RACED: Routing in PAyment Channel NEtworks Using Distributed Hash Tables

Kartick Kolachala
New Mexico State University
Las Cruces, NM, USA
kart1712@nmsu.edu

Mohammed Ababneh
New Mexico State University
Las Cruces, NM, USA
mababneh@nmsu.edu

Roopa Vishwanathan
New Mexico State University
Las Cruces, NM, USA
roopav@nmsu.edu

ABSTRACT

The Bitcoin scalability problem has led to the development of off-chain financial mechanisms such as payment channel networks (PCNs) which help users process transactions of varying amounts, including micro-payment transactions, without writing each transaction to the blockchain. Since PCNs only allow path-based transactions, effective, secure routing protocols that find a path between a sender and receiver are fundamental to PCN operations. In this paper, we propose *RACED*, a routing protocol that leverages the idea of Distributed Hash Tables (DHTs) to route transactions in PCNs in a fast and secure way. Our experiments on real-world transaction datasets show that *RACED* gives an average transaction success ratio of 98.74%, an average pathfinding time of 31.242 seconds, which is 1.65×10^3, 1.8×10^3, and 4×10^2 times faster than three other recent routing protocols that offer comparable security/privacy properties. We rigorously analyze and prove the security of *RACED* in the Universal Composability framework.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Distributed systems security**; **Security protocols**.

ACM Reference Format:
Kartick Kolachala, Mohammed Ababneh, and Roopa Vishwanathan. 2024. RACED: Routing in PAyment Channel NEtworks Using Distributed Hash Tables. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3634737.3637653

1 INTRODUCTION

The development of cryptocurrencies, which began with the Bitcoin white paper [39] in 2009, has disrupted banking and financial processes across the globe. As of February 2023, Bitcoin's market capitalization stands at 453 Billion USD [10]. However, the throughput of transactions involving cryptocurrencies is extremely low due to the high latency of transaction confirmation on the blockchain. For instance, the transaction processing speed of Bitcoin is 5-7 transactions per second and that of Ethereum is 15-30 transactions per second [7–9, 11]. This is in sharp contrast with traditional fiat currency's throughput, e.g., Visa processes over 65,000 transactions per second [60]. One of the most promising solutions to this problem is off-chain payment channels. Two parties create a payment channel on a blockchain with some initial balance, following which they can send an unlimited number of payments to each other using that channel without writing anything to the blockchain. Access to the blockchain is only needed either if there is a dispute or the two parties involved decide to close the channel.

This idea can be extended to enable transactions between two parties that may not have a payment channel currently open between them. Decentralized payment channel networks (PCNs) that enable transitive payments have been proposed such as [19, 26, 35, 37, 51], where two unconnected users can send/receive payments if there exists a path comprising of several users with payment channels between them. The first such network was the Lightning Network, which operates on top of the Bitcoin blockchain [26]. Lately, Lightning Network has become one of the fastest-growing PCNs. Between January 2021 and December 2021, there were a total of 28 Million unique channels opened in the Lightning Network, with an average of 73,733 new channels created every day. The number of unique nodes (unique public key pairs) involved in channel opening during this period was 6.5 Million [12]. The market capitalization of Lightning Network is USD 1 Million as of 2023. Several other payment channel networks and credit networks have been developed, which have later evolved into blockchain-based decentralized financial ecosystems, such as Ripple [44], which has a current market value of 20 Billion USD [47], (increased from 9.97 Billion USD in 2017 and peaked at 64 Billion USD in April 2021) and Stellar [55]. Between January 2021 to December 2021, there were a total of 15 Million transactions recorded on the Ripple ledger, with an average of 1 Million transactions recorded every month [45]. These numbers indicate the size and growth of PCNs.

A major advantage of PCNs is that they facilitate micro-payments between users that can be as small as 10^{-7} BTC [30]. Apart from this, the fees charged by PCNs to route payments are a fraction of the on-chain transaction fees charged by the underlying blockchain. The problem of finding an efficient route between a sender and receiver in a PCN is challenging and has attracted considerable attention from the research community [22, 35, 42, 43, 61]. While there have been many elegant routing protocols developed recently for PCNs, each one comes with its own set of limitations. Some routing protocols do not provide security of transactions nor privacy of the users [51, 61, 62], while others do not support concurrent transactions [29, 35, 53, 61, 62]. Some routing protocols need trusted entities to route payments [35], while others implement source routing, in which the network topology needs to be known to all nodes [61]. In this paper, we present a novel routing mechanism

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07.
https://doi.org/10.1145/3634737.3637653

called *RACED* that uses distributed hash tables (DHT) to route the payment from the sender to the receiver in PCNs.

Current routing protocols for PCNs traverse the entire network in the worst case to route a payment from a sender to a receiver, with a maximum path length of $n-1$ hops, in a network of n nodes. Using DHTs will help us in reducing this path length since the complexity of locating any node in a DHT is logarithmic in the number of nodes in the DHT. This consequently reduces the overall pathfinding time (time taken to find paths between two nodes in the network) and routing time (time taken to route the payment).

All PCN routing protocols have an overall routing complexity that is *linear* in the number of nodes in the PCN in the worst case. Reducing this bound to *sub-linear* while preserving the *privacy of nodes and network topology* and also ensuring the *atomicity* of the payments is a significant research challenge.

Our contributions are:

1) We design an efficient decentralized routing protocol, *RACED* with no trusted entities, using DHTs to reduce the routing time from $O(n)$ to $O(\log r + u)$, where n is the total number of users/nodes in a PCN, r is the total number of routing helpers (untrusted nodes that aid transaction routing), and u is the number of non-routing helper nodes.

2) *RACED* preserves the privacy of nodes and their channel balances, as well as maintains privacy of the network topology.

3) We experimentally demonstrate the scalability and efficiency of *RACED* using transaction data from the Ripple network [45], and prove its security in the Universal Composability (UC) framework.

Outline: In Section 2 we discuss relevant related work, in Sections 3 and 4, we explain our system and adversary models respectively. In Sections 5 and 6 we present the construction of *RACED*, in Section 7, we present our experimental evaluation. In Section 8 we analyze the security of *RACED* in the UC framework, and in Section 9 we conclude the paper.

2 RELATED WORK

Routing protocols with security guarantees: The main security property that we want for routing protocols in PCNs is that honest parties should not lose funds because of malicious behavior by other parties in the system. To this end Malavolta *et al.* [35] proposed a routing protocol leveraging trusted entities called *landmarks* to provide secure routing between the sender and the receiver. The landmark finds a path between itself and the sender and itself and the receiver; these sub-paths are combined to get the full path. The idea of using untrusted entities to facilitate routing has been proposed by Panwar *et al.* in [42] that uses a set of well-connected nodes called *routing helpers* to facilitate routing. However, this protocol has a very high communication overhead during the pathfinding phase, in addition to using the blockchain as an auditing mechanism which makes it very expensive to deploy in the real-world.

Roos *et al.* proposed a routing mechanism in [51] that uses graph embedding, where the routing is carried out by constructing a spanning tree of the entire network. While this work improved upon [35] by supporting concurrent transactions, the sender picks a random amount to be transmitted along a path without knowing whether the path has sufficient liquidity, which could lead to a high rate of transaction failure. Besides, frequently needing to update the embedding for a dynamic network topology results in a heavy computational overhead. The routing protocol proposed by Pietrzak *et al.* in [43] uses the idea of Private Information Retrieval (PIR). The shortest paths between all the nodes are computed and stored in trusted servers which incur a large storage overhead. A honest majority is assumed among the servers. When a payment needs to be routed, the sender queries these trusted servers for the available list of shortest paths to the intended receiver. This would also require the sender to download the complete network topology. The protocol proposed by Subramanian *et al.* [57] leverages the idea of distributed hash tables to replenish the depleted link weights of nodes in a PCN, in a process called *rebalancing*, and does not focus on pathfinding or routing of transactions, hence their work is orthogonal to *RACED*. None of the aforementioned works can route transactions in disjoint graphs.

Routing protocols with no privacy/security guarantees: There are a few works that use breadth first search (BFS) or max-flow algorithms to design routing protocols for PCNs [22, 29, 62] but do not provide security/privacy of nodes in the PCN. Besides, using traditional max-flow algorithms such as Ford-Fulkerson (implemented using Edmonds-Karp method) and Goldberg-Tarjan algorithms incur significant overheads of $O(|V||E|^2)$ [15] and $O(|V|^3)$ [21] respectively, in a graph $G(V, E)$, which is not scalable to large PCNs. The ideas proposed by Abdelrahman *et al.* in [2–4] present distributed versions of Dijkstra's shortest path algorithm, and the minimum cost flow problem, both of which can be potentially applied to perform routing in PCNs. The distributed version of Dijkstra's shortest path algorithm has a computational complexity of $O(|V|^2) + O(|V|)$, which makes it non-scalable to large scale PCNs. The computational complexity of the distributed version of the minimum cost-flow problem is $O(|V|^8 log(|V|))$ and the communication complexity is $O(|V|^{10} log(|V|))$, which makes it infeasible to be applied for large scale PCNs. Due to space constraints, we give a detailed descriptions of the ideas proposed in [2–4] in Appendix A.

The idea proposed in [24] by Kadry *et al.* uses a machine learning-based approach to find a path between the sender and receiver. This work does not focus on route discovery but instead focuses on selecting the best path amongst the ones that have already been chosen using BFS. The work proposed in [53] uses buffers (called Spider Routers) in the form of queues to store and route transactions. However, it makes transactions wait for an indefinite amount of time before they are routed, besides it does not take into account the privacy of the nodes involved in a transaction. Other works such as [64] and [17] have been proposed that do not provide privacy of nodes in the PCN.

RobustPay+ [66] and its preliminary version, Robustpay [65] focus on building a routing protocol for PCNs that supports multiple paths from a sender to a receiver from which the sender chooses only one path to route the payment. Their main contribution lies in constructing multiple paths such that there is no overlap in terms of nodes between any pair of paths. This is done to prevent transaction failures caused by nodes becoming unresponsive or going offline in the PCN. The idea proposed by Chen *et al.*, MPCN-RP [14], focuses on building a source routing protocol that minimizes the transaction fees. This protocol presents a modified version of Dijkstra's algorithm, in which the length of the path (in terms of hop-count) is taken into consideration along with the edge weights.

Table 1: Comparison of Routing Protocols in PCNs

Routing protocols	Concurrency	Privacy	Balance security	Topology privacy	Avoids source routing	Decentralized	Atomicity	Disjoint graphs
MPCN-RP [14]	✗	✗	✓	✗	✗	✗	✓	✗
Eckey *et al.* [17]	✗	✗	✓	✓	✓	✓	✓	✗
Vein [22]	✗	✗	✗	✗	✗	✗	✗	✗
Auto tune [23]	✗	✗	✗	✗	✗	✗	✗	✗
Kadry *et al.* [24]	✗	✗	✗	✗	✗	✗	✗	✗
FSTR [29]	✗	✗	✗	✗	✗	✗	✗	✗
SilentWhispers [35]	✗	✓	✓	✓	✓	✓	✓	✗
Blanc [42]	✓	✓	✓	✓	✓	✓	✓	✗
SpeedyMurmurs [51]	✓	✓	✗	✓	✓	✓	✗	✗
Spider [53]	✗	✗	✗	✗	✗	✗	✗	✗
Flash [61]	✗	✗	✗	✗	✗	✗	✓	✗
Coinexpress [62]	✓	✗	✗	✓	✓	✓	✓	✗
Webflow [64]	✗	✓	✗	✓	✓	✓	✗	✗
Robustpay [65]	✗	✗	✓	✗	✗	✗	✓	✗
Robustpay+ [66]	✗	✗	✓	✗	✗	✗	✓	✗
RACED	✓	✓	✓	✓	✓	✓	✓	✓

Unlike Robustpay and Robustpay+, [14] constructs only a single path from the sender to the receiver and the entire amount is routed along this path. Auto-Tune [23] proposed by Hong *et al.* is a routing protocol that supports structured payments. Auto-Tune computes a total of k shortest paths between a sender and receiver where k, an arbitrary number, is decided by the sender. Unlike [14], the amount to be transacted is split across these multiple paths and is routed to the receiver. This work does not take into account the presence of redundant nodes along the k shortest paths, which makes it different from the ideas in [65, 66]. We refer the reader to the Table 1 for the differences between *RACED* and [14, 23, 65, 66].

In Table 1, we give a qualitative comparison between *RACED* and other routing protocols based on the comparison metrics defined as follows. 1) Concurrency: Concurrency is achieved when several transactions are routed simultaneously. 2) Privacy: Privacy is achieved when the identity of a node is not known to any other node in the network except its immediate neighbors. 3) Balance Security: Security is achieved when no honest party loses funds because of malicious behavior by other parties in the system. 4) Topology privacy: Topology privacy is achieved when no node in the network knows the entire network topology. 5) Avoids source routing: Source routing is avoided when the sender does not construct the entire path from itself to the receiver. 6) Decentralization: Decentralization is achieved when there is/are no central entity/entities that construct the path for the sender. 7) Atomicity: Atomicity is achieved when all the link weights of the nodes along the transaction path go back to the state that they were in before the transaction was initiated in the event of a transaction failure. 8) Disjoint graph applicability: A routing protocol is said to be applicable to disjoint graphs, if it works even when the network graph is not fully connected. For the routing protocols in Table 1, we conjecture that support for concurrency, privacy and atomicity can be provided (in the protocols that do not already have them) by using HTLCs [31] and the identity generation mechanism used in this paper. Modifying these protocols to achieve the remaining properties of topology privacy, avoiding source routing, decentralization, and making them applicable for disjoint graphs is non-trivial and is not a part of their design goals.

3 SYSTEM MODEL

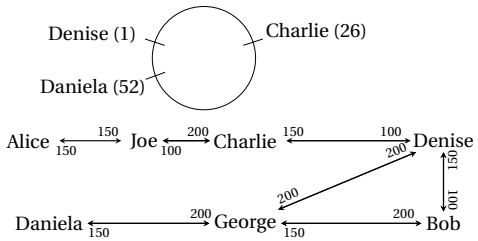

Figure 1: Three routing helpers in a DHT overlay over a PCN

In this section, we introduce the components of *RACED*, the parties involved, and the terminology we use in the rest of the paper.

A PCN can be modeled as a directed graph where a directed edge from a node i to j with an edge weight of α signifies the balance of node i in the payment channel between i and j, denoted by $lw_{i,j} = \alpha$. For instance, referencing Figure 1, in the link between Denise and Bob, $lw_{\text{Bob,Denise}} = 100$ and $lw_{\text{Denise,Bob}} = 150$.

3.1 Parties

Routing Helpers (RH**)**: In *RACED*, a routing helper (RH) is a node that helps the sender and the receiver route transactions between each other. We define a dynamic set $\mathbb{RH}$ that contains all the routing helpers. RHs in *RACED* are similar to the "routing nodes" or trampoline nodes used by the real-world PCN, Lightning Network [28].

If a node volunteers to become an RH, it needs to join a Distributed Hash Table (DHT) overlay and establish channels with a few other nodes. This is independent of the underlying PCN topology. We do not assume RHs are trusted, nor do we assume any honest majority among them. In *RACED*, the RHs charge a fees for providing their services and *RACED* is resilient to (n-2) RH failures for "n" RHs. We organize all RH $\in \mathbb{RH}$ as part of a DHT to ensure that they can route transactions in $O(\log |\mathbb{RH}|)$ time, using consistent hashing to locate each other. In this paper we instantiate the DHT using Chord [56], however, there are no technical impediments to using other DHT protocols such as Pastry [52], Kademlia [36], Tapestry [67] and more. In Figure 1, we depict three routing helpers, Charlie, Denise and Daniela. The numbers adjacent to the routing helpers represent their unique identifiers inside the DHT ring.

Sender and Receiver (sender**,** receiver**)**: With respect to Figure 1, the sender, Alice, is a node in the PCN who initiates a payment that needs to be routed across the network to receiver Bob. She only knows the link weights of her immediate neighbors. Once a path has been found between Alice and Bob using *RACED*, Bob generates parameters needed for establishing HTLCs (Hashed Time-Lock Contracts) [31] to complete the payment process. The purpose of establishing HTLC is to ensure atomicity of payments. We assume Alice and Bob can communicate with each other using a secure out-of-band communication channel, but can only do path-based routing of transactions. This is similar to real world PCNs, such as the Lightning Network [26], where out-of-band communication channels are used by the receiver to communicate the digest required to complete the HTLC payment to the sender.

End Routing Helper (*endRH***) and Nearest Routing Helper (***nearRH***)**: *endRH* is the routing helper from the DHT ring that is closest to Bob based on hop count. Similarly, *nearRH* is the nearest routing helper based on hop count to Alice. If we assume that the path taken is Alice → Joe → Charlie → Denise → Bob, the *endRH* is Denise and *nearRH* is Charlie.

Blockchain: *RACED* can be deployed on any permissioned or permissionless blockchain that supports HTLCs. *RACED* is compatible with the Lightning Network, which runs on top of the Bitcoin blockchain. In *RACED* we only use the blockchain for dispute resolution and it is not used during transaction routing and processing.

3.2 Setup and Terminology

Keys setup: In *RACED*, every user i in the PCN has a long-term signing and verification keypair denoted by (sk_i,vk_i), and a pseudonymous, temporary signing and verification keypair (SK_i,VK_i). In a decentralized network, each node generates its own keys. [1] A node's long-term public key in *RACED* is used within the network to establish an encrypted and authenticated connection with its neighbors. The temporary keys in *RACED* provide pseudonymity and hide the real identity of the node from its non-neighboring nodes in the PCN. To enable this, the temporary verification key is signed by the long-term signing key to produce a signature: $\mathsf{Sign}_{\mathsf{sk}_i}(\mathsf{VK}_i) \rightarrow \sigma$. Each user i exchanges its temporary and long-term verification key with all its neighbors, who verify σ using i's long-term verification key. Two nodes that are not immediate neighbors, use their temporary signing keys to sign messages and their temporary verification keys to verify the corresponding signatures. If Alice intends to route a payment to Bob, we assume both of them will know each other's real identities, since a sender will not typically route a payment to an unknown receiver.

Immediate Neighbor: Consider two nodes i and j that have a payment channel between them with the link weights denoted by $lw_{i,j}$ and $lw_{j,i}$. These two nodes are each other's immediate neighbors.

Pathfinding and routing times: We define the pathfinding time as the time taken to find a path involving several intermediate nodes between the sender and the receiver. Routing time is defined as the time taken to route the payment after a path has been found.

Routing fees: In PCNs, every node charges fees for forwarding the payment from its predecessor to its successor along the path; the fee structure varies according to the PCN being used. For instance, Lightning Network charges two types of fees, the base fee, which is fixed irrespective of the transaction amount, and rate fees that vary according to the amount being routed [27]. In this paper, we assume a unit fee is charged per hop, making the routing fees and the path length equal.

4 ADVERSARY MODEL

In this section, we outline the trust assumptions for the parties involved in *RACED*, and state our security and privacy goals. The sender and receiver in a transaction can be un-trusted and can arbitrarily deviate from protocol steps. Either of them can choose to abandon a transaction in-progress, or introduce delays in a transaction, with the goal of locking up collateral along paths. In *RACED*, we assume each sender and receiver have access to each other's real identities, and the receiver will know the amount, amt being transacted between them, since users do not send payments to unknown entities with unspecified amounts. All the nodes in the PCN, including the sender and the receiver, will know the real identities of all the routing helpers, RHs in the DHT ring, and will also know the maximum amount that each RH can route to its finger table entries. Every node in the PCN, including the routing helpers will know the balances they have and will also know the balance of their immediate neighbor in the payment channel between the node and its immediate neighbor. In addition to this, the nodes in the PCN present along the path for routing a transaction between a sender and a receiver will know the real identities of all their immediate neighbors and the amount being transacted between the sender and receiver along that path. The nodes in the PCN that are *not* along the path for a transaction between the sender and receiver will only know the real identities of their immediate neighbors and will not have access to any information regarding the transaction, such as the amount, transaction id, etc.

The RHs in *RACED* can also be malicious. They can arbitrarily deviate from the protocols, although we assume at least two RHs will be available at a given point of time to route transactions. For addressing distributed denial of service attacks where all the nodes in a DHT are taken down by an adversary, we refer the reader to existing mitigation strategies [6, 54, 59]. We also assume the adversary will be economically rational, i.e., it will always try to

[1] For instance, in transactions involving Bitcoin in the Lightning Network, each node generates a long-term keypair on Bitcoin's secp256k1 elliptic curve [32].

maximize its profit. The *nearRH* knows the pseudonymous identity of the sender and the *endRH* will know the pseudonymous identity of the receiver. Other routing helpers (which are neither *nearRH* or *endRH*) will know the amount being routed for a transaction if they are present in the finger table of the *nearRH* or if they are present in the finger tables of RHs which are present in the *nearRH*'s finger table. If a RH is present in both the finger table of *nearRH* and *endRH*, it will have access to the amount being transacted.We now give our security and privacy goals.

Defining our Security and Privacy Goals. 1) **Balance Security**: No honest node along a transaction path should lose funds even if all the other nodes, including the intermediaries, and/or the sender, receiver, are malicious. If the *nearRH* or *endRH* turn malicious at any point and decide to leak the identity of the sender or the receiver, respectively, it will only reveal their pseudonymous identities since the real identities of the sender and receiver are not known to any RHs in the DHT ring. 2) **Sender/receiver privacy**: The real identities of the sender and the receiver are only known to each other and their immediate neighbors in the network. 3) **Link privacy**: Every node only knows the balance in the channel it shares with its immediate neighbors. 4) **Atomicity**: If a transaction does not go through for any reason, all the link weights of the nodes along the transaction path should go back to the state that they were in before the transaction was initiated.

5 CONSTRUCTION

In this section, we present the challenges associated with leveraging DHTs for secure routing in PCNs and we describe the key ideas in *RACED* that solve these challenges and describe the detailed construction of *RACED*.

To address the challenge identified in Section 1, our idea is to use a DHT comprising of RHs which guarantees a logarithmic routing time. We note that it is non-trivial to apply DHTs to perform secure PCN routing due to the following challenges:

Challenge 1: DHTs were designed to facilitate information sharing in a p2p network, whereas PCNs were developed for facilitating financial transactions between users. Nodes in the DHT communicate with each other using the standard IPv4 communication protocol. In PCNs, though nodes communicate with each other using the same standard, they also need to exchange payments between them for which the IPv4 communication standard cannot be used. As a solution to this challenge, RHs in *RACED* open a payment channel on the blockchain with each of the RHs in their finger tables to facilitate payments.

Challenge 2: In DHTs, there is no notion of privacy; each peer in the DHT knows the details of the file (information) segments that every other peer is responsible for. Whereas in PCNs, the local channel balance of a node is known only to its immediate neighbor. In *RACED*, to safeguard its local channel balance in a payment channel, each RH i decides on a maximum amount that it can transact with its finger table entry k and only this maximum amount is known to all the other nodes in the PCN, providing link privacy to the RHs.

Challenge 3: In a DHT, the property of atomicity (defined in Section 4) is not required since nodes only exchange information. Whereas in PCNs atomicity is very important since it ensures that no honest party loses their funds because of malicious behavior by other parties in the system. In *RACED* atomicity is ensured since RHs (and all the other non-routing helper nodes) process payments between each other using HTLCs.

Challenge 4: Transacting money between nodes in the PCN causes a depletion in the balance of node, whereas, no such depletion exists in DHTs. We solve this challenge using the maximum amount computation described in Protocol 1.

Addressing these challenges and utilizing DHTs to reduce the overall routing complexity in PCNs requires careful design and is non-trivial. In what follows, we first give a brief overview that describes the working of *RACED* that solves all the aforementioned challenges at a high level and we follow it up with a detailed description of its construction. For the reader's easy reference, we give a table of notations in Table 2. In *RACED*, we instantiate the DHT using Chord [56]. Due to space constraints, we give an overview of Chord in Appendix B.

Table 2: Notations

Notation	Description
λ	Security parameter
amt	Amount to be paid by the sender to the receiver
$\mathbb{RH}$	Set of routing helpers
n	Number of nodes in the PCN
$\mathbb{I}_i$	Set of immediate neighbors of a node i in the PCN
$(\mathsf{SK}_i,\mathsf{VK}_i)$, $(\mathsf{sk}_i, \mathsf{vk}_i)$	Temporary and long-term signing/verification keypair of node i
endRH, *nearRH*	End routing helper and nearest routing helper, respectively
δ	Time interval for signature generation
$\mathsf{max}_{i,j}$	Maximum amount that can be transacted between nodes i and j
$\mathsf{hc}_{i,j}$	Number of hops between nodes i and j
txid	Transaction identifier
$tc_{i,j}$	Current timestamp for signature created on $\mathsf{max}_{i,j}$
$tv_{i,j}$	Time until which the signature created on $\mathsf{max}_{i,j}$ is valid

5.1 Technical Overview

Let us consider a sender Alice in a PCN, as depicted in Figure 2, who intends to route an amount, amt = 50 coins to a receiver, Bob. We use RHs, Charlie, Daniela, and Denise that belong to a set $\mathbb{RH}$, to route the payment. During the key-generation and setup phase (described in Protocol 6), each node creates a pair of long-term and temporary signing and verification keys. The temporary identity of a node is tied into the long-term identity as described in Section 3. This is done to hide the real identity of a node in the PCN from its non-neighboring nodes, which helps in achieving the goal of sender/receiver privacy as described in Section 4. In the DHT setup phase (described in Protocol 1), the first node that volunteers to be an RH establishes the DHT overlay by creating a unique identifier (depicted as the number next to an RH's name) and populating its local hash table. This local hash table is termed as a node's *finger table* in Chord [56]. Since we use Chord to instantiate the DHT in *RACED*, for terminological consistency, we refer to a node's local hash table as finger table in the rest of the paper (we give a detailed description of Chord in Appendix B). All the subsequent nodes that volunteer to be RHs join the DHT, create unique identifiers and populate their finger tables with the identifiers of other RHs. The RHs create signatures (using their long-term signing keys) on the maximum amount that they are willing to route to the other RHs in their finger tables. This is done to hide the actual channel capacities between the RHs in the DHT, which helps us in achieving our goal of link privacy.

Alice finds a path to the RH nearest to her, *nearRH* using any of the existing constructions such as [35, 42, 51]. These existing routing algorithms have an end-to-end worst-case pathfinding time complexity of $O(n)$, where n is the number of nodes in the PCN. Our goal in this paper is to improve this worst-case upper bound by using DHTs. Inside the DHT, the worst case pathfinding complexity is logarithmic in the number of nodes, $O(\log |\mathbb{RH}|)$ which improves the overall complexity of pathfinding. Hence, we focus on routing inside the DHT ring, and assume the sender/receiver can find a path to RHs using existing methods. In Figure 2 we assume the *nearRH* is Charlie. Alice requests Charlie to find paths from himself to all the other RHs in the DHT ring, i.e., Denise and Daniela. Charlie has to find paths such that the amount, amt to be sent by Alice is less than or equal to the maximum amount, $\mathsf{max}_{i,k}$ that each RH i on a given path is willing to route to the next RH k in the path. Charlie finds the paths in two phases. In Phase 1, Charlie finds paths from himself to all the RHs that are a part of his finger table and that can route the amount, amt specified by Alice and adds these RHs to a stack $\mathbb{P}$ maintained locally by him. In Phase 2, Charlie finds paths from himself to the RHs that are *not* part of his finger table but can still route the amount, amt specified by Alice. These two phases are described in detail in Protocol 3. Once the pathfinding phases are complete, the stack $\mathbb{P}$ that contains the list of paths and signatures is sent to Alice. Alice then verifies the signatures of the RHs in the stack $\mathbb{P}$ on the maximum amount that they can route to the RHs in their finger tables.

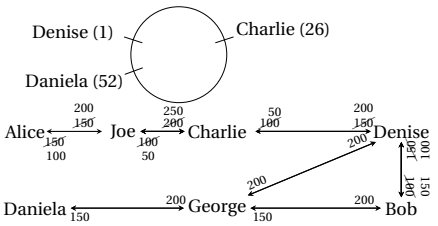

Figure 2: Alice transmitting 50 coins to Bob via RHs Charlie and Denise in the DHT ring.

Upon successfully verifying the signatures, Alice sends the *endRH* in each path to Bob via a secure out-of-band communication channel. Bob then picks the *endRH* that is nearest to him based on hop count. In Figure 2, the *endRH* is Denise. Bob notifies Alice about his choice of the *endRH* via a secure out-of-band communication channel. Alice then picks the path containing the RH picked by Bob as *endRH*. This finalizes the path along which amt needs to be routed inside the DHT ring. At any point, if the signatures do not verify, Alice writes the publicly verifiable signature details to the blockchain, at which point they can be verified by miners, and others involved in the system. This ensures that cheating RHs will be caught, and made to leave the DHT ring. Once Alice and Bob have agreed on the *endRH* Denise, Bob chooses a random preimage X, and hashes it to produce a digest Y. The payment mechanism is initiated using HTLCs. Using HTLCs ensures that no honest party loses any funds because of malicious behavior by other parties in the system, which helps us in achieving our goal of balance security. HTLCs also ensure that all the link weights of the nodes along the transaction path go back to the state they were in prior to the commencement of the transaction if the transaction fails for any reason. This achieves our goal of atomicity.

5.2 Helper functions

We now describe the helper functions used in *RACED*'s protocols.
1) $\underline{\mathsf{ChoosePath}}\ (\mathbb{P}, \mathit{endRH}) \rightarrow \mathbb{P}'$: This function picks a path for routing the payment. It takes two parameters, stack $\mathbb{P}$, and the *endRH* as inputs, and returns a path $\mathbb{P}'$ that contains the *endRH* as the last node in the path. If multiple paths with the same RH as the *endRH* are present, the last RH, it returns the shortest path.
2) $\underline{\mathsf{NH}\ (i, j) \rightarrow hc_{i,j}}$: This function calculates the hop count between two nodes, i and j inside the DHT ring.
3) $\underline{\mathsf{PC.Open}\ (\mathsf{VK}_i, \mathsf{VK}_j, lw_{i,j}, lw_{j,i}) \rightarrow \{success, failure\}}$: This function opens a new payment channel between two nodes. It takes in the temporary verification keys of the nodes involved in opening the channel, denoted by VK_i and VK_j, and the amounts being deposited on the links as input parameters. The nodes interested in opening a payment channel create a transaction tuple that contains the VK of the nodes and the amounts they individually intend to deposit in the channel. This tuple is signed by both nodes with their temporary signing keys, SK_i and SK_j, and is posted to the blockchain. The two nodes involved in the opening of the payment channel sign a single transaction tuple, making it a 2-2 multisig transaction. For representational clarity, we have abstracted the description of blockchain writes. Upon a successful opening of a payment channel

between the nodes, this function returns a *succcess*.

4) HTLC.Pay (vk_i, vk_j, *txid*, amt) $\rightarrow$ {*success*, *failure*}: This function completes the payment between two nodes once the preimage used to create the digest is revealed by one node to another. It takes the long-term verification key of the payer, vk_i, the long-term verification key of the payee, vk_j, the unique transaction id *txid*, and the amt being transacted as inputs. Once the payee has revealed the preimage using which the digest was produced, this function checks if the preimage being revealed is correct. If yes, the payer updates the link weights between them accordingly. Upon a successful release of the correct preimage by the payee and the updating of the link weights by the payer, it returns *success*. Else it returns a *failure*.

5) FT.Lookup (i, j) $\rightarrow$ l: This function performs the lookup operation for a node nearest to a destination node based on node identifier. It takes the node identifier of the source denoted by i and the node identifier of the destination denoted by j as the inputs, and returns the node l who has the largest node identifier and is less than or equal to j from the finger table of i.

6) FT.Retrieve (i) $\rightarrow$ $\mathcal{B}_i$: This function retrieves the entries of a node's finger table. This function takes node identifier i of the node in the DHT ring as an input and returns a stack containing the node identifiers of the entries present in i's finger table.

7) RetrieveNext ($\mathcal{B}$) $\rightarrow$ i: This function takes a list as an input argument and returns the node identifier of the element that the head of the list points to.

8) RetrieveNeighbors (vk_i) $\rightarrow$ $\mathbb{I}_i$: This function is used to retrieve the immediate neighbors of a node. It takes the long-term verification key of a node i, vk_i, as an input and outputs a list, $\mathbb{I}_i$ containing the verification keys of the immediate neighbors of the node i.

9) Succ.Lookup (i) $\rightarrow$ j: This function looks up the successor of a node in the DHT ring. It takes the node identifier of a node denoted by i, and returns its immediate successor which is defined as the smallest node identifier in the DHT ring that is larger than i.

10) FT.Search (i,j) $\rightarrow$ {*success*, *failure*}: This function searches for the presence of a node in another node's finger table. It takes in the node identifier of the node calling this function i and the node identifier of the node being searched j as inputs, and returns *success* if j is present in the finger table of i.

6 PROTOCOLS

RACED consists of seven protocols: Key Setup (Protocol 6), DHT Setup (Protocol 1), DHT Processing (Protocol 2), Find Path (Protocol 3), Path Validation (Protocol 4), Node Joining And Node Leaving (Protocol 5), Routing Payment (Protocol 7).

The protocol Key Setup handles the generation of long-term and pseudonymous identities for all the nodes in the PCN. These keys are used to sign and verify messages in the subsequent protocols of *RACED*. The steps of this protocol are self-explanatory. Due to space constraints, we give the protocol and its full description in Appendix C.

DHT Setup, Protocol 1: This protocol handles the DHT ring setup and the computation of signatures on the maximum amount each RH can route to the RHs in its finger table. The DHT ring setup facilitates the joining of nodes as RHs and the signatures created on the maximum amount from this protocol are used by the sender while selecting a suitable path to route the amt to the receiver. Each node i in the PCN that volunteers to be an RH hashes its IP address, $ipaddress_i$ with a collision-resistant and consistent hash function. The resulting digest of the hash becomes the node identifier of the RH. Each RH i locally maintains two stacks $\mathcal{B}_i$ and $\mathcal{J}_i$, and a list $\mathcal{L}_i$. RH i will then compute the list of RHs that are a part of its finger table and adds them to the stack $\mathcal{B}_i$, which contains repeated entries. The unique entries from $\mathcal{B}_i$ are added to the stack $\mathcal{J}_i$. RH i creates signatures on the max amount that it can route to the RHs in the stack $\mathcal{J}_i$. The computation of the maximum amount and the corresponding signatures is done to hide the actual channel capacities between the nodes in the DHT, which helps us in achieving our goal of *link privacy*. These signatures are created by i and the RHs in its finger table using their long-term signing keys and are added to the list $\mathcal{L}_i$. In addition, two timestamps, tc, which is the timestamp at which the signature was created, and tv, which is the timestamp until which the signature is valid, are added to $\mathcal{L}_i$. This protocol can be run in parallel by all the RHs in the DHT ring.

DHT Processing, Protocol 2: This protocol handles the creation of signatures on the new value of maximum amount, $\mathsf{max}'_{i,k}$, that each RH i can route to each RH k in its finger table once the time epoch δ expires, or when the maximum amount, $\mathsf{max}_{i,k}$ that can be routed from a RH i to its finger table entry k exceeds the link weight (balance) that an RH has in the payment channel with its finger table entries. The signatures created in this protocol will be used by the sender to check that the liquidity that exists between RHs is sufficient to route the amt specified by her. The value of δ is a system parameter that the RHs decide during the DHT Setup phase. Once the time epoch expires or when the maximum amount, $\mathsf{max}_{i,k}$ that can be routed between an RH i and its finger table entry k exceeds the balance, $lw_{i,k}$ that the RH i has in the payment channel between i and k, each RH i (that satisfies either of the two conditions in lines 3 or 16,) in the DHT ring retrieves the RHs from its finger table, removes the duplicate entries, and adds the unique

Protocol 1: DHT Setup

1 All RHs decide on the value of $\delta \leftarrow\$ \mathbb{R}^+$ and a hash function H: $\{0,1\}^\lambda \rightarrow \{0,1\}^m$
2 **for** $i = 1; i \leq |\mathbb{RH}|; i{+}{+}$ **do**
3 node i that joins the DHT ring hashes its $ipaddress_i$ and creates a digest Y_i: $H(ipaddress_i) \rightarrow Y_i$
4 node identifier of $i = Y_i$
5 node i will broadcast Y_i and vk_i to all the nodes in the PCN.
6 node i maintains $\mathcal{B}_i = \emptyset$, $\mathcal{J}_i = \emptyset$ and $\mathcal{L}_i = \emptyset$
7 $\forall\, j \in [1..m]$ node i does $\mathcal{B}_i$.Add($i + 2^{(j-1)} \mod m$)
8 node i does RemoveDuplicates($\mathcal{B}_i$) $\rightarrow$ $\mathcal{J}_i$
9 **while** ($\mathcal{J}_i$.empty = *False*) **do**
10 Pop.($\mathcal{J}_i$) $\rightarrow$ k
11 node k does $\mathsf{Sign}_{\mathsf{sk}_k}(\mathsf{max}_{i,k}) \rightarrow \sigma^k_{\mathsf{max}_{i,k}}$
12 node i does $\mathsf{Sign}_{\mathsf{sk}_i}(\mathsf{max}_{i,k}) \rightarrow \sigma^i_{\mathsf{max}_{i,k}}$ and
13 node i does $\mathcal{L}_i$.Add($\mathsf{max}_{i,k}$, $\sigma^i_{\mathsf{max}_{i,k}}$, $\sigma^k_{\mathsf{max}_{i,k}}$, $tc_{i,k}$, $tv_{i,k}$)

Protocol 2: DHT Processing

/* All members of RH check this condition */
for $i=1; i \leq |\mathbb{RH}|; i$++ **do**
  node i does FT.Retrieve(i)$\rightarrow \mathcal{B}_i$ and RemoveDuplicates($\mathcal{B}_i$) $\rightarrow \mathcal{J}_i$
  while $\mathcal{J}_i$.empty = *False* **do**
    node i retrieves ($\max_{i,k}$, ·, ·, ·, ·) from $\mathcal{L}_i$
    if $\max_{i,k} > lw_{i,k}$ **then**
      node i assigns the maximum amount that can be routed to node k to $\max'_{i,k}$ and does $\mathsf{Sign}_{\mathsf{sk}_i}(\max'_{i,k}) \rightarrow \sigma^{i}_{\max'_{i,k}}$
      node k does $\mathsf{Sign}_{\mathsf{sk}_k}(\max'_{i,k}) \rightarrow \sigma^{k}_{\max'_{i,k}}$ and node i does $\mathcal{L}_i$.Delete($\max_{i,k}$, $\sigma^{i}_{\max_{i,j}}$, $\sigma^{k}_{\max_{i,k}}$, $tc_{i,k}$, $tv_{i,k}$) and $\mathcal{L}_i$.Add($\max'_{i,k}$, $\sigma^{i}_{\max'_{i,k}}$, $\sigma^{k}_{\max'_{i,k}}$, $tc'_{i,k}$, $tv'_{i,k}$)
if ($curr_{time}$ mod δ = 0) **then**
  for $i=1; i \leq |\mathbb{RH}|; i$++ **do**
    node i does FT.Retrieve(i)$\rightarrow \mathcal{B}_i$ and RemoveDuplicates($\mathcal{B}_i$) $\rightarrow \mathcal{J}_i$
    while ($\mathcal{J}_i$.empty = *False*) **do**
      node i does Pop.$\mathcal{J}_i \rightarrow k$
      node i retrieves ($\max_{i,k}$, ·, ·, ·, ·) from $\mathcal{L}_i$
      node i assigns the maximum amount that can be routed to node k for the current time epoch to $\max'_{i,k}$
      if ($\max_{i,k}$ == $\max'_{i,k}$) **then**
        $tv'_{i,k} = tv_{i,k} + \delta$
        Update ($\max_{i,k}$, ·, ·, ·, $tv_{i,k}$) with ($\max_{i,k}$, ·, ·, ·, $tv'_{i,k}$) in $\mathcal{L}_i$
      else
        node i does $\mathsf{Sign}_{\mathsf{sk}_i}(\max'_{i,k}) \rightarrow \sigma^{i}_{\max'_{i,k}}$
        node k does $\mathsf{Sign}_{\mathsf{sk}_k}(\max'_{i,k}) \rightarrow \sigma^{k}_{\max'_{i,k}}$
        node i does $\mathcal{L}_i$.Delete($\max_{i,k}$, $\sigma^{i}_{\max_{i,k}}$, $\sigma^{k}_{\max_{i,k}}$, $tc_{i,k}$, $tv_{i,k}$) and $\mathcal{L}_i$.Add($\max'_{i,k}$, $\sigma^{i}_{\max'_{i,k}}$, $\sigma^{k}_{\max'_{i,k}}$, $tc'_{i,k}$, $tv'_{i,k}$)
        return $\mathcal{L}_i$
else
  do nothing

ones to the stack $\mathcal{J}_i$. Each RH i will compute the signatures on $\max'_{i,k}$ for each RH present in its finger table. These signatures are computed by the RHs using their long-term signing keys. After the computation of signatures, the signatures attesting to the max in $\mathcal{L}_i$ from the previous time epoch are replaced with the new ones. In addition, the previous time stamps tc and tv are replaced with the fresh ones in the list $\mathcal{L}_k$. If for any reason, the $\max_{i,k}$ between a RH i and its finger table entry k has not changed from the previous time epoch, only the time stamp of the signature validity, tv will be incremented by δ and updated in the list $\mathcal{L}_i$. This protocol can be run by each RH in parallel.

Find Path, Protocol 3: This protocol finds paths from the *nearRH* to all the other RHs in the DHT ring that can route the amt specified by the sender. Initially, the sender, Alice, creates a random transaction id, *txid*, contacts the *nearRH* and sends the *txid*, the amt that she intends to route, her signature, σ_{amt}, and her temporary verification key $\mathsf{VK}_{\mathsf{Alice}}$ through a path-based transaction to *nearRH*. The $\mathsf{vk}_{\mathsf{receiver}}$ (in this case Bob), *txid*, amt are added to a list $\mathbb{K}$ by Alice. Alice can be involved in several transactions with several receivers. The list $\mathbb{K}$ helps Alice maintain a record of all the transactions in which she is the sender. The path from the sender to the *nearRH* can be calculated using the constructions described in [35, 42, 51]; we do not describe this process in this paper. The *nearRH* locally maintains two stacks $\mathbb{P}$ and $\mathbb{W}$. Upon successful verification of the signature of Alice on the amt, the *txid*, amt is sent by *nearRH* to all the RHs in its finger table. For each entry, in its finger table, the *nearRH* checks if amt is less than the max that the *nearRH* can route to that RH in $\mathcal{B}_{nearRH}$ (stack containing *nearRH*'s finger table entries). For each such entry, the *nearRH* randomly samples a path identifier, denoted by pathid and adds the pathid, the node identifier of *nearRH*, node identifier of the RHs in its finger table, the corresponding max, the signatures of the *nearRH* and the RH on the max, the time stamps of signature creation and signature validity and the tuple is pushed on to the stack $\mathbb{P}$. In this manner, *nearRH* finds paths from to all RHs in its finger table. *nearRH* now finds paths from itself to the RHs in the stack $\mathbb{W}$. The *nearRH* pops the first RH in $\mathbb{W}$, denoted by p, and selects the RH closest to p based on its node identifier and assigns it to i. The *nearRH* checks if the liquidity between *nearRH* and i is suitable to route the amt specified by Alice and sends a tuple consisting of the *txid*, FindpathReq message and the node identifier of p to i. This transfers the control flow from *nearRH* to i. i then checks for the presence of p in its finger table. If present, i constructs a tuple $\mathbb{Q}_i$ = (i, p, $\max_{i,p}$, $\sigma^{i}_{\max_{i,p}}$, $\sigma^{p}_{\max_{i,p}}$, $tc_{i,p}$, $tv_{i,p}$) and sends this tuple to *nearRH* along with FindpathResp response message. If the RH p is not present in the finger table of i, i looks up the closest RH to p in its finger table based on node identifier. This RH is denoted by k. i checks if the liquidity between i and k is sufficient to route the amt specified by Alice, if yes, k is sent to *nearRH*. The *nearRH* then assigns the node identifier of k to i. This process is repeated until a suitable path to p is found. If a path to p is found, the *nearRH* samples a path identifier, pathid, at random. The *txid*, pathid and all the tuples $\mathbb{Q}_i$, for each RH i generated until this point are pushed on to the stack $\mathbb{P}$. Phase 2 is repeated until the stack $\mathbb{W}$ becomes empty, upon which *nearRH* sends $\mathbb{P}$ to Alice. The *nearRH* can turn malicious at any point in any of these phases and try to manipulate the contents of any tuple returned by the RHs. However, the malicious behavior of *nearRH* will be caught when Alice verifies the amounts and signatures in Protocol 4.

Path Validation, Protocol 4: Path Validation is the fifth protocol in *RACED*. Alice calls this protocol once she receives the stack $\mathbb{P}$ containing tuples of paths from the *nearRH* to all the RHs in the DHT ring. In this protocol, Alice verifies the signatures of the RHs contained in $\mathbb{P}$ on the max that can be routed between each pair of RHs that are immediate neighbors in the DHT ring. Initially, Alice pops

Protocol 3: Find Path

Alice does $k \leftarrow\$ \{0,1\}^{\lambda}$, $H(k) \rightarrow txid$ and sends ($txid$, amt, σ_{amt}, VK_{Alice}) to $nearRH$, initializes a list $\mathbb{K} = \emptyset$, creates a tuple ($\text{vk}_{\text{receiver}}$, $txid$, amt) and adds it to $\mathbb{K}$
$nearRH$ initializes stacks $\mathbb{P}$, $\mathbb{W} = \emptyset$ and does **if** $(\text{Verify}_{\text{VK}_{Alice}}(\text{amt}, \sigma_{\text{amt}}) \rightarrow 0)$ **then**
  return $\perp$
else
  $\text{FT.Retrieve}(nearRH) \rightarrow \mathcal{B}_{nearRH}$
  $\forall$ RH i $\in \mathcal{B}_{nearRH}$, $nearRH$ sends ($txid$, amt, σ_{amt}, VK_{Alice}) to each node i and $\forall\ j \in \mathbb{RH}$ and $j \notin \mathcal{B}_{nearRH}$ does $\mathbb{W}$.Push(j)
  while $(\mathcal{B}_{nearRH}.\text{empty} = False)$ **do**
    each node $i \in \mathcal{B}_{nearRH}$ does **if** $(\text{Verify}_{\text{VK}_{Alice}}(\text{amt}, \sigma_{\text{amt}}) \rightarrow 0)$ **then**
      return $\perp$
    else
      $\text{Pop.}(\mathcal{B}_{nearRH}) \rightarrow$ i
      if $(\text{amt} \leq \text{max}_{nearRH,\text{i}})$ **then**
        $nearRH$ picks pathid $\leftarrow\$\ \{0,1\}^{\lambda}$ and does $\mathbb{P}$.Push($txid$, pathid, $nearRH$, i, $\text{max}_{nearRH,\text{i}}$, $\sigma^{\text{i}}_{\text{max}_{nearRH,\text{i}}}$, $\sigma^{nearRH}_{\text{max}_{nearRH,\text{i}}}$, $tc_{nearRH,\text{i}}$, $tv_{nearRH,\text{i}}$, VK_{Alice})
  while $(\mathbb{W}.\text{empty} = False)$ **do**
    $nearRH$ checks **if** (FindPathResp, k, $txid$) *has been received* **then**
      $nearRH$ does i = k, sends ($txid$, FindPathReq, p) to i
    else
      $nearRH$ does $\text{Pop.}(\mathbb{W}) \rightarrow$ p, $\text{FT.Lookup}(nearRH, \text{p}) \rightarrow$ i
    $nearRH$ checks **if** $(\text{amt} \leq \text{max}_{nearRH,\text{i}})$ **then**
      $nearRH$ sends ($txid$, FindpathReq, p) to i
      `/* Node i runs steps 21-25 */`
    if $(\text{NH}(\text{i,p} == 1))$ **then**
      if $(\text{amt} \leq \text{max}_{\text{i,p}})$ **then**
        i retrieves ($\text{max}_{\text{i,p}}$, $\sigma^{\text{i}}_{\text{max}_{\text{i,p}}}$, $\sigma^{\text{p}}_{\text{max}_{\text{i,p}}}$, $tc_{\text{i,p}}$ and $tv_{\text{i,p}}$) from $\mathcal{L}_i$ and constructs a tuple $\mathbb{Q}_{\text{i}}$ = (i, p, $\text{max}_{\text{i,p}}$, $\sigma_{\text{max}^{\text{i}}_{\text{i,p}}}$, $\sigma_{\text{max}^{\text{p}}_{\text{i,p}}}$, $tc_{\text{i,p}}$, $tv_{\text{i,p}}$) and sends the tuple ($txid$, FindpathResp, $\mathbb{Q}_{\text{i}}$) to $nearRH$. $nearRH$ samples pathid $\leftarrow\$\ \{0,1\}^{\lambda}$ and does $\mathbb{P}$.Push($txid$, pathid, $\mathbb{Q}_{\text{i}}$)
      else
        i construct a tuple ($txid$, FindpathResp, i, p, $\perp$) sends it to $nearRH$
    else
      i does $\text{FT.Lookup}(\mathcal{B}_{\text{i}}) \rightarrow$ k, sends ($txid$, amt, VK_{Alice}) to k **if** $(\text{amt} \leq \text{max}_{\text{i,k}})$ **then**
        i constructs a tuple $\mathbb{Q}_{\text{i}}$ = (i, k, $\text{max}_{\text{i,k}}$, $\sigma^{\text{i}}_{\text{max}_{\text{i,k}}}$, $\sigma^{\text{k}}_{\text{max}_{\text{i,k}}}$, $tc_{\text{i,k}}$, $tv_{\text{i,k}}$,VK_{Alice}) and sends ($txid$, Findpathresp, k, $\mathbb{Q}_{\text{i}}$) to $nearRH$
$nearRH$ sends $\mathbb{P}$ to Alice

Protocol 4: Path Validation

Alice maintains a list $\mathbb{W}_{Alice} = \emptyset$ and receives $\mathbb{P}$ from the $nearRH$
while $(\mathbb{P}.\text{empty} = False)$ **do**
  Alice does $\text{Pop.}(\mathbb{P}) \rightarrow txid$
  Alice does **if** $\text{Pop.}(\mathbb{P}) \rightarrow$ pathid **then**
    Alice records a message (New Path) and does $\mathbb{W}$.Add(pathid)
  else
    Alice maintains a list $\mathbb{K}_{Alice} = \emptyset$
    Alice does $\text{Pop.}(\mathbb{P}) \rightarrow \mathbb{Q}_j$ = (j, $j+1$, $\text{max}_{j,j+1}$, $\sigma^{j}_{\text{max}_{j,j+1}}$, $\sigma^{j+1}_{\text{max}_{j,j+1}}$, $tc_{j,j+1}$, $tv_{j,j+1}$,VK_{Alice})
    if $(\text{amt} \leq \text{max}_{j,j+1})$ **then**
      if $(\text{curr}_{\text{time}} < tv_{j,j+1})$ **then**
        if $(\text{Verify}_{vk_j}(\text{max}_{j,j+1}, \sigma^{j}_{\text{max}_{j,j+1}}) \rightarrow 1)$ **then**
          if $\text{Verify}_{vk_{j+1}}(\text{max}_{j,j+1}, \sigma^{j+1}_{\text{max}_{j,j+1}}) \rightarrow 1$ **then**
            add $j, j+1$ to $\mathbb{K}_{Alice}$
          else
            BC.Write(j, $j+1$, $\text{max}_{j,j+1}$, $\sigma_{\text{max}^{j+1}_{j,j+1}}$, VK_{Alice})
        else
          BC.Write($j+1$, j, $\text{max}_{j,j+1}$, $\sigma_{\text{max}^{j}_{j,j+1}}$, VK_{Alice})
      else
        do nothing
    else
      $\text{BC.Write}(j, j+1, \text{amt}, \text{max}_{j,j+1}, \text{VK}_{\text{Alice}})$
  for $i=1;\ i \leq |\mathbb{W}|_{Alice}\ ;\ i{+}{+}$ **do**
    if $\mathbb{W}_i = \mathbb{W}_{i+1}$ **then**
    $\text{BC.Write}(\text{vk}_{nearRH}, \text{vk}_{Alice}, \mathbb{W}_i, \mathbb{W}_{i+1})$

the $txid$ and the pathid from the stack $\mathbb{P}$. This stack now contains the tuples $\mathbb{Q}_i$, where $i \in [1..(|\mathbb{RH}| - 1)]$. Alice adds the pathid in each tuple to a list $\mathbb{W}$ that she locally maintains. Alice retrieves each tuple and initially verifies if the amount that she intends to route is less than the maximum amount that can be routed between the RHs in the tuple. If this verification fails, Alice writes the max, the long-term verification key of the routing helpers involved, and the amt she intends to route to the blockchain. Upon successful verification, Alice checks if the time stamp of signature validity, tv, is less than that of the current system time, $\text{curr}_{\text{time}}$. Upon successful verification, the signatures on the maximum amount created by the RHs are verified. If the signature verification does not pass, Alice posts node identifiers of the malicious routing helpers involved, the amt she intends to route, the max that can be transacted between the RHs and the signature of the malicious RH to the blockchain. The nature of punitive actions taken against malicious parties in *RACED* may vary across PCNs, and across implementations of *RACED*, e.g., banning malicious parties temporarily or permanently, reporting them to law enforcement, etc. Describing them is beyond the scope of this paper.

Protocol 5: Node Leaving and Node Joining the DHT

```
   /* Node leaving                                                    */
1  Let L be the leaving node and L does FT.Retrieve(L) → B_L
2  if (B_L.empty = False) then
3  |  L does FT.Delete(B_L)
4  else
   |  /* The lines 7-20 only run at the expiration
   |     of epoch δ                                                  */
5  |  for i=1;i≤ |RH|;i++ do
6  |  |  if (L ∈ B_i) then
7  |  |  |  each node i does Succ.Lookup(L) → S
8  |  |  |  if (FT.Search(S, i)→ success) then
9  |  |  |  |  do nothing
10 |  |  |  else
11 |  |  |  if (PC.Open(vk_S, vk_i,
12 |  |  |  lw_{vk_S,vk_i}, lw_{vk_i,vk_S}) → success) then
13 |  |  |  channel is established
14 |  |  else
15 |  |  |  do nothing
   /* New node joining                                                */
16 Let the joining node be J and J does H(ipaddress_J) → J,
     FT.Compute(J) → B_J and J does RemoveDuplicates(B_J) →
     J_J and J does
17 while (J_J.empty = False) do
18 |  Pop(J_J) → i
19 |  if (PC.Open(vk_J, vk_i, lw_{vk_J,vk_i}, lw_{vk_i,vk_J}) → success)
   |   then
20 |  |  channel is established
```

Node Leaving and Node Joining the DHT, Protocol 5: This protocol handles the joining of a node in the PCN as a RH and also handles the leaving of an existing RH from the DHT ring. First, we give a description of the process of an existing RH leaving the DHT ring and follow it up with a description of a node in the PCN joining as RH. We denote the RH leaving the DHT ring by L. Initially, L calls the PC.Close function to close all the payment channels with the RHs in its finger table. L maintains local storage that stores the long-term verification keys of its neighbors. These keys are retrieved from this storage during the closing of payment channels; for brevity, we have abstracted these details in the protocol. Once all the payment channels have been closed, the RHs in whose finger table L was a member finds L's successor, S. S performs a search operation to find the RH in whose finger table L was a member, but S is not a member. S then establishes payment channels with all such RHs. Upon successful establishment of the payment channels, the process of an existing RH leaving the DHT ring is completed. The second part of this protocol handles the joining of a new node as a RH in the DHT ring. The node that joins the DHT ring is denoted by J. It initially finds its successor based on its node identifier in the DHT ring, denoted by $\mathsf{curr}_{\mathsf{succ}}$. Using the node identifier of $\mathsf{curr}_{\mathsf{succ}}$, J computes the RHs in its finger table and adds them to the stack $\mathcal{J}_\mathsf{J}$. J opens payment channels with all the RHs in $\mathcal{J}_\mathsf{J}$. This completes the process of a new node joining as a RH in the DHT.

The protocol Routing Payment, Protocol 7 handles the routing of payment between Alice and Bob using HTLCs [31]. The steps for this protocol are self explanatory. Due to space constraints, we give the protocol and its full description in Appendix C.

7 EXPERIMENTAL EVALUATION

In this section, we explain our dataset collection, experimental setup and the results of our evaluation.

7.1 Dataset and Simulation Setup

In *RACED*, we use the transaction data from Ripple for our experimental evaluation. Transaction data about the most popular PCN, the Lightning Network, in particular, the data about the number and the amount of transactions is not publicly available. Due to this, and the fact that Lightning and Ripple are the only PCNs in use currently, we use transaction data from Ripple for our experiments. *RACED*, however, can be deployed on Lightning Network without any modification to the underlying structure of Lightning Network. The only overhead that *RACED* causes when deployed on Lightning Network and Ripple is the opening of payment channels in Lightning Network (called trustlines in Ripple [49]) by the RHs with the entries in their finger tables and the creation and verification of pseudonymous identities for every node in the PCN.

Table 3: Number of cryptographic operations performed/TX. Legend: $\mathbb{LM}$: landmark, d: size of the hash digest used in the DHT, T: number of cryptographic operations performed outside the DHT.

Operations	Protocols		
	RACED	Blanc	SW
Signing	$1+T$	13	$8\lvert\mathbb{LM}\rvert+1$
Verification	$\lvert\mathbb{RH}\rvert+T$	12	$7+\lvert\mathbb{LM}\rvert$
Hash	$3+T$	7	0
Encryption	T	7	0
Decryption	T	6	0
FT.Lookup	$O(\log d)$	0	0
FT.Compute	$\log d$	0	0

For the experiments, we collected transaction data from the Ripple network from 01-01-2021 to 12-31-2021 [44]. We chose to collect Ripple data due to the fact that Ripple's XRP token, has the sixth largest capitalization for a cryptocurrency and Ripple's market cap is the largest among all payment channel networks [46]. We used the Ripple API [45] to collect all the "Payment" transactions that were recorded on the Ripple ledger during the aforementioned time period. We only consider the "Payment" transactions since they are path-based transactions that involve several intermediate nodes between the sender and receiver. These transactions were recorded in several different currencies. Direct transactions between a sender and receiver pair which do not involve intermediaries were excluded from our collected data. We collected a total of 15,634,656 path-based transactions. We pre-processed the collected data to remove two types of anomalies that we have noticed: invalid currencies, and incomplete hash digests of transactions. Once the transaction data was collected and pruned, we created a directed

graph using the Ripple APIs [49] for every month from January 2021 to December 2021. We removed the edges with negative and zero link weights and converted all the link weights to USD. This gave us a graph of 225,264 nodes and 1,717,347 edges which was used in our experiments. All our experiments were run on a single machine equipped with AMD ™ EPYC processor (64 bit architecture) with 16 cores and 512 GB of RAM and a clock speed of 3.2 GHz. The code for all the routing algorithms was written in Python 3.8 and the NetworkX library [41] was used for simulations.

7.2 Evaluation And Results

We implement and experimentally compare *RACED* with several other comparable routing algorithms, specifically with SilentWhispers [35], SpeedyMurmurs [51] and Blanc [42] and show our results in Table 4. For all the experimental settings, we set the RHs for *RACED* and Blanc [42], and the number of landmarks for SilentWhispers [35] and SpeedyMurmurs [51] to eight. These routing helpers/landmarks were picked as the nodes with the highest out-degree in the graph. In our experiments, we constructed the graph for each month in 2021 and routed the transactions accordingly. By doing so, we capture the growth of the Ripple network over the year through our graph. Our experiments thus simulate the network's evolving, dynamic nature. The graphs for each month constructed from the Ripple data were disjoint. Hence we extracted the largest strongly connected component for each month's graph and routed the transactions (involving the USD currency) by selecting the sender-receiver pairs and routing helpers from that component. This is the "*RACED*-1-SCC" setting in the Table 4. In order to demonstrate the effectiveness of having the routing helpers connected via a DHT such as Chord (which is the central idea of *RACED*), we extracted the top eight (by node count) strongly connected components for each month. We selected one routing helper from each strongly connected component (based on the highest out-degree), connected them via a Chord ring, and all the transactions that were recorded for the USD currency in our dataset were routed through these RHs. We randomly sampled the sender and receiver from the dataset to ensure that no sender and receiver pair is from the same SCC. This is the "*RACED*-k-SCC" setting in Table 4.

We measured a total of four metrics for each month: 1) the transaction success ratio, which is the ratio of the number of transactions successful to the total number of transactions routed, 2) the average path length found, which is the total number of hops between the sender and receiver, 3) pathfinding time, which is the time taken to find a path between the sender and the receiver, and 4) routing time, which is the time taken to route the payment (nodes adjusting link weights) along the path from the sender to the receiver. We computed the mean of each metric across all twelve months along with the standard deviation. A total of 52,943 transactions which is the number of transactions that took place with the currency as USD during 2021-2022 on the Ripple ledger were routed concurrently.

Message passing between nodes in the course of a routing protocol in a p2p network such as PCN is an implementation-specific scenario that depends on the network in which the routing protocols are deployed. For instance, the Lightning network uses the in-built IPv4 or IPv6 connection that exists between the nodes in the PCN for message passing. For more information on this, we refer the reader to [33]. Ripple uses the Ripple Protocol Consensus Algorithm (RPCA) [48], which internally handles message passing. Similar to these, in our simulations for *RACED*, we use the NetworkX library, which handles message passing internally.

We used Dijkstra's shortest path algorithm to simulate the pathfinding outside the DHT in *RACED*. In *RACED*, the number of edges $|E|$ is significantly lower than $|V|^2/\log|V|$ for the PCN graph $G(V, E)$, hence we implemented the priority queue for Dijkstra's algorithm using a binary heap [16]. For *RACED*, the shortest path (in terms of the hop count) was chosen to route the payment from sender to *nearRH*, *nearRH* to other RHs in the DHT ring, and *endRH* to receiver. If multiple paths with the same hop count were present, the paths with the highest liquidity were chosen. If the hop-count and the liquidity between paths were the same, then a path was randomly chosen.

For the simulation of SilentWhispers [35], the number of landmarks was chosen as eight, and the landmarks were chosen as the nodes with the highest out-degree. Unlike *RACED*, SilentWhispers cannot be applied to disjoint graphs, since it uses BFS (breadth first search) to find a path between the sender and receiver. Even though BFS is asymptotically more efficient than Dijkstra's algorithm, the routing time and pathfinding times are significantly higher than *RACED*, since the overhead contributed by the number of cryptographic operations (signing and verification) is very high. Besides, unlike *RACED*, [35] offers no support for concurrent transactions. In *RACED* the path length inside the DHT ring is always 3 hops since we have chosen a total of 8 routing helpers.

SpeedyMurmurs [51] uses an embedding-based routing mechanism called VOUTE [50]. VOUTE uses a BFS-based approach to compute the embedding of all the nodes in the network with respect to their distances from the landmark. Apart from this, the BFS needs to be run by all the landmarks when a new node joins or an existing node leaves the network, which gives this protocol a high stabilization overhead. Hence the pathfinding time for this is higher than *RACED*. However, since there are no cryptographic operations involved, the pathfinding time is lower than that of SilentWhispers. The routing time for this protocol is also less than that of SilentWhispers and Blanc since the actual payment routing does not involve any cryptographic operations. This protocol also offers no privacy guarantees unlike *RACED*, where we offer sender, receiver, and transaction privacy.

The routing protocol Blanc [42], was simulated with the number of RHs chosen as eight similar to all the other protocols. It needs two RHs between the sender and the receiver, one picked by the sender and the other picked by the receiver. The routing/pathfinding time is very high in this protocol in comparison to the other protocols, since the pathfinding phase uses broadcasting in three segments: sender to RH1, RH1 to RH2, RH2 to receiver. Blanc does not address the issue of multiple SCCs. The routing time is higher than *RACED*, SilentWhispers, and SpeedyMurmurs, because Blanc involves the creation of pair-wise contracts (after the pathfinding phase) between nodes involved along the path attesting to the amount that will be transacted. Table 3 represents the number of cryptographic operations performed by each routing protocol per every transaction being routed. The transaction processing time for

Table 4: Performance of different pathfinding and routing protocols. Legend: PL: path length, 1-SCC: one large strongly connected component in the PCN graph, k-SCC: k strongly connected components. Metrics: success ratio (higher is better), mean path length (lower is better), pathfinding time (lower is better), routing time (lower is better).

Protocols	Success ratio	Mean PL (hop-count)	Pathfinding time (sec)	Routing time (millisec)
RACED 1-SCC (single graph)	98.85 ± 0.027	6.64 ± 0.287	31.24169 ± 56.144	3.126 ± 0.0254
RACED k-SCC (several disjoint graphs)	98.73 ± 0.148	7.31 ± 0.295	31.24289 ± 56.147	3.165 ± 0.03
SM 1-SCC (single graph) [51]	98.23 ± 1.64	4.21 ± 0.245	12460.0688 ± 11089.897	3500 ± 0.042
SW 1-SCC (single graph) [35]	94.63 ± 7.08	7.87 ± 0.387	51707.89 ± 11651.260	225000 ± 630
Blanc 1-SCC (single graph) [42]	97.92 ± 0.029	10.983 ± 0.754	56344.229 ± 186149.669	391164 ± 725.458

fiat currencies varies from a couple of hours to a couple of days depending on the geographical location of the sender and the receiver [18, 20, 40]. Our experimental evaluation shows that the average pathfinding time for *RACED* is 31 seconds and the average routing time is 3 milliseconds across 52,000 transactions that were recorded for an year. This shows the efficiency of *RACED* in particular.

Setup time analysis: The setup time for Blanc and SilentWhispers is equal (4.565 seconds) since their setup involves only the creation of the signing and verification keys for the nodes and the creation and verification of pseudonymous identities. This step can be parallelized. The setup time for SpeedyMurmurs is the highest, ≈ 5.9 hours since it involves computation of the embedding coordinates of nodes in the PCN. The one-time setup time for *RACED* is also significantly high, ≈ 4.12 hours since it involves an additional setup for the establishment of the DHT ring and the creation and verification of a node's long-term and pseudonymous identities.

Tradeoffs: *RACED* introduces a delay whenever the finger table of an RH needs to be updated in the event of another RH joining or leaving the DHT ring. In a DHT, at most $log(m)$ entries in a node's finger table can be distinct where m is the size of the digest obtained by the hashing the node's IP address. Updating an existing RH's (when another RH leaves the DHT) finger table or a newly joined RH creating a finger table involves the existing RH or the newly joined RH opening payment channels with their finger table entries. In the worst case, a RH night need to open $log(m)$ payment channels. This introduces a delay of α, where α is the time to process payment channel openings depending on the blockchain on which *RACED* will be deployed. The payment channels between RHs and their finger table entries can be opened in parallel. For BTC this delay varies from 60 to 90 minutes.

The payment channels opened by the RHs can be used for routing multiple transactions, which amortizes the delay α over several thousands of transactions. Our evaluations show this delay being amortized over 52,000 transactions with an average pathfinding time of thirty one seconds and an average routing time of three milliseconds. In other words, the setup time of *RACED*, which is close to four hours is amortized over routing 52,000 transactions being routed in three milliseconds. In addition to this, updating the maximum amount at the end of each time epoch (δ) introduces a delay of β, which is the time taken for a RH and a corresponding finger table entry to sign the maximum amount that can be transacted between them. This makes the total delay introduced by *RACED* as $(\alpha + \beta)$.

8 *RACED* SECURITY ANALYSIS

In this section, we provide a formal analysis of *RACED*. We define an ideal functionality $\mathsf{F}_{\mathsf{RACED}}$, that consists of six functionalities: $\mathcal{F}_{init}$, $\mathcal{F}_{DHT}$, $\mathcal{F}_{aux}$ $\mathcal{F}_{Findpath}$, $\mathcal{F}_{Payment}$, and $\mathcal{F}_{htlc}$, and two helper functionalities $\mathcal{F}_{sig}$ [13], and $\mathcal{F}_{\mathcal{PCN}}$ [34]. The definition of the ideal functionalities and the proof of the following theorem is given in the full version of the paper [25].

Theorem 8.1. *Let* $\mathsf{F}_{\mathsf{RACED}}$ *be an ideal functionality for RACED. Let* $\mathcal{A}$ *be a probabilistic polynomial-time (PPT) adversary for RACED, and let* $\mathcal{S}$ *be an ideal-world PPT simulator for* $\mathsf{F}_{\mathsf{RACED}}$*. RACED UC-realizes* $\mathsf{F}_{\mathsf{RACED}}$ *for any PPT distinguishing environment* $\mathcal{Z}$*.*

9 CONCLUSION

In this paper, we have designed *RACED*, a PCN pathfinding and routing protocol that uses distributed hash tables to route transactions in PCNs. Our protocol does not need the presence of a trusted third party, is fully decentralized and can route concurrent transactions. *RACED* also ensures the privacy of sender and receiver, and atomicity of payments. We have demonstrated the efficiency of raced *RACED* by evaluating it on real-world transaction data, and have proven the security of *RACED* in the UC framework. The ideas presented in *RACED* can potentially be leveraged to decentralized networks in diverse domains such as edge computing and IoT networks.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Award No. 2148358, 1914635, and the Department of Energy. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and the Department of Energy.

REFERENCES

[1] Moaath Alshaikh and Akram Morie. 2022. Development of Multipath Dynamic Address Routing Protocol in MANET to Improve Data Transfer in Poor Infrastructure Environment. In *2022 International Conference on Computer Science and Software Engineering (CSASE)*. 368–373. https://doi.org/10.1109/CSASE51777.2022.9759630

[2] Abdelrahaman Aly. 2015. *Network flow problems with secure multiparty computation*. Ph.D. Dissertation. Catholic University of Louvain, Louvain-la-Neuve, Belgium.

[3] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. 2013. Securely solving simple combinatorial graph problems. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*. Springer, 239–257.

[4] Abdelrahaman Aly and Mathieu Van Vyve. 2015. Securely Solving Classical Network Flow Problems. In *Information Security and Cryptology - ICISC 2014*, Jooyoung Lee and Jongsung Kim (Eds.). Springer International Publishing, Cham, 205–221.

[5] Abdelrahaman Aly and Mathieu Van Vyve. 2016. Practically Efficient Secure Single-Commodity Multi-market Auctions. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9603)*, Jens Grossklags and Bart Preneel (Eds.). Springer, 110–129. https://doi.org/10.1007/978-3-662-54970-4_7

[6] Hakem Beitollahi and Geert Deconinck. 2008. Comparing Chord, CAN, and Pastry overlay networks for resistance to DoS attacks. In *2008 Third International Conference on Risks and Security of Internet and Systems*. 261–266. https://doi.org/10.1109/CRISIS.2008.4757488

[7] Bitkan Explorer [n.d.]. Bitkan Explorer. https://bit.ly/3LTPiYN

[8] Blockchair [n.d.]. Blockchair. https://blockchair.com/ethereum

[9] BTC, ETH tx throughput [n.d.]. BTC, ETH tx throughput. https://academy.binance.com/en/glossary/transactions-per-second-tps

[10] BTC market cap [n.d.]. BTC market cap. https://coinmarketcap.com/currencies/bitcoin/

[11] BTC tx throughput [n.d.]. BTC tx throughput. https://www.blockchain.com/explorer/charts/transactions-per-second

[12] BTCPAY Server [n.d.]. BTCPAY Server. https://bit.ly/3q3oAlU

[13] Ran Canetti. 2004. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* IEEE, 219–233.

[14] Yanjiao Chen, Yuyang Ran, Jingyue Zhou, Jian Zhang, and Xueluan Gong. 2022. MPCN-RP: A Routing Protocol for Blockchain-Based Multi-Charge Payment Channel Networks. *IEEE Transactions on Network and Service Management* 19, 2 (2022), 1229–1242. https://doi.org/10.1109/TNSM.2021.3139019

[15] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[16] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. 2008. *Algorithms*. McGraw-Hill Higher Education New York.

[17] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. 2020. Splitting Payments Locally While Routing Interdimensionally. *IACR Cryptol. ePrint Arch.* 2020 (2020), 555.

[18] Experian. 2023. Zelle limit. https://bit.ly/49lkpr6.

[19] Flare PCN [n.d.]. https://flare.xyz/the-flare-network/

[20] Forbes. 2023. xoom. https://bit.ly/47iPMkt.

[21] A V Goldberg and R E Tarjan. 1986. A New Approach to the Maximum Flow Problem. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (Berkeley, California, USA) *(STOC '86)*. Association for Computing Machinery, New York, NY, USA, 136–146. https://doi.org/10.1145/12130.12144

[22] Qianyun Gong, Chengjin Zhou, Le Qi, Jianbin Li, Jianzhong Zhang, and Jingdong Xu. 2021. VEIN: High Scalability Routing Algorithm for Blockchain-based Payment Channel Networks. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 43–50. https://doi.org/10.1109/TrustCom53373.2021.00024

[23] Hsiang-Jen Hong, Sang-Yoon Chang, and Xiaobo Zhou. 2022. Auto-Tune: Efficient Autonomous Routing for Payment Channel Networks. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. 347–350. https://doi.org/10.1109/LCN53696.2022.9843633

[24] Heba Kadry and Yasser Gadallah. 2021. A Machine Learning-Based Routing Technique for Off-chain Transactions in Payment Channel Networks. In *2021 IEEE International Conference on Smart Internet of Things (SmartIoT)*. 66–73. https://doi.org/10.1109/SmartIoT52359.2021.00020

[25] Kartick Kolachala, Mohammed Ababneh, and Roopa Vishwanathan. 2023. RACED: Routing in Payment Channel Networks Using Distributed Hash Tables. arXiv:2311.17668 [cs.CR]

[26] Lightning Network [n.d.]. Lightning Network. https://lightning.network/

[27] Lightning Network Fees [n.d.]. Lightning Network Fees. https://github.com/lightning/bolts/blob/master/07-routing-gossip.md#htlc-fees

[28] Lightning Pool [n.d.]. Lightning Pool. https://lightning.engineering/lightning-pool-whitepaper.pdf

[29] Siyi Lin, Jingjing Zhang, and Weigang Wu. 2020. FSTR: Funds Skewness Aware Transaction Routing for Payment Channel Networks. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 464–475. https://doi.org/10.1109/DSN48063.2020.00060

[30] LN minimum payment [n.d.]. LN minimum payment. https://dci.mit.edu/lightning-network

[31] LND HTLC [n.d.]. LND HTLC. https://docs.lightning.engineering/the-lightning-network/multihop-payments/hash-time-lock-contract-htlc

[32] LND keypair [n.d.]. LND keypair. https://github.com/lightning/bolts/blob/master/08-transport.md

[33] LND message passing [n.d.]. LND message passing. https://github.com/lightning/bolts/blob/master/07-routing-gossip.md

[34] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 455–471.

[35] Giulio Malavolta, Pedro A. Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2016. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks. *IACR Cryptol. ePrint Arch.* 2016 (2016), 1054.

[36] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 53–65.

[37] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. 2017. Sprites: Payment Channels that Go Faster than Lightning. (02 2017).

[38] Saleh Khalaj Monfared and Saeed Shokrollahi. 2023. DARVAN: A fully decentralized anonymous and reliable routing for VANets. *Computer Networks* 223 (2023), 109561.

[39] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[40] Nerdwallet. 2023. Xoom limit minimum. https://bit.ly/3QnPOAu.

[41] Networkx library [n.d.]. Networkx library. https://networkx.org/

[42] Gaurav Panwar, Satyajayant Misra, and Roopa Vishwanathan. 2019. Blanc: Blockchain-based anonymous and decentralized credit networks. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. 339–350.

[43] Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. 2021. LightPIR: Privacy-Preserving Route Discovery for Payment Channel Networks. In *2021 IFIP Networking Conference (IFIP Networking)*. 1–9. https://doi.org/10.23919/IFIPNetworking52078.2021.9472205

[44] Ripple [n.d.]. Ripple. https://ripple.com/

[45] Ripple API [n.d.]. Ripple API. https://data.ripple.com/

[46] Ripple current cap [n.d.]. Ripple current cap. https://www.slickcharts.com/currency

[47] Ripple market value [n.d.]. Ripple market capitalization. https://bit.ly/3AGVnT0

[48] Ripple message passing [n.d.]. Ripple message passing. https://ripple.com/files/ripple_consensus_whitepaper.pdf

[49] Ripple trustline API [n.d.]. Ripple trustline API. https://xrpl.org/account_lines.html

[50] Stefanie Roos, Martin Beck, and Thorsten Strufe. 2016. Voute-virtual overlays using tree embeddings. *arXiv preprint arXiv:1601.06119* (2016).

[51] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2018. Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society.

[52] Antony Rowstron and Peter Druschel. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 329–350.

[53] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathy Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Mohammad Alizadeh, and Giulia Fanti. 2020. High Throughput Cryptocurrency Routing in Payment Channel Networks. arXiv:1809.05088 [cs.NI]

[54] Mudhakar Srivatsa and Ling Liu. 2009. Mitigating Denial-of-Service Attacks on the Chord Overlay Network: A Location Hiding Approach. *IEEE Transactions on Parallel and Distributed Systems* 20, 4 (2009), 512–527. https://doi.org/10.1109/TPDS.2008.125

[55] Stellar Network [n.d.]. Stellar Network. https://www.stellar.org/?locale=en

[56] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (San Diego, California, USA) *(SIGCOMM '01)*. Association for Computing Machinery, New York, NY, USA, 149–160. https://doi.org/10.1145/383059.383071

[57] Lalitha Muthu Subramanian, Roopa Vishwanathan, and Kartick Kolachala. 2020. Balance transfers and bailouts in credit networks using blockchains. In *2020 IEEE*

International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, 1–3.
[58] Hieu Tran, Miao Miao, Farokh Bastani, and I-Ling Yen. 2023. Multi-Keyword Based Information Routing in Peer-to-Peer Networks. In *2023 International Conference on Information Networking (ICOIN)*. 791–796. https://doi.org/10.1109/ICOIN56518.2023.10049045
[59] Zied Trifa. 2019. Preventing Sybil attacks in chord and Kademlia protocols. *International Journal of Internet Protocol Technology* 12, 3 (2019), 157–166. https://doi.org/10.1504/IJIPT.2019.101364 arXiv:https://www.inderscienceonline.com/doi/pdf/10.1504/IJIPT.2019.101364
[60] VISA transactions [n.d.]. VISA transactions. https://www.visa.co.uk/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf Accessed: 2023-02-23.
[61] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: Efficient Dynamic Routing for Offchain Networks *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 370–381. https://doi.org/10.1145/3359989.3365411
[62] Ruozhou Yu, Guoliang Xue, Vishnu Kilari, Dejun Yang, and Jian Tang. 2018. CoinExpress: A Fast Payment Routing Mechanism in Blockchain-Based Payment Channel Networks. 1–9. https://doi.org/10.1109/ICCCN.2018.8487351
[63] Saleem Zahid, Kifayat Ullah, Abdul Waheed, Sadia Basar, Mahdi Zareei, and Rajesh Roshan Biswal. 2022. Fault tolerant DHT-based routing in MANET. *Sensors* 22, 11 (2022), 4280.
[64] Xiaoxue Zhang, Shouqian Shi, and Chen Qian. 2021. WebFlow: Scalable and Decentralized Routing for Payment Channel Networks with High Resource Utilization. *CoRR* abs/2109.11665 (2021). arXiv:2109.11665 https://arxiv.org/abs/2109.11665
[65] Yuhui Zhang and Dejun Yang. 2019. RobustPay: Robust Payment Routing Protocol in Blockchain-based Payment Channel Networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. 1–4. https://doi.org/10.1109/ICNP.2019.8888094
[66] Yuhui Zhang and Dejun Yang. 2021. RobustPay+: Robust Payment Routing With Approximation Guarantee in Blockchain-Based Payment Channel Networks. *IEEE/ACM Transactions on Networking* 29, 4 (2021), 1676–1686. https://doi.org/10.1109/TNET.2021.3069725
[67] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. 2004. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications* 22, 1 (2004), 41–53.

A EXTENDED RELATED WORK

The distributed version of Dijkstra's shortest path algorithm presented in [2–4] requires each vertex in the graph to reveal the vertices it is not connected to, to an "algorithm designer" who runs the algorithm. Revealing this information eventually reveals the entire network topology and cannot be leveraged to PCNs because of privacy violations.

The solution presented in [2–4] for the minimum cost-flow problem also has the same assumptions as the distributed version of the Dijkstra's shortest path algorithm.

The minimum mean cycle problem presented in [2–4] is orthogonal to our work since it focuses on finding cycles in graph which have the least number of edges, whereas *RACED* focuses on performing secure routing in PCNs. The idea proposed by Abdelrahman *et al.* in [5] proposes an auction mechanism in which sellers sell the maximum flow that is transmittable through them and the bidders bid for these maximum flows. This idea assumes the total amount of flow transmittable through the network (the network throughput) to be public and also requires a trusted entity called "control agency" that oversees the auction. This idea cannot be leveraged to perform secure routing in PCNs. PCNs are distributed networks where having a central root of trust is not possible. Revealing the complete throughput of the network for PCNs is a violation of privacy.

DHTs were developed initially to facilitate file-sharing among a set of cooperating peers [36, 52, 56, 67]. DHTs are also being explored for solving routing challenges in MANETs (Mobile Ad-hoc Networks), VANETs (Vehicular Ad-hoc Networks), and for data sharing across IoT (Internet Of Things) devices [1, 38, 58, 63]. However, in the case of PCNs, nodes (peers) do not share data/files but send and receive money. Peer-to-peer routing protocols that use DHT do not take part in any payment channel opening/closing, do not interact with a blockchain, and do not route payments among each other. Finally, in DHTs it suffices if a node is able to locate another node in the network for peer-to-peer communication. However in PCNs, in addition to finding efficient paths between nodes, the paths should also have enough liquidity to route the amount specified by the sender.

B OVERVIEW OF CHORD

Chord [56] is a scalable, peer-to-peer, distributed lookup protocol that locates a node that stores a particular data item in p2p networks. It uses a consistent hashing mechanism that enables the lookup to be completed in time that is logarithmic in the number of nodes present in the DHT ring. The nodes in Chord are placed in the form of a circle called the identifier circle. Each node hashes its IP address to produce an m bit digest that acts as its node identifier, denoted by the numbers next to each node in Figure 3. Each node in the Chord ring in Figure 3 is responsible for storing a key (represented as a digest) that points to a certain fragment of data. This key, k is the digest obtained by hashing the identifier of the key with the same hash function that was used to create the node identifiers. Each key k will be assigned to the node whose identifier is equal to or follows the identifier of k in the identifier circle. Each node in the Chord ring maintains a look-up table called *finger table* that contains at most m entries with $log(m)$ being distinct. Each node also maintains a table containing its first $log(n)$ successors, called the successor table. The first entry in a node's finger table is the node's immediate successor in the identifier circle. Consider Figure 3, where seven nodes are a part of a Chord ring. The finger table entries of a node identified by i are computed thus: $(i + 2^{(i-1)} \mod m)$. If we set $m = 6$, the finger table entries of node Charlie are: $(26 + 2^0 \mod 2^6)$, $(26 + 2^1 \mod 2^6)$, $(26 + 2^2 \mod 2^6)$, $(26 + 2^3 \mod 2^6)$, $(26 + 2^4 \mod 2^6)$, $(26 + 2^5 \mod 2^6)$ which gives us the set of identifiers {27, 28, 30, 34, 38, 61}, which map to nodes [Amit, Amit, Amit, Amit, Jill and Garcia]. In case the identifier is not assigned to any node in the Chord ring (27 in this example), the corresponding finger table entry would be the next node in the ring whose identifier is greater than 27, in this case, Amit. In Chord, when a node receives a request to locate a key k that is not in its possession, it forwards the request to the closest predecessor of k in its finger table. For example, if Denise wants to resolve a query to locate node Jill, Denise needs to locate the node that precedes Jill in the Chord ring, which is Amit. Now from Denise's finger table shown in Figure 3, the node closest to Amit (based on node identifiers) in Alice's finger table, which contains [Rajiv, Rajiv, Rajiv, Rajiv, Rajiv, Amit] is Amit himself (Amit is in the finge table of Denise), and Jill is the first node in the finger table of Amit which contains [Jill, Daniela, Garcia, Denise]. Hence the distance between Denise and Amit is greater than the distance between Amit and Jill, so Amit is closer to Jill than Denise. Hence Denise passes the request of locating Jill, to Amit. Since the finger table of Amit contains [Jill, Daniela, Garcia, Denise], Jill reaches Amit in one hop. In this manner, the number of steps is halved every time a node locates another node in the identifier circle. This

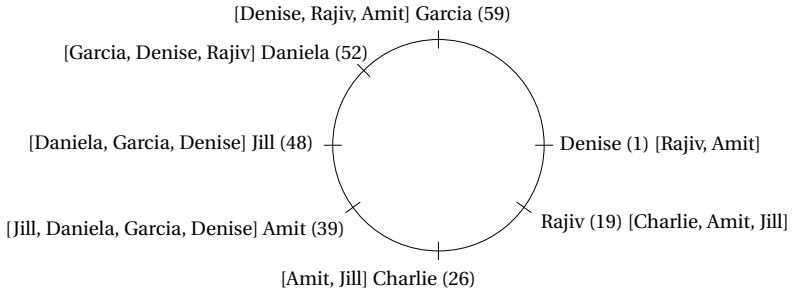

Figure 3: An example Chord ring with 7 routing helpers. The values in parenthesis adjacent to the node represents the node identifier. The values in the square brackets [...] represent finger table entries. In each finger table, we only show unique entries.

reduces the lookup time to $log(n)$, where n is the number of nodes in the Chord ring.

C PROTOCOLS

Protocol 6: Key Setup

for $i = 1; i \leq n; i++$ **do**
  node i does $\mathsf{KeyGen}(1^\lambda) \rightarrow \mathsf{sk}_i, \mathsf{vk}_i$
  /* creating temporary identities */
  node i does $\mathsf{KeyGen}(1^\lambda) \rightarrow \mathsf{SK}_i, \mathsf{VK}_i$
  node i does $\mathsf{Sign}_{\mathsf{sk}_i}(\mathsf{VK}_i) \rightarrow \sigma_{\mathsf{VK}_i}$
  node i calls $\mathsf{RetrieveNeighbors}(\mathsf{vk}_i) \rightarrow \mathbb{I}_i$
  node i sends vk_i to all the nodes in $\mathbb{I}_i$
  for $j=1;j\leq |\mathbb{I}_i|;j++$ **do**
    if $\mathsf{Verify}_{\mathsf{vk}_i}(\mathsf{VK}_i, \sigma_{\mathsf{VK}_i}) \rightarrow 0$ **then**
      j returns $\perp$
    else
      $\mathsf{PC.Open}(\mathsf{VK_i}, \mathsf{VK}_j, lw_{i,j}, lw_{j,i})$

Key Setup, Protocol 6: This protocol handles the generation of long-term and temporary identities for all the nodes in the PCN. These keys are used to sign and verify messages in the subsequent protocols of *RACED*. Initially, all the nodes create their temporary and long-term signing and verification keypairs, denoted by (SK, VK) and (sk, vk) respectively using the KeyGen function. All the nodes in the PCN send their long-term verification key to their immediate neighbors. Nodes that are immediate neighbors of each other exchange payments in the PCN and hence they need to know each other's real identities. The temporary verification key of each node is signed using the long-term signing key to produce a signature. This signature ties the long-term identity of a node to its temporary identity. This signature is then verified by all the immediate neighbors of a node using the node's long-term verification key. If the signature verifies, the two nodes open a payment channel. The creation of temporary identities is done to hide the real identity of a node in the PCN from its non-neighboring nodes. This helps us achieve our goal of *sender/receiver privacy*.

Routing Payment, Protocol 7: This protocol facilitates the routing of payment between Alice and Bob. Initially, Alice retrieves the *endRH* from each path tuple sent to her in the stack $\mathbb{P}$ by the *nearRH* at the end of Protocol 3. The node identifier of the *endRH* is the last value in each tuple present in $\mathbb{P}$. These *endRH*s are added to the list $\mathbb{T}$ that Alice maintains locally. Alice sends this list to Bob using a secure out-of-band communication channel. Bob picks one *endRH* that is closest to him based on the minimum hop count between him and the *endRH*. Bob notifies Alice regarding his choice of *endRH* using the same channel. Alice chooses the shortest path that contains the RH picked by Bob as the *endRH* using the ChoosePath function. This function returns the path, $\mathbb{P}'$. Bob samples a random pre-image X, hashes it to produce a digest Y, and sends Y to Alice. For each consecutive node along the path from Alice to Bob, every node sends the tuple (inPath, Y, *txid*) to its immediate neighbor.

Protocol 7: Routing Payment

Alice maintains a list $\mathbb{T} = \emptyset$
for $i=1;i \leq |\mathbb{P}|;i++$ **do**
  Alice performs $\mathbb{T}.\mathsf{Add}(endRH_i)$
Alice sends $\mathbb{T}$ to Bob out-of-band and Bob picks and sends $endRH_{Bob} = \min(\mathsf{hc}_{endRH_i,Bob})\ \forall\ i \in \mathbb{T}$
Alice calls $\mathsf{ChoosePath}(\mathbb{P}, endRH_{Bob}) \rightarrow \mathbb{P}'$
Bob does $X \leftarrow\$ \{0,1\}^\lambda$, $H(X) \rightarrow Y$ and sends Y to Alice
for *each pair of consecutive nodes i, j along the path of txid from Alice to Bob* **do**
  Alice retrieves the *txid* for the transaction to be sent to Bob from the tuple $\mathbb{K} = (\cdot, txid, \cdot)$
  previous = i, next = j
  previous sends (inPath, Y, *txid*) to next and
  previous establishes HTLC with next and previous = next and next = previous +1
for *every pair of consecutive nodes along the path of txid from Bob to Alice* **do**
  if previous *reveals X to* next **then**
    if $\mathsf{HTLC.Pay}(\mathsf{vk}_{\mathsf{previous}}, \mathsf{vk}_{\mathsf{next}}, txid, \mathsf{amt}) \rightarrow$ *success* **then**
      previous = next and next = previous -1
    else
      return $\perp$
  else
    return $\perp$

Upon receiving this tuple, every node along the path establishes an HTLC with its immediate neighbor. Every pair along the path from Bob to Alice reveals the secret used for the HTLC. Upon successful revealing of this secret from every node to its immediate neighbor along the path from Bob to Alice, the payment process is completed. Using HTLCs ensures that no honest party loses any funds because of malicious behavior by other parties in the system, which helps us in achieving our goal of *balance security*. HTLCs also ensure that all the link weights of the nodes along the transaction path go back to the state they were in prior to the commencement of the transaction if the transaction fails for any reason. This achieves our goal of *atomicity*.