# Route Discovery in Private Payment Channel Networks

Zeta Avarikioti[1], Mahsa Bastankhah[2], Mohammad Ali Maddah-Ali[2], Krzysztof Pietrzak[1], Jakub Svoboda[1], and Michelle Yeo[1]

[1] IST Austria
{zetavar, krzysztof.pietrzak, jakub.svoboda, michelle.yeo}@ist.ac.at
[2] Sharif University of Technology, Iran
mahsa.bastankhah@ee.sharif.edu, Maddah_ali@sharif.edu

**Abstract.** In this work, we are the first to explore route discovery in private channel networks. We first determine what "ideal" privacy for a routing protocol means in this setting. We observe that protocols achieving this strong privacy definition exist by leveraging (topology hiding) Multi-Party Computation but they are (inherently) inefficient as route discovery must involve the entire network. We then present protocols with weaker privacy guarantees but much better efficiency. In particular, route discovery typically only involves small fraction of the nodes but some information on the topology and balances – beyond what is necessary for performing the transaction – is leaked. The core idea is that both sender and receiver gossip a message which then slowly propagates through the network, and the moment any node in the network receives both messages, a path is found. In our first protocol the message is always sent to all neighbouring nodes with a delay proportional to the fees of that edge. In our second protocol the message is only sent to one neighbour chosen randomly with a probability proportional to its degree. While the first instantiation always finds the cheapest path, the second might not, but it involves a smaller fraction of the network. We also discuss some extensions to further improve privacy by employing bilinear maps.
Simulations of our protocols on the Lightning network topology (for random transactions and uniform fees) show that our first protocol (which finds the cheapest path) typically involves around 12% of the 6376 nodes, while the second only touches around 18 nodes ($< 0.3\%$), and the cost of the path that is found is around twice the cost of the optimal one.

**Keywords:** Payment Channel Networks · Privacy · Bitcoin · Route Discovery

## 1 Introduction

Payment channel networks (PCNs) are one of the most promising approaches in making cryptocurrencies scalable. They allow any pair of users to set up a payment channel between them, thereby enabling an unlimited number of costless transactions between them without requiring consensus on the blockchain. Users who are not directly connected to each other with a payment channel can still transact with each other by routing the transaction through intermediate nodes in the network. These intermediate nodes typically charge a fee for forwarding these transactions. There are several PCN proposals [27,10,21,9,13,12,4,11,17]; two of the most widely used are the Bitcoin Lightning Network [21] and Raiden [2] for Ethereum.

The route discovery problem is the problem of discovering "short" (i.e. low-cost) routes in the network. This is a well-researched problem with several existing solutions [22,15,25,26,28,20]. However all these solutions assume that the *entire topology* of the payment network is known by at least one party (for instance the users who download the entire network [29] or trampoline nodes [22]).

**Unknown Network.** In this work we consider the problem of route discovery in PCNs where the capacities, fees and even the mere *existence* of channels can be partially unknown. This is motivated by the fact that in a PCN the topology and capacities of a payment network constantly change. Moreover the exact capacities are typically kept private as an adversary who can observe the exact channel capacities could easily reconstruct what payments happened in the network [19]. The Lightning Network will soon allow private channels using taproot [7]. The basic idea is to use Schnorr signatures [16] to aggregate public keys and signatures, making a transaction involving multiple users indistinguishable from a transaction involving just two users on the blockchain.

**Efficiency and Privacy of Route Finding.** The route discovery protocols we propose do not assume any knowledge about the network other than the minimal requirement that nodes know about their own channels. In this setting, the only thing a sender and receiver can do to find an (optimal) route is to send exploratory messages through the network.

Our goal is to construct protocols that are efficient, private and find a path whose fees are minimal, or at least close to it. An important metric is the fraction of the network that may typically be involved in any route finding attempt as few involved nodes directly translate into better efficiency and also better privacy. Concerning privacy, even the users that are involved in the path finding process should learn as little as possible about senders, receivers and amounts of route finding attempts. Moreover nodes that end up in a payment path should ideally not learn any information beyond the amount and the nodes right before and after them in the path, even the sender and receiver jointly should not learn the users on a path other than their direct neighbours.

**Our Contribution.** To the best of our knowledge, this is the first work which considers private route discovery in a PCN with *private* channels. Our work makes the following contributions:

- (Ideal Notion) We put forward a security notion for private route discovery and give a feasibility result using multi-party computation (MPC). Our notion is inspired by security notions from topology hiding MPC. This solution is inefficient, not just because MPC computations are expensive, but also because it must involve the *entire* network (and this inherent for any protocol achieving our ideal notion).
- (Practical Protocols) We present a family of route discovery protocols on private PCNs that are much more efficient and in particular only touch a small fraction of the network. These protocols work by propagating exploratory messages from the sender and receiver through the PCN. When an intermediary node receives both messages, a path is found. The first protocol we propose is Forward-to-All where nodes forward messages on all their edges but one every edge, with a delay that is proportional to the fee on that edge. In our second protocol Degree-Proportional Random Walk nodes just send messages to one neighbour, chosen randomly with a probability proportional to their degree. Forward-to-All always finds the shortest path, but it involves a larger fraction of the network than Degree-Proportional Random Walk.
- (Simulations) We simulated our protocols on the Lightning Network and a certain class of graphs (Barabási–Albert) that are used to model PCNs. Our simulation show that Forward-to-All typically involves around 800 of the 6000 nodes in Lightning, while Degree-Proportional Random Walk only involves around 20, and the paths that are found are around twice as long as the optimal ones.
- (Analysis) We prove some analytical bounds for our algorithms on particular classes of graphs.

## 2 Model and Definitions

We model a payment channel network (PCN) as a directed graph $G = (V, E)$ where each node in the set $V$ represents a user in the PCN and an edge $(u, v)$ in the set $E$ indicates an open channel between the users $u$ and $v$ in $V$. We denote with $f_{u,v}(.)$ the fee function, i.e., $u$ charges $f_{u,v}(x)$ to transfer $x$ coins over the channel $(u, v)$. In existing PCNs like the Lightning Network, $f_{u,v}(.)$ is set by $u$.

The route discovery problem in a PCN represents the task of finding the path with the smallest aggregated fees, or the cheapest path, in a PCN for a given pair of sender/receiver nodes $u_s, u_r \in V$ and amount $x$, i.e., a path $(u_0 = u_s, u_1, \ldots, u_\ell = u_r)$, minimizing the aggregated fees, as

$$\sum_{i=1}^{\ell} f_{u_{i-1},u_i}(x + \phi_{i-1}).$$

In above formula, $\phi_i$, $i = 0, \ldots, \ell - 1$, is the aggregated fees that nodes $u_{i+1}, u_{i+2}, \ldots, u_{\ell-1}$ charge. More precisely,

$$\phi_{i-1} = \phi_i + f_{u_i,u_{i+1}}(x + \phi_i).$$

Since the receiver $u_\ell$ is the last node in the path, $\phi_{\ell-1} = 0$. We use the notation $\mathsf{shortestPath}_G(u_s, u_r, x) \mapsto \{u_0 = u_s, u_1, \ldots, u_{\ell-1}, u_\ell = u_r\}$ to describe the functionality that takes two nodes $u_s$ and $u_r$ and a transaction amount $x$ as inputs and outputs the cheapest path between those two nodes.

## 3 Ideal Privacy

In this section we define an ideal notion of privacy for route discovery and outline how to construct protocols achieving this notion, albeit very impractical ones.

Privacy is ideal, this means that each party only learns the bare minimum information required to participate in the transaction: its predecessor and successor on the payment path and the amount to be transferred. This information is minimal, if we assume that users know at the very least the current balances on their own channels (as in Lightening Network). In this case they learn the predecessor, successor, and amount of a transaction they were involved in by simply comparing the balances on their channels before and after the transaction.

For our ideal security, we only consider path finding protocols $\Pi$, which always find the cheapest path. Defining ideal privacy for protocols, which output a not necessarily cheapest path seems more complicated, as the privacy loss depends on the path that was found. We also only consider passive adversaries, that is, an adversary can corrupt users and learn their internal state, but not make them deviate from honestly executing the protocol (say, by providing wrong or inconsistent input).

Let us stress that we only consider the route discovery process. Once a path is found, execution of the transaction itself using hash lock time contracts (HTLCs) will inevitably leak some more information (on the position of the corrupted users in the cheapest path).

Our privacy notion is inspired by the Indistinguishability under Chosen Topology Attack (IND-CTA) security definition from work on topology-hiding multiparty computation [18].

We consider an adversary that initially chooses two networks and a transaction for each of them, and also a subset of nodes to corrupt. We then require that given the view of the corrupted nodes after the path finding protocols has been executed on one of the two networks, an adversary cannot determine which. Of course we must require that the adversary chooses the networks, transactions, and corrupted nodes such that the corrupted nodes have the same neighbours and fee functions, and the final output of the corrupted nodes (either they are not on the path, and if, they learn their predecessor, successor and amount to be transferred) is identical in both cases, otherwise distinguishing is trivial for any protocol as one can distinguish using just the initial view and final output of the protocol. We give a more formal definition below.

**The Ideal Privacy Security Game.** We consider a security game involving an adversary $\mathcal{A}$ against a path-finding protocol $\Pi$. The protocol is run by the players $V$ on a network $G = (V, E)$. Each player initially gets as inputs its neighbours and fee functions.

When the protocol starts, two players $u_s, u_r$ get as extra input $(u_s, u_r, x)$ informing them they are, respectively, the sender and the receiver of some amount $x$. The correctness we require from our protocol is that every $u \notin \mathsf{shortestPath}_{G^b}(u_s^b, u_r^b, x)$ which is not on the cheapest path outputs $\perp$, while every $u$ on the path outputs its predecessor, successor and amount they transfer in this optimal path. The security game goes as follows:

– $\mathcal{A}$ chooses the following for $i \in \{0, 1\}$:
  1. A network (directed graph) $G^i = (V^i, E^i)$, where every edge $(u, v)$ is labelled with a fee function $f^i_{u,v}(.)$.
  2. A sender and receiver pair $(u_s^i, u_r^i)$ and amount $x$.

  $\mathcal{A}$ choses a subset $S \subset V_0 \cap V_1$ of nodes to corrupt. These nodes must have the same neighbourhood and fee functions in both networks, and their final output (predecessor, successor and amount) must be identical.
– We choose a random bit $b \in \{0, 1\}$ and run $\Pi$ on $G^b$ (with input $(u_s^b, u_r^b, x)$).
– $\mathcal{A}$ gets the transcripts of the corrupted nodes.
– $\mathcal{A}$ outputs a bit $b'$. If $b' = b$, $\mathcal{A}$ wins the game.

Let us call a path finding protocol $\Pi$ $\epsilon$-private if $\mathcal{A}$ wins the above game with probability at most $1/2 + \epsilon$, and private if it is $\epsilon$ secure for some negligible $\epsilon$.

**Protocols with Ideal Privacy from MPC.** If we assume a trusted third party $\mathcal{T}$ (that cannot be corrupted by the adversary and has a channel to every node in the network) we can trivially get a private path-finding protocol by simply letting each party in the network send their data to $\mathcal{T}$, which will then locally compute the cheapest path and send the output, i.e., either $\perp$ or the amount, successor and predecessor in the cheapest path, to every node.

To get an actual protocol without a trusted third party, we can instantiate $\mathcal{T}$ using a multi-party computation (MPC) protocol. MPC is a protocol between $N$ users $u_1, \ldots, u_N$ who agree on some outputs functions $f_1, \ldots, f_N$. Every user $u_i$ holds an input $x_i$, and at the end user $u_i$ learns some output $f_i(x_1, \ldots, x_N)$. The security requirement is that any coalition of users learns nothing about the inputs of the other users beyond what is revealed through their own outputs. As here the users need to share pairwise channels, they need to know about the other participants, which means in our security definition we need the users $V^1$ and $V^2$ in the two notworks to be identical $V^1 \equiv V^2$.

Our security notion is inspired by notions from topology-hiding MPC. While there the goal was to hide the topology of the communication channels, we assume pairwise channels but want to hide the topology of the payment network. But of course we could also use a topology-hiding MPC to instantiate $\mathcal{T}$, in which case we would only require communication channels between users that share a channel. In this setting one could potentially also achieve security in settings where $V^1 \neq V^2$ as nodes would only talk to their neighbours, and for the corrupted nodes these are identical.

**Executing the Transaction.** After the cheapest path is found, the users need to execute the transaction. This can be achieved using HTLCs and MAP-PCN [29], but it will reveal a bit of extra information (beyond the minimal information revealed through the path discovery protocol) to users on the cheapest path (and thus to the adversary should this node be corrupted). This additional information is about the position on the payment path. To illustrate this, assume in the security game the two cheapest paths were $u_s \to a \to b \to c \to u_r$ and $u_s \to a \to c \to u_r$ and the adversary corrupted $u_s$ and $u_r$. If $u_s, u_r$ now imitate the transaction using HTLCs, messages must be passed along the path, and this will take longer in the first network as there the path is longer. Thus, by simply looking at the timing the HTLC needs to pass through the network an adversary can distinguish whether it is in the first or second case, even though it could not do so after the path finding algorithm was executed.

## 4 A Family of Protocols

Consider a sender $u_s \in V$ who wants to transfer $x$ coins to a receiver $u_r \in V$ and thus needs to know a path for the transaction, ideally the cheapest one. In this section, we present a family of exploratory route discovery protocols that provide solutions to this problem. At its core, these protocols employ local probing: nodes send exploratory messages (originating at the sender and receiver nodes) to their neighbours who in turn propagate them. Our protocols only require nodes to know their incident channels, and some also require a degree estimate of each neighbour.

The protocols run in three phases, (1) exploration, which runs until the first node receives both messages, i.e., the one originating at the sender and the receiver. (2) notification, where the relevant nodes are informed that a path was found and (3) stopping, where the nodes currently participating in the exploration phase are informed so they do not propagate messages further. Phase (1) is running slow, i.e., messages are propagated with some delay which should be significantly larger than the typical network delay, while in phase (2) and (3) messages are relayed immediately. The main reasons why we need Phase (1) to be slow is so the messages in the stopping phase can easily "catch up" to the nodes which are in the exploration phase. We will also mention how this helps to improve correctness and even privacy.

On one extreme, we have Forward-to-All that involves nodes sending these exploratory messages to all their neighbours, where each message is delayed for some time proportional to the fees (which guarantees that the first path that is found is also the cheapest one). On the other end of the spectrum, we have a more parsimonious protocol, namely Degree-Proportional Random Walk, that only involves sending messages to a few neighbours. As the protocols in the family are similar, mostly differing in the rule on when and to whom to forward the exploratory messages, we first present a generic overview of Forward-to-All, before briefly describing how to modify Forward-to-All to get Degree-Proportional

Random Walk. We then suggest some improvements to boost the privacy of this family of protocols.

**Forward-to-All Exploration Phase.** In this protocol, both the sender $u_s$ and receiver $u_r$ create messages with a special identifier (so intermediate nodes who receive messages from $u_s$ and $u_r$ can associate them together), an amount $x$ that $u_s$ wants to send to $u_r$, as well as a tag Sender or Receiver which specifies whether they are sending or receiving the transaction. The sender and receiver then propagate these messages through the graph by sending these messages to all their neighbours who then in turn propagate the message to all their neighbours. At every step of the propagation, the nodes update the transaction amount with fees to reflect the amount they would want to get by forwarding the transaction, and they also wait before forwarding for some time that is linear in those fees. All nodes store the messages they received, as well as an id (not to be confused with identifier) of the node that sent them the message. The precise rule and fee computation differs, however, depending on whether a node gets a message from the sender or the receiver.

*Fee computation for messages from receiver* Apart from the receiver, each intermediate node $u_i$ upon receiving a message with the Receiver tag from another node $u_{i+1}$, updates the transaction amount to add a fee for sending the transaction amount along the channel $(u_i, u_{i+1})$. This is to reflect the fee $u_i$ would charge for forwarding the transaction to $u_{i+1}$. Figure 1 illustrates this process where the receiver $u_r$ sends a message with the transaction amount $x$ to all of $u_r$'s neighbours. Upon receiving the message from $u_r$, $u_{i+1}$ adds a fee of $f_{u_{i+1},u_r}(x)$ to the transaction $x$. Messages with this updated transaction amount of $x + f_{u_{i+1},u_r}(x)$ would be sent to all of $u_{i+1}$'s neighbours.

*Fee computation for messages from sender* The fee computation for the sender and the nodes that receive messages with the Sender tag is trickier. Although the sender knows the transaction amount $x$, they do not know the total amount they would have to send at the end of the protocol as it would include the fees along the path which is still unknown. Thus the sender would have to add an estimate of the total fee of the path, $\delta$, to the transaction amount in their initial message. Each node that receives a message with the Sender tag, updates the transaction amount to subtract a fee for each edge they propagate the message to. This is to account for the fees the node will charge to forward the transaction. For instance in Figure 1, the node $u_{i-1}$, upon receiving a message with transaction amount $x + \delta$, subtracts $f_{u_{i-1},v}(x + \delta)$ from the transaction amount before forwarding the message with this new transaction amount to the node $v$. The node $u_{i-1}$ does the same but subtracts $f_{u_{i-1},u_i}(x + \delta)$ from the transaction amount before sending the message to $u_i$.

*Delay time computation* Let $d$ be a publicly available delay function that maps fees to delay times. Let $d_{u,v}$ denote the delay time for the total fee for sending

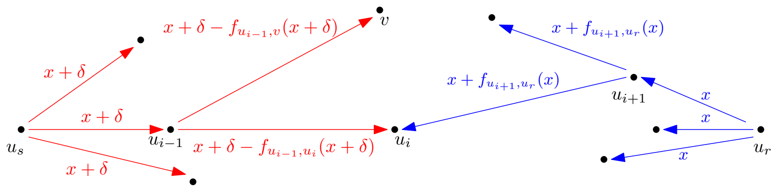


Fig. 1: Propagating exploratory messages from sender and receiver in the Forward-to-All protocol. Each directed edge $(u,v)$ is labelled with the transaction amount in the message that $u$ sends to $v$.

an arbitrary but fixed amount $x$ over the channel $(u,v)$, i.e. $d_{u,v}=d(f_{u,v}(x))$. Every node (except the sender and receiver) computes a delay time with the delay function $d$ and the fees computed as described above. In Figure 1 for instance, since the sender $u_s$ and receiver $u_r$ do not have fees, $u_s$ and $u_r$ will send their exploratory messages immediately. The node $u_{i+1}$ will wait $d_{u_{i+1},u_r}$ before forwarding the message to $u_i$, and $u_{i-1}$ will wait $d_{u_{i-1},u_i}$ before forwarding the message to $u_i$.

**Forward-to-All Notification Phase.** Upon receiving a new message, a node checks its identifier with the identifiers of the stored messages to see if the message identifiers can be associated together. When a node $u_i$ finds an association of identifiers that indicate two messages are from the sender and receiver of a given sender-receiver pair, $u_i$ begins a process of notifying the sender and the receiver that a path exists between them. We denote the two nodes that sent $u_i$ the associated sender and receiver messages by $u_{i-1}$ and $u_{i+1}$ respectively. We also denote the transaction amounts in these messages as $x_s$ and $x_r$ respectively. Then, $u_i$ immediately sends $u_{i-1}$ a message with the message identifier and amount $x_s$ (resp. $x_r$ to $u_{i+1}$). Both $u_{i-1}$ and $u_{i+1}$ then search for the nodes that sent them the messages with the same identifier and forward these messages to these nodes. Refer to Figure 2 for an illustration of the process.

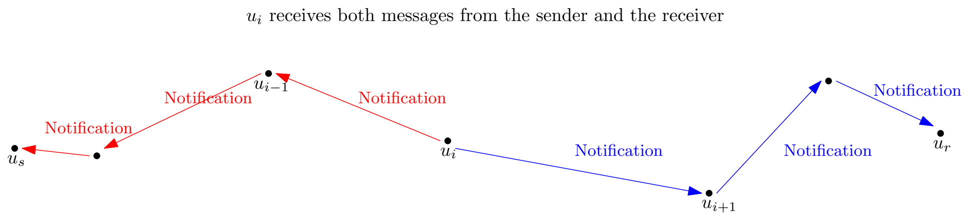


Fig. 2: Sending informative messages back to sender and receiver.

This process repeats itself until the sender and receiver get the message. At this point, the sender has enough information to proceed with the transaction. In particular, the sender can easily compute the total fee of the path from $x, x+\delta, x_s$ and $x_r$ (communicated by the receiver).

*Optimality of the discovered route* Using delay times guarantees that in Forward-to-All, either the notification message corresponding to the shortest path (subject to the accuracy of the fee estimation of the sender) always reaches the sender first, or the sender can find out if someone on the path deviated from the delay protocol. For example, let $L^*$ be the optimal path from $u_s$ to $u_r$ and $L'$ be a strictly more expensive path. Suppose several adversarial nodes on $L'$ immediately forward messages without delay and as a result, $u_s$ receives the notification message from an intermediate node on $L'$ first. Since $u_s$ knows the time they sent the first exploratory messages, $u_s$ can extract the fees from the message received and check if the total delay time on this path is larger than the difference of the current time and the time $u_s$ sent the first exploratory messages.

**Forward-to-All Stopping Phase.** When both sender and receiver are aware that a path exists between them and the sender is satisfied with the cost of the path, both sender and receiver can stop the protocol by sending a *stop message* with their identifiers to the nodes they sent the exploratory messages to. Nodes that have not yet sent the exploratory message to their neighbours would, upon receiving the stop message with the identifier, cancel the sending of the message. Nodes that have already send the exploratory messages would forward the stop message to the neighbours they sent the exploratory message to. This process is fast and thus will reach the slow propagation of the exploratory messages.

**Degree-Proportional Random Walk.** The Degree-Proportional Random Walk protocol is analagous to the Forward-to-All protocol with the exception that each node only forwards the message to *one* neighbour. Specifically, each node chooses a neighbour to forward the message to randomly with probability proportional to its degree. Thus, the messages are propagated according to two weighted random walks on the network, one starting from the sender and the other from the receiver, with the weight of any directed edge $(u, v)$ corresponding to the degree of $v$. We observe that due to the probabilistic nature of Degree-Proportional Random Walk, optimality of the discovered path is not guaranteed unlike in the case of Forward-to-All.

**Improving Privacy.** From the messages that originate at the sender and receiver and propagate through the network we only need the property that one can efficiently recognize once a message from both is received. The simplest solution is to simply sample some random nonce $I$ and propagate it together with a one bit tag specifying whether it is a sender or receiver originating message.

These messages are completely linkeable, this is unfortunate as it means that even if many path finding protocols are executed over the network at the same time, a potential adversary that controls some nodes in the network will still recognize with certainty which messages belong to the same path finding request, thus we do not leverage the fact that many protocols are running at the same time to improve privacy.

*Making messages unlinkeable using bilinear maps.* We can improve this situation a bit by using a bilinear map [14] $e : G_1 \times G_2 \to G_T$ (such a map allows "for one multiplication in the exponent" as $e(g_1^a, g_2^b) = g_T^{a \cdot b}$ where $g_1, g_2, g_T$ are generators of $G_1, G_2, G_T$) for a group where the DDH assumption holds in $G_1$ and $G_2$. Concretely, the sender and receiver sample a random $x$ and then the sender for every outgoing edge samples a random $r$ and propagates $(g_1^{x \cdot r} g_1^r)$ as the identifier, while the receiver propagates $(g_2^{x/r'}, g_2^{r'})$.

A node that receives $(g_1^{x \cdot r}, g_1^r)$ (similarly for the receiver tuples) propagates it only after re-randomizing it by exponentiating both elements with some fresh $r'$ which gives a tuple $(g_1^{x \cdot r''}, g_1^{r''})$ where $r'' = r \cdot r'$. This way an adversary (who does not know $x$) will not be able to distinguish a pair of tuples of the form $(g_1^{x \cdot r}, g_1^r), (g_1^{x \cdot r'}, g_1^{r'})$ from random, and thus cannot decide whether they belong to the same instantiation of the path finding protocol. Let us stress that the unlinkeability is limited as it only holds if the adversary gets to see either only messages originating at the sender or at the receiver, and this in inherent as we need parties who receive tuples $(a, b)$ and $(a', b')$ originating at both to efficiently recognize this. This can be done by checking whether $e(a, a') = e(b, b')$ as

$$(a, b) = (g_1^{x \cdot r}, g_1^r), (a', b') = (g_2^{r'/x}, g_2^{r'}) \Rightarrow e(a, a') = e(b, b') = g_T^{r \cdot r'}$$

*Quantising the transaction and encrypting the fees.* Messages that contain the exact transaction amount are also linkeable, even when fees are added, as the fees are typically miniscule compared to the transaction amount. To reduce this linkeability (at the cost of accuracy in fee estimation), the sender and receiver can quantise the amount of the transaction by rounding it up to a predefined value (for instance, a power of 2). Then, instead of adding the fees to the quantised transaction amount, nodes encrypt their fees using an additive homomorphic encryption scheme like additive ElGamal encryption.

Specifically, the sender and receiver use the exact (not quantised) transaction amount $x$ as their secret key and $g^x$ as their public key, where $g$ is the generator of a cyclic group of order $p$. Intermediary nodes would compute their fee using the quantised amount $x_q$. In this way the sender and receiver can guarantee that the path found would have sufficient capacity to forward the transaction $x$ since $x \leq x_q$. If a lot of protocols are running simultaneously, nodes would see a lot of messages with the same quantised transaction amount and so linkeability is reduced. The accuracy of the fee estimate would depend on how close $x$ is to $x_q$. However, since the transaction amount is typically a lot larger than the fees, we argue that making such an estimate is plausible.

## 5 Protocol Details

In this section we describe Forward-to-All in detail. We leave out the details of Degree-Proportional Random Walk as it is analogous. In the Forward-to-All protocol:

1. $u_s$ picks a random identifier $I$ from $\mathbb{Z}_p$
2. $u_s$ sends $I$ to the receiver $u_r$ through a secure communication channel.
3. $u_s$ and $u_r$ both send path discovery requests (PDRs) to all their neighbours. The PDR that $u_s$ sends is $\texttt{PDR}_s := (\textbf{identifier} : I, \textbf{amount} : x + \delta, \textbf{tag} : \textsf{Sender})$ and the PDR that $u_r$ sends is $\texttt{PDR}_r := (\textbf{identifier} : I, \textbf{amount} : x, \textbf{tag} : \textsf{Receiver})$.

Both $\texttt{PDR}_s$ and $\texttt{PDR}_r$ are gossiped through the network. Nodes execute either Protocol 1 or Protocol 2 depending on whether they receive $\texttt{PDR}_s$ or $\texttt{PDR}_r$ respectively. Note that a node $u_i$ may receive several $\texttt{PDR}_s$ (resp. $\texttt{PDR}_r$) with the same identifier. In this case $u_i$ should not forward the message, unless it has a lower fee than the previously forwarded one (in an ideal network with honest parties this case will never happen as the cheaper message will arrive first, in practice this could happen due to network delays or because some parties propagate too fast). In Degree-Proportional Random Walk the situation is different, here receiving the request a second time means the path closed a cycle, so it should be forwarded.

The moment a node $u_i$ receives two matching $\texttt{PDR}_s$ and $\texttt{PDR}_r$ (from say $u_{i-1}$ and $u_{i+1}$), $u_i$ waits for a delay time denoted by $\Delta$. The delay time $\Delta$ is calculated based on the following rule:

1. If $\texttt{PDR}_s$ arrives first, or both messages arrive at the same time, $\Delta = d_{u_i,u_{i+1}}$
2. If $\texttt{PDR}_r$ arrives first, $\Delta = \min(d_{u_i,u_{i+1}}, t)$ where $t$ is the time the $\texttt{PDR}_s$ arrives starting from the time $u_i$ receives the receiver message. If $t < d_{u_i,u_{i+1}}$, i.e. the message from the sender arrives while $u_i$ is still in the middle of the delay, then $u_i$ truncates the remaining delay period to $\Delta = \frac{d_{u_i,u_{i+1}} - t}{2}$.

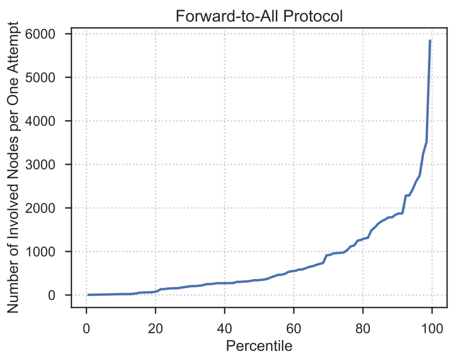
The reason why $u_i$ should incorporate a delay time of $\Delta$ before forwarding the notification messages is to account for the fees of $u_i$'s channels. After the delay, $u_i$ creates and sends a path found message (PFM) to these nodes. Specifically, $u_i$ sends $\texttt{PFM}_s = (\textsf{Identifier}, amount_s, \textsf{Sender}, \textsf{Path\_found\_message})$ to $u_{i-1}$ and $\texttt{PFM}_r = (\textsf{Identifier}, amount_r, \textsf{Sender}, \textsf{Path\_found\_message})$ to $u_{i+1}$. This is detailed in Protocol 3. Finally, Protocol 4 shows how the nodes that receive $\texttt{PFM}_s$ can update their information about the found path and continue forwarding $\texttt{PFM}_s$ until it reaches the sender. The protocol for $\texttt{PFM}_r$ is essentially analogous to Protocol 4 but in the opposite direction towards the receiver and hence we omit describing it.

Upon receiving the PFMs, both sender and receiver should send a stopping request SR to all their neighbours to halt the PDR forwarding procedure. The stopping request is of the form $\texttt{SR} := (\textbf{identifier} : I, \textbf{tag} : \textsf{Stopping\_Request})$. Upon first receiving a SR, a node $u_i$ immediately stops sending all PDRs. Then $u_i$ retrieves all the PDRs with the same identifier and forwards the SR to all neighbours that are saved as either a **previousNode** or **nextNode** in the PDR. In this way, every node that is involved in the protocol gets notified that a path exists and so stops forwarding messages corresponding to this transaction.

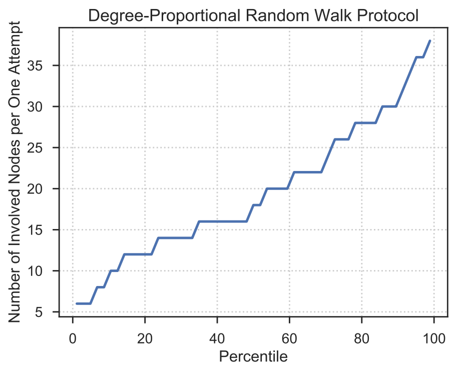
## 6 Evaluation

### 6.1 Efficiency

Our efficiency metric in this work is the average number of involved nodes in one route discovery attempt, i.e., the number of nodes who receive at least one $\mathtt{PDR}_s$ or $\mathtt{PDR}_r$. To compare the efficiency of Forward-to-All and Degree-Proportional Random Walk, we chose 100 random pairs of sender and receivers in a recent snapshot of the Lightning Network downloaded from [8], and ran both protocols (we used NetworkX [1], a python library for analysing complex networks). Figure 3a and Figure 3b shows the percentile distribution plot of the number of involved nodes in Forward-to-All and Degree-Proportional Random Walk respectively. As we can see from the plots, there is a significant difference between the efficiency of these two protocols. In Forward-to-All for instance, around 800 nodes out of 6376 nodes are involved on average while in Degree-Proportional Random Walk in Figure 3b, the average is around 18 nodes.



(a)



(b)

Fig. 3: Comparison of the percentile distribution plot of the number of involved nodes in Forward-to-All and Degree-Proportional Random Walk. We picked 100 sender-receiver pairs randomly and ran both protocols to find a route. The y-axis shows the number of involved nodes for each attempt and the x-axis corresponds to the percentile.

**Length of the Path** The fee-proportional delay function that we described in the Notification Phase of Section 4 guarantees that the first PFM that reaches the sender and the receiver corresponds to the shortest path in Forward-to-All. There is no such guarantee, however, for Degree-Proportional Random Walk due to its probabilistic nature. We compare the average path length found by Degree-Proportional Random Walk in the 100 runs as described above to the average actual shortest path length. As we see in Figure 4, in 80% of the runs the length of the path in Degree-Proportional Random Walk is a 2 approximation of the shortest path.

### 6.2 Privacy

The number of involved nodes is also a good measure of the privacy of our protocols, as nodes that receive messages can glean information from timings and the transaction amount. We note that the significantly smaller number of involved nodes in Degree-Proportional Random Walk can certainly be construed as an improvement of privacy compared to Forward-to-All. However, in a majority of the runs in our simulation of Degree-Proportional Random Walk, the involved nodes tend to be the same set of high degree nodes. Thus, if an adversary corrupts some of these nodes, they could potentially gain access to a lot more information compared to Forward-to-All. To mitigate this issue, nodes can "soften" the distribution of neighbouring nodes to select from (for instance, reducing the scale of the dependency on the degree from linear to logarithmic). This reduces the probability of always selecting nodes with high degrees.

### 6.3 Barabási–Albert Model

The Barabási–Albert model [6] (denoted by $BA(n, m)$) is a popular algorithm to create scale free networks using a preferential attachment mechanism (for details on the algorithm, see Appendix A). The number of nodes in the final network is $n$ and $m$ is the preferential attachment parameter. Many real world networks [5], including the Lightning Network [23], are characterized as scale-free. We run our protocols on Barabási–Albert graphs of varying sizes to see how these protocols would scale if the Lightning Network grows in the future. We used $BA(n, 1)$ because our implementation shows that the graph generated by $BA(n, 1)$ is more similar to the actual Lightening Network.

Figure 5 compares the number of involved nodes when running Forward-to-All and Degree-Proportional Random Walk on Barabási–Albert graphs with the number of vertices $n$ ranging from 2000 to 20000. Since the y-axis is logarithmic in Figure 5, we see that the number of involved nodes in Degree-Proportional Random Walk is an order of magnitude lower than in Forward-to-All. Indeed, in Section 8 we prove that the number of involved nodes in Degree-Proportional Random Walk grows sub-linearly with the number of nodes in the graph.

## 7 Discussion: Optimality and Efficiency Trade-off

In Forward-to-All, nodes forward PDRs to all their neighbours, and as a result a large fraction of the network is involved in the route-discovery process. On the other hand, reaching every node is the only way to guarantee the shortest path is always found. In Degree-Proportional Random Walk, each node just forwards PDRs to one neighbour and thus only a very small fraction of the graph is involved.

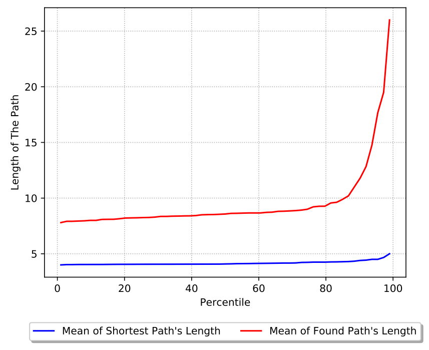


Fig. 4: Comparison between length of the shortest path and length of the path found using Degree-Proportional Random Walk. The x-axis is the percentile and the y-axis is path length.

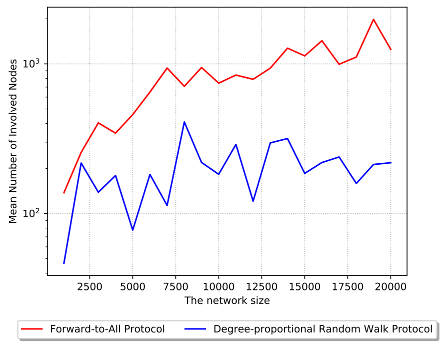


Fig. 5: Comparison of efficiency of Forward-to-All and Degree-Proportional Random Walk on Barabási–Albert graphs. The x-axis is the number of vertices in the graph. The y-axis is the mean number of involved nodes (logarithmic scale).

However, there is no a guarantee that the algorithm finds the shortest path and, as we show in Section 6.1, the paths are twice as long on average. We note that one can trade off between optimality and efficiency/privacy by running a protocol that is somewhere in between these two extreme protocols. For example, nodes can send PDRs to an $\alpha$ fraction of their neighbours (rounded up to the nearest whole neighbour) based on their degrees, where $\alpha \in (0, 1]$. By tweaking this parameter $\alpha$, one moves across the range of efficiency/privacy and optimality in our family of protocols.

## 8 Analysis

Consider a Barabási–Albert graph $G$ generated by $BA(n, 1)$. We select two vertices randomly to be the sender and receiver. Consider the truncated Degree-Proportional Random Walk protocol that stops immediately when any node receives messages from both sender and receiver. Theorem 1 (proof in Appendix B) shows an upper bound of $O(\sqrt{n} \cdot \frac{\log^2 n}{\log \log n})$ on the expected number of nodes involved in the truncated Degree-Proportional Random Walk protocol.

**Theorem 1.** *The expected number of involved nodes in the truncated Degree-Proportional Random Walk protocol on $G$ is $O(\sqrt{n} \cdot \frac{\log^2 n}{\log \log n})$.*

## 9 Related Work

Existing work on route discovery in PCNs can be broadly classified into two categories: solutions which focus on efficiency, and solutions which focus on privacy.

*Efficiency-oriented approaches* Flare [22] and SilentWhispers [15] route payments through highly connected nodes to improve the scalability of route discovery. SpeedyMurmurs [25] and VOUTE [24] employ a similar routing technique called prefix embeddings, which makes the process even faster. These solutions require nodes to have global knowledge of the network (or at least knowledge of these highly connected nodes), whereas nodes in our protocols only need local knowledge of their neighbours and degrees. Spider Network [26] splits payments into smaller units and routes them over multiple paths using waterfilling. However, this does not guarantee the discovery of an optimal path, whereas our protocol guarantees optimality by adding a fee-proportional delay in the path discovery process. Flash [30] uses a modified max-flow algorithm to find the optimal path, but requires nodes to have global knowledge of the network. Perun [13] avoids routing through intermediaries altogether by introducing the notion of virtual channels. However, this does not solve the route discovery problem.

*Privacy-oriented approaches* MAPPCN [29] focuses on anonymity and privacy during transaction execution, but does not address the issue of route discovery as users are required to know the payment path. LightPIR [20] uses private information retrieval to perform private route discovery efficiently, but does not account for optimality of the route in the case of private channels. In contrast, our protocols employ local probing, thus our solutions are still optimal even with private channels.

*Topology hiding computation* The notion of privacy in our work is heavily inspired by topology hiding MPC [18,3]. These techniques are inefficient as the whole network is involved in the route discovery process. Our protocols, in contrast, involve a much smaller fraction of the network at the cost of less privacy.

## 10 Conclusion

In this work, we study the route discovery problem for private PCNs. We formalise an ideal notion of privacy and then show the ideal notion of privacy is feasible but inefficient. We then present a family of practical route discovery protocols which trade off between optimality and efficiency/privacy. We validate our approach on both the Lightning Network and Barabási–Albert graphs. Finally, we prove an upper bound on the number of involved nodes for the Degree-Proportional Random Walk protocol on Barabási–Albert graphs that is sublinear in the number of vertices.

## References

1. Software for complex networks. https://networkx.org/documentation/stable/index.html, accessed: 2020-07-27
2. Raiden network. https://raiden.network/ (2017), accessed: 2021-09-08

3. Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. J. Cryptology pp. 176–227 (2020). https://doi.org/10.1007/s00145-019-09318-y
4. Avarikioti, Z., Kogias, E.K., Wattenhofer, R., Zindros, D.: Brick: Asynchronous incentive-compatible payment channels. In: FC (2021), https://fc21.ifca.ai/papers/168.pdf
5. Barabási, A.L., Albert, R., Jeong, H.: Scale-free characteristics of random networks: the topology of the world-wide web. Physica A: statistical mechanics and its applications pp. 69–77 (2000)
6. Barabási, A.L., Pósfai, M.: Network science. Cambridge University Press, Cambridge (2016), http://barabasi.com/networksciencebook/
7. Bitcoin community: Bitcoin core 0.21.0-based taproot client 0.1. https://bitcointaproot.cc/ (2021)
8. Decker, C.: Lightning network research; topology, datasets. https://github.com/lnresearch/topology, accessed: 2020-10-01
9. Decker, C., Russell, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. https://blockstream.com/eltoo.pdf (2018)
10. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Stabilization, Safety, and Security of Distributed Systems. pp. 3–18. Springer (2015)
11. Dong, M., Liang, Q., Li, X., Liu, J.: Celer network: Bring internet scale to every blockchain (2018)
12. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 625–656. Springer (2019)
13. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: IEEE Symposium on Security and Privacy. pp. 327–344 (2017)
14. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008). https://doi.org/https://doi.org/10.1016/j.dam.2007.12.010
15. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in decentralized credit networks. In: NDSS (2017)
16. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019). https://doi.org/10.1007/s10623-019-00608-x
17. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: International Conference on Financial Cryptography and Data Security. pp. 508–526 (2019)
18. Moran, T., Orlov, I., Richelson, S.: Topology-hiding computation. In: Theory of Cryptography Conference. pp. 159–181. Springer (2015)
19. Nisslmueller, U., Foerster, K., Schmid, S., Decker, C.: Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. In: ICISSP. pp. 7–14 (2020). https://doi.org/10.5220/0009429200070014
20. Pietrzak, K., Salem, I., Schmid, S., Yeo, M.: Lightpir: Privacy-preserving route discovery for payment channel networks. In: Proc. IFIP Networking (2021)
21. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
22. Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network. White Paper (2016)

23. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 347–356. IEEE (2019)
24. Roos, S., Beck, M., Strufe, T.: Voute-virtual overlays using tree embeddings. arXiv: 1601.06119 (2016), http://arxiv.org/abs/1601.06119
25. Roos, S., Moreno-Sanchez, P., Kate, A., Goldberg, I.: Settling payments fast and private: Efficient decentralized routing for path-based transactions. arXiv: 1709.05748 (2017), https://arxiv.org/abs/1709.05748
26. Sivaraman, V., Venkatakrishnan, S.B., Alizadeh, M., Fanti, G., Viswanath, P.: Routing cryptocurrency with the spider network. In: Proc. 17th ACM Workshop on Hot Topics in Networks. pp. 29–35 (2018)
27. Spilman, J.: Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html, accessed: 2020-11-22
28. Tochner, S., Zohar, A., Schmid, S.: Route hijacking and dos in off-chain networks. In: Proc. ACM Conference on Advances in Financial Technologies (AFT) (2020)
29. Tripathy, S., Mohanty, S.K.: Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In: International Conference on Financial Cryptography and Data Security. pp. 481–495. Springer (2020)
30. Wang, P., Xu, H., Jin, X., Wang, T.: Flash: Efficient dynamic routing for offchain networks. In: International Conference on Emerging Networking Experiments And Technologies. p. 370–381 (2019). https://doi.org/10.1145/3359989.3365411

## A Barabási–Albert Network Creation Algorithm

The Barabási–Albert algorithm to create a graph with $n$ vertices and preferential attachment parameter 1 has the following steps:

1. We add the first node.
2. Each new node is connected to 1 existing nodes with a probability proportional to the degree that the existing nodes has. Formally the probability of connecting to node $i$ is :
$$\frac{k_i}{\sum_{j=1}^{n} k_j}$$
where $k_i$ is the degree of node $i$.

Note that the second step is done sequentially for the $n-1$ nodes.

## B Proof of Theorem 1

*Proof.* Let $G$ be the graph formed by the $BA(n,1)$ algorithm. Label the vertices in $G$ by the order in which they join the graph according to the $BA(n,1)$ algorithm [6]. Thus, the initial vertex is labelled $v_1$, the second vertex $v_2$, and the last vertex $v_N$. We show that if two random nodes on the network, start sending two matching packets based on Degree-Proportional Random Walk protocol, after $\mathcal{O}(\sqrt{n} \cdot \frac{\log^2 n}{\log \log n})$ steps the probability that $v_1$ haven't received both packets is bounded.

From [6], for $j > i$, the degree of a vertex $v_i$ at time $j$ is $\sqrt{\frac{j}{i}}$. At any time step $i$, the sum of degree of all the vertices in the graph is $2i$ as there are $i$ edges at time $i$. Thus,

$$\mathbb{P}[v_i \text{ connects to } v_1] = \frac{\sqrt{\frac{i}{1}}}{2i} = \frac{1}{2\sqrt{i}}$$

Now we will compute the probability of any vertex $v_i$ passing a message to $v_1$ given that there is an edge between $v_i$ and $v_1$. The degree of $v_1$ at the end of the graph creation process is $\sqrt{\frac{n}{1}} = \sqrt{n}$. If $v_i$ is connected to $v_1$ then since $m = 1$ we know that other neighbours of $v_i$ were added to the graph after $v_i$, i.e.

$$\text{If } v_j \text{ is } v_i\text{'s neighbour and } \ j \neq 1 \Rightarrow j > i \Rightarrow \sqrt{\frac{n}{j}} < \sqrt{\frac{n}{i}}$$

Thus,

$$\sum_{v_j \in N(v_i)} \deg(v_j) < \sqrt{n} + \sqrt{\frac{n}{i}} \cdot \sqrt{\frac{n}{i}}$$

Since in Degree-Proportional Random Walk, nodes choose one of their neighbours proportional to the degree:

$$\mathbb{P}[v_i \text{ passes a message to } v_1 \mid v_i \text{ is connected to } v_1] \geq \frac{\sqrt{n}}{\sqrt{n} + \sqrt{\frac{n}{i}}^2} = \frac{i}{i + \sqrt{n}}$$

So if $v_i$ is a random node, the probability that following the Degree-Proportional Random Walk $v_i$ forwards the message to $v_1$ is:

$$\mathbb{P}[v_i \text{ passes a message to } v_1] =$$

$$\mathbb{P}[v_i \text{ passes a message to } v_1 \mid v_i \text{ is connected to } v_1] \cdot \mathbb{P}[v_i \text{ is connected to } v_1] \geq$$

$$\frac{i}{i + \sqrt{n}} \cdot \frac{1}{2\sqrt{i}} > \frac{\sqrt{i}}{2(i + \sqrt{n})} > \frac{1}{4\sqrt{n}}$$

We know that the diameter of $BA(n, 1)$ is $\frac{\log n}{\log \log n}$ and after visiting $n^x$ nodes, the number of edges incident to them is below $n^{\frac{1+x}{2}}$. That means that in $\frac{\log n}{\log \log n}$ steps we visit a not visited vertex with probability at least $\frac{n - n^{\frac{1+x}{2}}}{n}$.

Imagine that $v_i$ initiates a Degree-Proportional Random Walk. After visiting $t$ different vertices the probability that the message has not met $v_1$ is:

$$\mathbb{P}[v_1 \text{ did not get the message after s steps}] < (1 - \frac{1}{4\sqrt{n}})^t$$

Now consider the bidirectional Degree-Proportional Random Walk that we described in section 4, if both the sender and the receiver send the message, the probability that after visiting $t$ different vertices, at least one of their message has not reached to $v_1$ we say that the protocol is not successful. Using union bound we say:

$$\mathbb{P}[\text{no success after } t \text{ vertices}] < 2 \cdot (1 - \frac{1}{4\sqrt{n}})^t$$

If we put $t = 4\sqrt{n}\log n$, the probability of not being successful when $n$ is large (using $1 - x \leq e^{-x}$) is :

$$2 \cdot (1 - \frac{1}{4\sqrt{n}})^{4\sqrt{n}\log n} < \frac{2}{n}$$

And to visit $t = 4\sqrt{n}\log n$ unvisited nodes, we need $4\sqrt{n}\frac{\log^2 n}{\log\log n}$ steps.

## C Forward-to-All Protocol Algorithms

**Protocol 1:** On receiving a new $\mathtt{PDR}_s$(Exploration Phase)

```
   /* Here we consider node $u_i$ on receiving a $\mathtt{PDR}_s$ from $u_{i-1}$         */
   **Input:** $\{\mathtt{PDR}_s\}$
1  **Goal** Forwarding the $\mathtt{PDR}_s$ or finding a matched $\mathtt{PDR}_r$ for it.
2  $u_i$ Parses $\mathtt{PDR}_s$ as $\mathtt{PDR}_s = (\mathsf{Identifier}, amount, \mathsf{Sender})$
3  $u_i$ searches for a path discovery data ($\mathtt{PDD}_r$) with the same Identifier. If
    he found he skips the next steps and does the protocol 3 otherwise he
    does the next steps.
4  $\forall\ v \in \{neighbours[u_i] \mid c\,(u_i, v) \geq amount\ ,\ v\ \neq u_{i-1}\}$ $u_i$ waits $d_{u_i,v}$
    and then sends $(\mathsf{Identifier}, amount - f_{u_i,v}(amount), \mathsf{Sender})$ to $v$.
5  $u_i$ saves the following path discovery data $\mathtt{PDD}_s$ on storage :
    $\mathtt{PDD}_s \coloneqq \{\mathtt{PDR} : (\mathsf{Identifier}, amount, \mathsf{Sender})\,,$ **previousNode** :
    $u_{i-1},$ **nextNode** : $-\}$
```

**Protocol 2:** On receiving a new $\texttt{PDR}_r$(Exploration Phase)

```
/* Here we consider node $u_i$ on receiving a $\texttt{PDR}_r$ from $u_{i+1}$ */
Input: $\{\texttt{PDR}_r\}$
1 Goal Forwarding the $\texttt{PDR}_r$ or finding a matched $\texttt{PDR}_s$ for it.
2 $u_i$ Parses $\texttt{PDR}_r$ as $\texttt{PDR}_r = (\mathsf{Identifier}, amount, \mathsf{Receiver})$
3 $u_i$ searches for a $\texttt{PDD}_s$ with the same $\mathsf{Identifier}$. If he found he skips the
  next steps and does the protocol 3 otherwise he does the next steps.
4 $\forall\, v \in \{neighbours[u_i] \mid c\,(v, u_i) \geq amount\ ,\ v\ \neq u_{i+1}\}$ $u_i$ waits $d_{u_i,u_{i+1}}$
  and sends $\big(\mathsf{Identifier}, amount + f_{u_i,u_{i+1}}(amount), \mathsf{Receiver}\big)$ to $v$.
5 $u_i$ saves the following path discovery data $\texttt{PDD}_r$ on storage :
  $\texttt{PDD}_r := \{\texttt{PDR} : (\mathsf{Identifier}, amount, \mathsf{Receiver})\,, \textbf{previousNode} :$
  $-, \textbf{nextNode} : u_{i+1}\}$
```

**Protocol 3:** On finding a matching $\texttt{PDR}_s$ and $\texttt{PDR}_r$(Exploration Phase)

```
/* Here we assume that $u_i$ has received a new $\texttt{PDR}_r$ from $u_{i+1}$ that
   has the same identifier as a $\texttt{PDD}_s$ that $u_i$ has stored before. In
   the case that $u_i$ receives $\texttt{PDR}_s$ the same protocol should be run
   with very minor differences. */
Input: $\{\texttt{PDR}_r, \texttt{PDD}_s, \varDelta\}$
/* the $\varDelta$ can be calculated using the timing of $\texttt{PDR}_s$ and $\texttt{PDR}_r$ by $u_i$
   */
1 Goal Matching the path discovery requests sent from the sender and
  the receiver.
2 $u_i$ parses $\texttt{PDR}_r$ and $\texttt{PDD}_s$ as follow: $\texttt{PDD}_s = \{\texttt{PDR} :$
  $(\mathsf{Identifier}, amount_s, \mathsf{Sender})\,, \textbf{previousNode} : u_{i-1}, \textbf{nextNode} : -\}$
  and $\texttt{PDR}_r = (\mathsf{Identifier}, amount_r, \mathsf{Receiver})$
3 Step 2: $u_i$ waits $\varDelta$
4 Step 3: $u_i$ sends a
  $\texttt{PFM}_s = (\mathsf{Identifier}, amount_s, \mathsf{Sender}, \mathsf{Path\_found\_message})$ to $u_{i-1}$ and a
  $\texttt{PFM}_r = (\mathsf{Identifier}, amount_r, \mathsf{Sender}, \mathsf{Path\_found\_message})$ to $u_{i+1}$
5 Step 4: $u_i$ completes the nextNode item of the $\texttt{PDD}_s$ with $u_{i+1}$.
```

**Protocol 4:** On receiving a new $\texttt{PFM}_s$(Notification Phase)

```
/* Here we consider node with $u_i$ on receiving a $\texttt{PFM}_s$ from $u_{i+1}$ */
Input: $\{\texttt{PFM}_s\}$
1 Goal Notifying the sender and the receiver that a path exists.
2 $u_i$ Parses $\texttt{PFM}_s$ as
    $\texttt{PFM}_s = (\mathsf{Identifier}, amount_s, \mathsf{Sender}, \mathsf{Path\_found\_message})$
3 $u_i$ retrieves the corresponding $\texttt{PDR}_s$ as $\texttt{PDR}_s = \{\texttt{PDR} : (\mathsf{Identifier}, amount_s, \mathsf{Sender}), \textbf{previousNode} : u_{i-1}, \textbf{nextNode} : -\}$
    and completes the nextNode item with $u_{i+1}$
4 $u_i$ forwards the $\texttt{PFM}_s$ to $u_{i-1}$
```