In the pair generation stage, the algorithm generates a pair of node-disjoint payment paths by discarding the reversed edges in the second payment path from both payment paths and reconstructing the remaining edges.

Currently, source routing is utilized in the Lightning Network [5], where the sender node is responsible for calculating the entire path, from the sender to the recipient. To do so, the sender node needs to download a snapshot of the PCN topology to learn each channel's transaction fee and HTLC tolerance. Because the channel balance information is not public due to privacy concerns and varies over time due to dynamics, the transactions may fail as the actual balances on the channels may not satisfy the payment. Therefore, RobustPay$^+$ does not adopt source routing. Instead, each node makes distributed routing decisions based on a distributed Bellman-Ford style algorithm. In this setting, a node checks the balance availability with its neighbors during the stage of payment path construction.

RobustPay$^+$ establishes HTLCs after the payment path construction stage. Thus, collateral is not locked during path construction. Indeed, it is possible that transactions may fail due to dynamic balance change in the PCN. Although the main goal of this work is to resist transaction failures due to unresponsive nodes, RobustPay$^+$ can also mitigate transaction failures caused by dynamic balance change. This is because RobustPay$^+$ establishes a pair of node-disjoint paths that are guaranteed to be edge-disjoint. Even if there are edge failures along one path due to the dynamic change between the payment path construction stage and the payment forwarding stage, the other path can still be used for the payment transaction.

5.2.2 Design of RobustPay$^+$

In this section, we describe the details of RobustPay$^+$, which is illustrated in Algorithms 1, 2 and 3.

The initialization stage is shown in RobustPay$^+$-Init (Algorithm 1). RobustPay$^+$-Init splits a node v_i into two nodes by creating an auxiliary node $v_{i'}$ and reassigning all outgoing edges on v_i as outgoing edges on $v_{i'}$. We use $\mathcal{IN}_i$ and $\mathcal{OUT}_i$ to denote the set of v_i's ingoing transferors and the set of v_i's outgoing transferees, respectively (Lines 1 and 2). Thus, v_i is $v_{i'}$'s transferor, and $v_{i'}$ is v_i's transferee

TABLE 3
Example of Routing Tables After Node Splitting

i	j				
A	B	C	D	E	F
$\Gamma^h(i,j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i,j)$	A'	-	-	A'	-
number of hops	2	-	-	2	-
feasible	✓	-	-	✓	-

i'	j				
A'	B	C	D	E	F
$\Gamma^h(i',j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i',j)$	A'	-	-	A'	-
number of hops	1	-	-	1	-
feasible	✓	-	-	✓	-

(Lines 3 and 4). An auxiliary edge $(v_i, v_{i'})$ connects v_i and $v_{i'}$. Because there is no constraint to transfer a payment from v_i to its auxiliary node $v_{i'}$, we set $f_{i,i'} = 0$, $\tau_{i,i'} = \infty$, and $b_{i,i'} = \infty$ (Line 5). In order to reassign all outgoing edges on v_i as outgoing edges on $v_{i'}$, RobustPay$^+$-Init adds edges to connect $v_{i'}$ and v_i's transferees (Lines 6 to 11) and removes edges that connect v_i and v_i's transferees. We shall run Algorithm 1 to initialize the node splitting for each node $v_i \in \mathcal{V}$. Tables 2 and 3 show the example of routing tables that are stored on v_i before and after node splitting.

The first routing stage revokes CheaPay [33] to find an (s,t)-TFM path p_R^1 in G. Note this path is not one of the two payment paths for the final payment transaction. A transaction fee table $\{\Gamma(i,j)\}_{v_j \in \mathcal{V}}$ is generated on each node v_i, which stores the transaction fees of the (i,j)-TFM paths. Therefore, RobustPay$^+$ can reuse the transaction fee table $\{\Gamma(s,i)\}_{v_i \in \mathcal{V}}$ for graph transformation in the next stage.

The graph transformation stage is shown in RobustPay$^+$-Trans (Algorithm 2). RobustPay$^+$-Trans transforms the original graph to a residual graph by reusing the transaction fee table $\{\Gamma(s,i)\}_{v_i \in \mathcal{V}}$ obtained from the previous stage. First, v_i requests $\Gamma(s,i)$ from its transferor along the (s,i)-TFM path (Lines 1 to 3). Then, v_i modifies the transaction fee of each edge (i',j) by replacing $f_{i,j}$ by $\tilde{f}_{i',j} = f_{i',j} - \Gamma(s,j) + \Gamma(s,i')$ (Line 5). Also, if the modified transaction fee of an edge on path p_R^1 is 0, RobustPay$^+$-Trans reverses the edge direction (Lines 6 to 8).

The final stage is shown in RobustPay$^+$-Output (Algorithm 3). RobustPay$^+$-Output outputs the (s,t)-MTFM path pair. First, RobustPay$^+$-Output runs RobustPay$^+$-Init to initialize the auxiliary node $v_{i'}$ on each node $v_i \in \mathcal{V}$. Second, RobustPay$^+$-Output revokes CheaPay to generate an (s,t)-TFM path in G. Third, RobustPay$^+$-Output runs RobustPay$^+$-Trans on each node $v_i \in \mathcal{V}$ to transform the original graph to a residual graph G_v. Then, RobustPay$^+$-Output revokes CheaPay again to generate an (s,t)-TFM path in G_v. Finally, RobustPay$^+$-Output outputs the (s,t)-MTFM path pair by discarding the reversed edges in the second payment path from both payment paths and reconstructing the remaining edges.

5.2.3 *Analysis of* RobustPay$^+$

In this subsection, we analyze the approximation ratio of RobustPay$^+$ as follows.

Theorem 2. RobustPay$^+$ *outputs a 2-approximation solution to the MTFM problem.*

Algorithm 1: RobustPay$^+$-Init

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a node v_i and an auxiliary node $v_{i'}$.
1 $\mathcal{IN}_i \leftarrow$ the set of v_i's transferor nodes;
2 $\mathcal{OUT}_i \leftarrow$ the set of v_i's transferee nodes;
3 Create an auxiliary node $v_{i'}$;
4 $\mathcal{OUT}_i \leftarrow \mathcal{OUT}_i \cup \{v_{i'}\}$; $\mathcal{OUT}_{i'} \leftarrow \emptyset$; $\mathcal{IN}_{i'} \leftarrow \{v_i\}$;
5 $f_{i,i'} \leftarrow 0$; $\tau_{i,i'} \leftarrow \infty$; $b_{i,i'} \leftarrow \infty$;
6 **for** $v_j \in \mathcal{OUT}_i$ **do**
7 $\quad f_{i',j} \leftarrow f_{i,j}$; $f_{i,j} \leftarrow \infty$;
8 $\quad \tau_{i',j} \leftarrow \tau_{i,j}$; $\tau_{i,j} \leftarrow 0$;
9 $\quad b_{i',j} \leftarrow b_{i,j}$; $b_{i,j} \leftarrow 0$;
10 $\quad \mathcal{OUT}_i \leftarrow \mathcal{OUT}_i \backslash \{v_j\}$; $\mathcal{OUT}_{i'} \leftarrow \mathcal{OUT}_{i'} \cup \{v_j\}$;
11 **end**
12 **return** $v_{i'}$

Algorithm 2: RobustPay$^+$-Trans

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a residual graph G_t.
1 **for** $v_k \in \mathcal{IN}_i$ **do**
2 $\quad$ Request $\Gamma(s,i)$ along the (s,i)-TFM path;
3 **end**
4 **for** $v_j \in \mathcal{OUT}_{i'}$ **do**
5 $\quad f_{i',j} \leftarrow f_{i',j} - \Gamma(s,j) + \Gamma(s,i')$;
6 $\quad$ **if** $f_{i',j} == 0$ *and* $(v_{i'}, v_j) \in p_R^1$ **then**
7 $\quad\quad f_{j,i'} \leftarrow 0$; $f_{i',j} \leftarrow \infty$;
8 $\quad$ **end**
9 **end**
10 **return** G_v

Proof. We first prove that RobustPay$^+$ outputs an optimal solution to the following Min-Sum Transaction Fee (MSTF) problem: *Given a payment request $R = (v_s, v_t, a)$ and a PCN $G = (\mathcal{V}, \mathcal{E})$, find a pair of timely and feasible node-disjoint paths, either of which can fulfill R, such that the summation transaction fee of these two paths is minimized.*

Because RobustPay$^+$ is designed based on a distributed version of the Suurballe's Algorithm [24], RobustPay$^+$ outputs an optimal solution to the MSTF problem.

Let $\{p_1^*, p_2^*\}$ be the pair of paths in an optimal solution to the MTFM problem and $OPT = \max\{F_{p_1^*}, F_{p_2^*}\}$. Let $\{p_1, p_2\}$ be the pair of paths generated by RobustPay$^+$. Thus, we can get

$$\begin{aligned} 2\max\{F_{p_1^*}, F_{p_2^*}\} &\geq F_{p_1^*} + F_{p_2^*} \\ &\geq F_{p_1} + F_{p_2} \\ &\geq \max\{F_{p_1}, F_{p_2}\}, \\ 2OPT \geq 2\max\{F_{p_1^*}, F_{p_2^*}\} &\geq \max\{F_{p_1}, F_{p_2}\}. \end{aligned}$$

Therefore, RobustPay$^+$ outputs a 2-approximation solution to the MTFM problem. ■

We now analyze the message complexity of RobustPay$^+$. First, RobustPay$^+$-Init builds $O(|\mathcal{V}|)$ auxiliary nodes, where $|\mathcal{V}|$ is the total number of nodes. Second, RobustPay$^+$ calls CheaPay once, whose message complexity is $O(|\mathcal{V}|^2|\mathcal{E}|)$. Then RobustPay$^+$-Trans sends messages along each (s,i)-TFM path. Therefore, RobustPay$^+$-Trans exchanges $O(|\mathcal{E}|)$

Algorithm 3: RobustPay$^+$-Output

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a payment request $R = (v_s, v_t, a)$.
Output: a payment path p_R.

$p_R^1 \leftarrow \emptyset$; $p_R^2 \leftarrow \emptyset$;
for $v_i \in \mathcal{V}$ **do** Init(G, v_i);
$p_R^1 \leftarrow$ CheaPay(G, R);
for $v_i \in \mathcal{V}$ **do** Trans(G, v_i);
$p_R^2 \leftarrow$ CheaPay(G, R);
for $(v_i, v_j) \in p_R^2$ **do**
  if $(v_j, v_i) \in p_R^1$ **then**
    $p_R^1 \leftarrow p_R^1 \backslash \{(v_j, v_i)\}$; $p_R^2 \leftarrow p_R^2 \backslash \{(v_i, v_j)\}$;
  end
end
Reconstruct the remaining edges of p_R^1, p_R^2;
return p_R^1, p_R^2

messages on each node. RobustPay$^+$-Output's message complexity is dominated by CheaPay and RobustPay$^+$-Trans on each node, which is $O(|\mathcal{V}|^2|\mathcal{E}|)$. In total, the overall message complexity of RobustPay$^+$ is $O(|\mathcal{V}|^2|\mathcal{E}|)$.

5.3 HTLC Establishment

A Hashed Time-Locked Contract (HTLC) is a script that permits a designated party (the transferee) to spend funds by disclosing the preimage of a hash. It also permits a second party (the transferor) to spend the funds after a timeout is reached, in a refund situation. The original HTLC introduced in [16] was designed for payment routing in a single payment path. In the HTLC, the on hold payment is refunded to the transferor, only if the transferee does not provide the preimage of H within the HTLC tolerance. However, the HTLC does not provide the transferee with flexible choices to cancel a transaction before the expiration. Even if the transferee decides to cancel a transaction, it can only wait until the expiration of the HTLC.

To adapt the HTLC protocol to RobustPay$^+$, we modify the original HTLC to provide more flexible choices as follows: If the transferee does not provide the preimage of H within the HTLC tolerance, or if the transferee cancels the transaction before the preimage of H is provided, the on hold payment in the HTLC is refunded to the transferor. A potential problem is that the recipient may claim the payment on both paths. In order to prevent this double-claim issue, we improve the HTLC by using two separate secrets on two payment paths, inspired by Boomerang [6]. However, we cannot directly apply Boomerang. Because Boomerang cannot prevent the sender from reverting both payments, since it only guarantees that the sender can revert all payments, if the recipient claims more than one payment. Thus, we modify it to prevent the double-reverting issue, such that only one of the payment paths is randomly selected to be reversible. With this modification, the sender can revert the payment on the reversible path by providing the secret of the irreversible path, if the recipient claims the payment twice by producing secrets on both paths.

The script of the modified HTLC takes the following form, and the modification of the HTLC is highlighted:

```
OP_IF
    OP_IF
        [HASHOP] ⟨digest⟩ OP_DROP OP_DUP
        OP_HASH160 ⟨buyer pubkey hash⟩
    OP_ELSE
        ⟨num⟩ [TIMEOUTOP] OP_EQUALVERIFY
        OP_DUP OP_HASH160 ⟨seller pubkey hash⟩
    OP_ENDIF
OP_NOTIF
    [CANCELOP] ⟨digest⟩ OP_DROP OP_DUP
    OP_HASH160 ⟨seller pubkey hash⟩
OP_ELSE
    ⟨num⟩ [TIMEOUTOP] OP_DROP OP_DUP
    OP_HASH160 ⟨buyer pubkey hash⟩
OP_ENDIF
OP_EQUALVERIFY
OP_CHECKSIG
```

A simple illustration of the modified HTLC is shown in Fig. 5. Such a modification on HTLC can provide flexible choices for PCN users and security for senders. We formally give the following security guarantee:

Theorem 3. RobustPay$^+$ *guarantees that the recipient cannot claim the payment twice, and the sender cannot revert both payments.*

Proof. Let $\{p_1, p_2\}$ be a pair of node-disjoint payment paths generated by RobustPay$^+$. Without loss of generality, let p_1 be the irreversible path, and let p_2 be the reversible path. Let R_1 and R_2 be the preimages for the recipient to claim the payments on p_1 and p_2, respectively. Assume that the recipient claims the payments on both p_1 and p_2 by providing R_1 and R_2. This indicates the sender knows both R_1 and R_2. By providing R_1 on p_2, the sender can revert the payment transaction on p_2. Thus, it guarantees that the recipient cannot claim the payment twice, and the sender cannot revert both payments. ■

5.4 Payment Forwarding

After the payment path construction and the HTLC establishment processes, the sender can forward the payment to the recipient via the constructed payment paths. Once one of the two payment paths successfully transfers the payment to the recipient, the other payment path should be invalidated. If the recipient tries to claim the payment on the second path, the sender can revert the payment on the first path. A simple illustration of the payment forwarding is shown in Fig. 6. Two node-disjoint payment paths have been constructed in the previous stage, where A is the sender and D is the recipient. An HTLC has been created on each IC on both payment paths. Since D receives the HTLC from G on the lower (red) payment path earlier, D provides the preimage of H to C and receives the payment from C. Therefore, the upper (green) payment path is invalidated. D can choose to cancel the transaction from C to D. The on hold payment in the HTLC between C and D is refunded to C. So are the rest on hold payments in the HTLCs on the ICs along the upper (green) payment path. If the D tries to claim the payment on the upper (green) path, the A can revert the payment on the lower (red) path by providing the preimage of H, which indicates that the payment has be claimed. This guarantees that the payment cannot be double claimed.

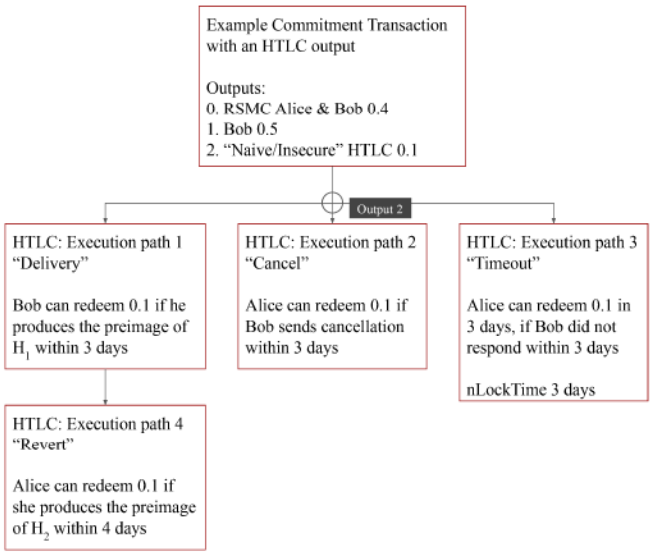


Fig. 5. Modified hashed time-lock contract (HTLC). Alice sends a payment of 0.1 to the Bob via B and C with an HTLC tolerance of 3. Note that there are two possible spends from an HTLC output. If Bob can produce the preimage of H_1 within 3 days, Bob can redeem path 1. If Alice can produce the preimage of H_2 after Bob produces the preimage of H within 4 days, Alice can redeem path 4. If Alice sends cancellation before Bob can produce the preimage of H_1 within 3 days, Alice can redeem path 2. After 3 days, Alice is able to redeem path 3, if there is no response from Bob.

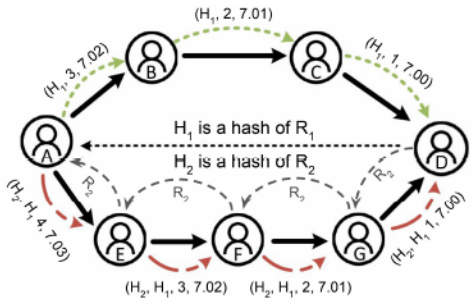


Fig. 6. Payment Forwarding in RobustPay$^+$. The sender (A) sends a payment of 7 to the recipient (D). Two node-disjoint payment paths are $A \to B \to C \to D$ and $A \to E \to F \to G \to D$. HTLCs are established on both payment paths simultaneously, from A to D, sequentially. The upper (green) path is not reversible, and the lower (red) path is reversible. D provides the preimage of H to G on the lower (red) payment path and refunds C on the upper (green) payment path.

6 Performance Evaluation

In this section, we evaluate the performance of RobustPay$^+$. As we surveyed in Section 2, there is no existing payment routing protocol that satisfies robustness, optimization, efficiency or distributedness in payment channel networks. Therefore, we demonstrate the performance of RobustPay$^+$ by comparing it to CheaPay [33] and baseline algorithms.

6.1 Environment Setup

We implemented and modified a simulator for PCNs [33] to model the transaction arrivals and settlements. Transactions are serial and routed according to the routing algorithms, if the timeliness and feasibility constraints are satisfied on the generated payment paths. The locked funds are unavailable for use by any node on the payment path. When a payment transaction is settled, these funds are released. The simulator supports payment transactions through a queue of pending payments. The queue is periodically polled to check if the transactions can progress further. The HTLC tolerance parameter is specified on each channel independently by the transferor on this channel, which represents the maximum time that the transferor is willing to wait for the preimage to confirm a transaction. Because the preimage is sent backwards from the recipient, the HTLC timeout parameter indicates the maximum distance (the number of hops) from a node to the recipient. Thus, the HTLC timeout parameter is at channel level rather than at a source-destination pair level. Since the HTLC timeout information is public in the Lightning Network, we use the real data in the simulation.

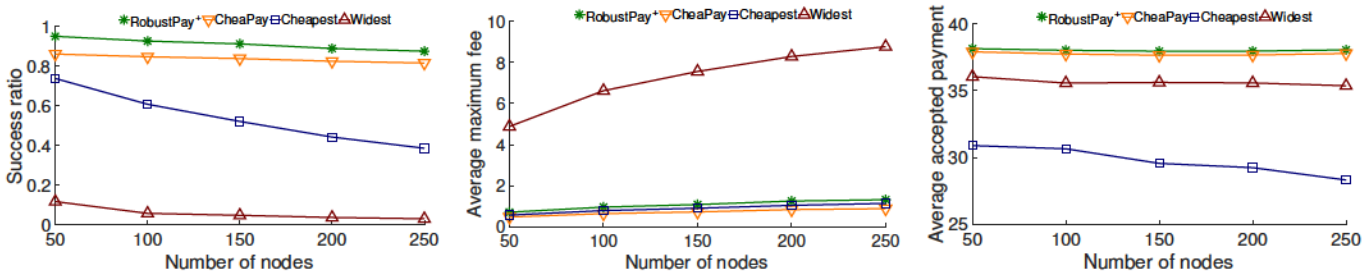
We obtained a real-world PCN topology from the Bitcoin Lightning Network [5]. In particular, we crawled a snapshot topology of the Lightning Network on July 14, 2020 [4]. To crawl the Lightning Network, we ran the Bitcoin Core daemon (bitcoind) [1], built a c-lightning [3] node on mainnet, and connected it to an existing Lightning node [2]. The network consists of $5,622$ nodes and $65,628$ channels. We used a real-world transaction dataset sampled from the path-based transaction network Ripple [20, 22]. To evaluate the impact of the network size, we extracted connected induced subgraphs. We assume that the transaction fees of all payment channels are distributed uniformly at random over $(0, 1]$. We compare RobustPay$^+$ to the following algorithms:

- CheaPay [33]: It finds a single payment path that minimizes the total transaction fee, while satisfying both the timeliness and feasibility constraints.
- Cheapest: First, it finds a payment path that minimizes the total transaction fee. Second, it finds a node-disjoint path that minimizes the total transaction fee by removing the intermediate node in the first path. Then, it checks if both paths satisfy both the timeliness and feasibility constraints. If both constraints are satisfied, the payment is accepted. Otherwise, the payment is rejected.
- Widest: First, it finds a payment path that maximizes the minimize channel balance on a path. Second, it finds a node-disjoint path that maximizes the minimize channel balance on a path by removing the intermediate node in the first path. Then, it checks if both paths satisfy both the timeliness and feasibility constraints. If both constraints are satisfied, the payment is accepted. Otherwise, the payment is rejected.

6.2 Performance Metrics

We use the following metrics for performance evaluation:

- **Success ratio:** The percentage of accepted payment requests.
- **Average maximum transaction fee:** Average maximum transaction fee of the pair of paths over all accepted payment requests.
- **Average accepted payment:** Average payment over all accepted payment requests.



(a) Success ratio (b) Average maximum transaction fee (c) Average accepted payment

Fig. 7. Impact of number of nodes on RobustPay$^+$, CheaPay, Cheapest, and Widest.

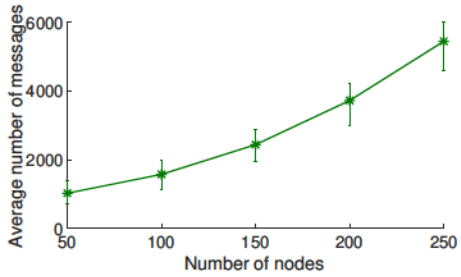


Fig. 8. Message complexity of RobustPay$^+$

6.3 Evaluation of RobustPay$^+$

Fig. 7 shows success ratios, average maximum transaction fees and average accepted payments of RobustPay$^+$, CheaPay, Cheapest and Widest.

Fig. 7(a) shows the comparison of success ratios achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. RobustPay$^+$ outperforms other algorithms, due to its robustness, timeliness, and feasibility guarantees. CheaPay gives a lower success ratio than RobustPay$^+$, because CheaPay generates a single path and does not provide robustness. We can witness the growing gap between RobustPay$^+$ and Cheapest, which indicates that focusing only on minimizing the total transaction fee without considering the timeliness and feasibility constrains decreases the success ratio significantly. Widest describes how well an algorithm can do without taking into account the minimization of maximum transaction fee, timeliness or feasibility constraint. As expected, Widest gives the worst success ratio, because Widest only focuses on maximizing the minimum channel balance on a path, which possibly makes the path non-timely or infeasible. All algorithms have dropping success ratios with more nodes. This is because although the number of nodes increases, the percentage of paths that satisfy both the timeliness and tolerance constraints decreases.

Fig. 7(b) shows the comparison of the average maximum transaction fees achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. Cheapest gives a slightly lower average maximum transaction fee than RobustPay$^+$, because Cheapest has a much lower success ratio and intends to accept paths with low transaction fees. We can witness the gap between RobustPay$^+$ and CheaPay, which indicates that RobustPay$^+$ sacrifices a little bit of transaction fee minimization for robustness. Widest gives the highest average maximum transaction fee, because Widest only focuses on maximizing the minimum channel balance on a path, but ignores minimizing the transaction fee.

Fig. 7(c) shows the comparison of average accepted payments achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. RobustPay$^+$ outperforms the other algorithms in terms of the average accepted payment, due to its robustness, timeliness, and feasibility guarantees. CheaPay gives a slightly lower average accepted payment than RobustPay$^+$, because CheaPay does not provide robustness. The average accepted payment of Widest drops with more nodes, because the minimum channel balances of paths decrease.

We also evaluate the convergence speed of RobustPay$^+$. The result is shown in Fig. 8. We can observe that the average number of messages increases with the the number of nodes. Since RobustPay$^+$ implements a variant of the distributed Suurballe's Algorithm [24], which is based on the distributed Bellman-Ford Algorithm [7, 10], RobustPay$^+$ has the growing trend of message complexity similar to that of Bellman-Ford.

7 Conclusion and Future Work

In this paper, we investigated the robust payment routing protocol to resist payment transaction failures in PCNs. We first suggested a set of crucial design goals for payment routing, which are referred to as robustness, efficiency, distributedness and approximate optimization. Following these design goals, we presented a distributed robust payment routing protocol RobustPay$^+$ consisting of three stages: Payment Path Construction, HTLC Establishment and Payment Forwarding. For Payment Path Construction, RobustPay$^+$ achieved robustness by constructing two payment paths, where either payment path can fulfill the payment request. To guarantee approximate optimization, we formulated the Maximum Transaction Fee Minimization (MTFM) problem and presented a distributed 2-approximation algorithm RobustPay$^+$. Moreover, we modified the original HTLC protocol to provide efficiency and robustness and adapted it to the robust payment routing protocol. Extensive simulations demonstrated that RobustPay$^+$ achieved outstanding success ratio and average acceptance value compared to baseline algorithms.

One future direction that we can work on is to improve privacy in payment routing. As studied in [25, 28], attackers can discover a large portion of channel balances by systematically probing for payment paths. The attack experiments have been conducted in the current LN setting

that adopts source routing. Therefore, a higher degree of privacy is desired for routing in PCNs. On the one hand, If an external attacker could probe intermediate channel balances, it would break the relationship anonymity and channel balance privacy. On the other hand, knowing channel balances would allow senders to avoid trying routes and thus improve transaction success ratio. Thus, it is worth studying the trade-off between routing efficiency and privacy in PCNs as the future work.

REFERENCES

[1] "Bitcoin Core daemon (bitcoind)." [Online]. Available: https://github.com/bitcoin/bitcoin/

[2] "Bitstamp's Lightning Network node." [Online]. Available: https://www.bitstamp.net/lightning-network-node/

[3] "c-lightning Daemon." [Online]. Available: https://github.com/ElementsProject/lightning/tree/master/lightningd/

[4] "Lightning Network Dataset." [Online]. Available: https://people.mines.edu/djyang/research/project-pcn/

[5] "The Lightning Network." [Online]. Available: https://lightning.network/

[6] V. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment networks," *arXiv preprint arXiv:1910.01834*, 2019.

[7] R. Bellman, "On a routing problem," in *Quarterly of applied mathematics*, vol. 16, no. 1, 1958, pp. 87–90.

[8] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *FC*. Springer, 2016, pp. 106–125.

[9] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *SSS*. Springer, 2015, pp. 3–18.

[10] L. R. Ford Jr, "Network flow theory," RAND CORP SANTA MONICA CA, Tech. Rep., 1956.

[11] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Mathematics*, vol. 26, no. 1, pp. 105–115, 1990.

[12] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in decentralized credit networks." in *IACR Cryptology ePrint Archive*, 2016, pp. 1054–1071.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." Working Paper, 2008.

[14] R. Network. [Online]. Available: https://raiden.network/

[15] C. H. Papadimitriou and D. Ratajczak, "On a conjecture related to geometric routing," in *Theoretical Computer Science*, vol. 344, no. 1. Elsevier, 2005, pp. 3–14.

[16] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016. [Online]. Available: lightning.network/lightning-network-paper.pdf

[17] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," in *Whitepaper*, 2016.

[18] E. Project. [Online]. Available: https://www.ethereum.org/

[19] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[20] Ripple. [Online]. Available: https://www.ripple.com/

[21] E. Rohrer, J.-F. Laß, and F. Tschorsch, "Towards a concurrent and distributed route selection for payment channel networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 411–419.

[22] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *NDSS*, 2017.

[23] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 29–35.

[24] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[25] S. Tikhomirov, R. Pickhardt, A. Biryukov, and M. Nowostawski, "Probing channel balances in the lightning network," *arXiv preprint arXiv:2004.00333*, 2020.

[26] M. Trillo, "Stress test prepares visanet for the most wonderful time of the year (2013)," 2013.

[27] P. F. Tsuchiya, "The landmark hierarchy: a new hierarchy for routing in very ltaiarge networks," in *SIGCOMM*, vol. 18, no. 4. ACM, 1988, pp. 35–42.

[28] G. van Dam, R. A. Kadir, P. N. Nohuddin, and H. B. Zaman, "Improvements of the balance discovery attack on lightning network payment channels," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2020, pp. 313–323.

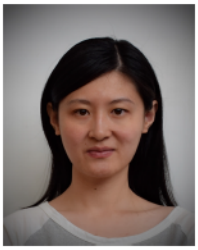
[29] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "Enforcing private data usage control with blockchain and attested off-chain contract execution," *arXiv preprint arXiv:1904.07275*, 2019.

[30] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *arXiv preprint arXiv:1904.04098*, 2019.

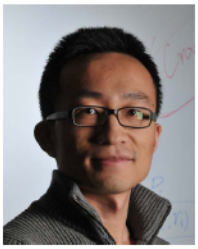
[31] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks," in *ICCCN*. IEEE, 2018, pp. 1–9.

[32] Y. Zhang and D. Yang, "Robustpay: Robust payment routing protocol in blockchain-based payment channel networks," in *ICNP*. IEEE, 2019.

[33] Y. Zhang, D. Yang, and G. Xue, "Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks," in *ICC*. IEEE, 2019.



Yuhui Zhang (S'16) received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2011. She is currently working toward the Ph.D. degree in Computer Science at Colorado School of Mines, Golden, CO, USA. Her main research interests lie in the areas of blockchain, game theory, location privacy, and crowdsourcing.



Dejun Yang (M'13–SM'19) received the B.S. degree in computer science from Peking University, Beijing, China, in 2007 and the Ph.D. degree in computer science from Arizona State University, Tempe, AZ, USA, in 2013.

Currently, he is an Associate Professor of computer science with Colorado School of Mines, Golden, CO, USA. His research interests include Internet of things, networking, and mobile sensing and computing with a focus on the application of game theory, optimization, algorithm design, and machine learning to resource allocation, security and privacy problems.

Prof. Yang has served as the TPC Vice-Chair for Information Systems for IEEE International Conference on Computer Communications (INFOCOM) and currently serves an Associate Editor for the IEEE Internet of Things Journal. He has received the IEEE Communications Society William R. Bennett Prize in 2019 (best paper award for IEEE/ACM Transactions on Networking and IEEE Transactions on Network and Service Management in the previous three years), Best Paper Awards at the IEEE Global Communications Conference (2015), the IEEE International Conference on Mobile Ad hoc and Sensor Systems (2011), and the IEEE International Conference on Communications (2011 and 2012), as well as a Best Paper Award Runner-up at the IEEE International Conference on Network Protocols (ICNP) (2010).