Route Discovery in Private Payment Channel Networks

No Author Given

No Institute Given

Abstract. In this work, we are the first to explore route discovery in private payment 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 significantly smaller fraction of the network. We additionally propose a more realistic notion of privacy in order to measure the privacy leakage of our protocols in practice. Our realistic notion of privacy challenges an adversary that join the network with a fixed budget to create channels to guess the sender and receiver of a transaction upon receiving messages from our protocols. Simulations of our protocols on the Lightning network topology (for random transactions and uniform fees) show that 1) forming edges with high degree nodes is a more effective attack strategy for the adversary, 2) there is a tradeoff between the number of nodes involved in our protocols (privacy) and the optimality of the discovered path, and 3) our protocols involve a very small fraction of the network on average.

Keywords: Payment Channel Networks · Privacy · Bitcoin · Route Discovery · Lightning Network

1 Introduction

Payment channel networks (PCNs) is one of the most promising approaches to scale cryptocurrencies like Bitcoin [16]. PCNs allow any pair of users to set up

a payment channel with each other, thereby enabling an unlimited number of costless off-chain transactions between them. Users who are not directly connected 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 [24,11,18,10,2,5]; the most widely used being the Bitcoin Lightning Network [18].

To route a transaction in a PCN from sender to receiver, a two-step process is necessary (but sometimes executed in parallel): (a) finding the optimal route or *route discovery*, which typically means finding the shortest or cheapest path from sender to receiver, and (b) executing the transaction payment in an atomic fashion, i.e., the transaction is either executed or aborted in all the channels of the path, so no party can lose money. The route discovery problem focuses on the first step, and thus the goal is to find the optimal route from sender to receiver in an efficient and privacy-preserving manner, while the atomic execution of transactions is not considered. This is a well-researched problem with several existing solutions [19,13,22,23,26,17]. However all these solutions assume that the *entire topology of the PCN is known* by at least one party (for instance the users who download the entire network [27] or trampoline nodes [19]).

Recently, however, the Bitcoin protocol upgraded to taproot [8] that uses Schnorr signatures [14] to aggregate public keys and signatures, making a transaction involving multiple users indistinguishable from a transaction involving just two users on the blockchain. As a result, the users of the Lightning Network can now form *private channels*, leading eventually to a private payment channel network with unknown topology. This in turn affects heavily the route discovery process as all existing algorithms utilize the known PCN topology. Hence, *designing route discovery algorithms suitable for private PCNs is paramount.*

Our Contribution. In this work, we consider for the first time the problem of route discovery in private PCNs where the capacities, fees, and even the mere *existence* of channels can be (partially) unknown. In particular, 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 (i.e., their balances, fees, and local neighborhood).

Our objective is to construct protocols that are *efficient, private, and optimal* in terms of minimal fees. We identify several key challenges: (a) we observe that in this setting, the only strategy a sender and receiver can follow to find an (optimal) route is to send exploratory messages through the network. However, this may incur high communication overhead. (b) To find the optimal path with certainty, all nodes should be involved in the route discovery process, again leading the high communication overhead. (c) To achieve ideal privacy, nodes that end up in a payment path should 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. Thus, no information can be leaked on the PCNs topology but even

simple topology hiding broadcast protocols are highly inefficient [15,3]. (d) If ideal privacy is out of reach, there is no practical notion of privacy for PCNs to measure the possible privacy leakage.

We explain how we address these key challenges below. At its core, we exploit a trade-off between the number of involved nodes (which defines efficiency and affects privacy) and how cheap the discovered route is (optimality). To practically measure the privacy leakage, we present a novel game for route discovery protocols over arbitrary networks. In detail, our contributions are:

- (Ideal and Practical Notion) We put forward a security notion for private route discovery in Section 3 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, confirming challenge (c)). To account for the inefficiency of our ideal security notion, we also define a practical notion of privacy in Section 3.3 inspired by metrics used in information retrieval (addressing challenge (d)). We empirically analyse our protocols with respect to our practical privacy notion.
- (Practical Protocols) We present a family of route discovery protocols on private PCNs in Section 4 that are much more efficient, involving a small fraction of the network (addressing challenges (a) and (b)). 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 each edge has a delay that is proportional to its fee. 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 in Section 5. Our simulation show that Forward-to-All typically involves around 3% of nodes in Lightning, while Degree-Proportional Random Walk only involves around 0.1%, and the discovered paths are around twice as long as the optimal ones.
- (Analysis) We also prove some analytical bounds in Section 5 for our algorithms on particular classes of graphs.

Transaction execution. Transaction execution is the counterpart problem to route discovery for successfully executing mulit-hop payments in a PCN, as discussed previously. Although we do not consider this problem it is worth noting that current solutions for this problem inevitably leak some information during the setup phase of the execution. In particular, the common practice for executing atomically multi-hop payments is to use Hash Timelock Contracts

(HTLCs) [18] which use decreasing timelocks depending on the exact position of the node in the path, therefore leaking their position in the path. MAPPCN [27] preserves the anonymity of the sender and receiver from the users on the payment path but the sender must know the nodes in the path. Alternative methods, e.g., Blitz [4], reveal instead the identities of all nodes in the selected path. Designing a truly privacy-preserving protocol for this task is thus an interesting and challenging research direction, albeit not the study of this work.

In the following, we use the terms user, node, and party interchangeably.

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 $\sum_{i=1}^{\ell} f_{u_{i-1},u_i}(x + \phi_{i-1})$, where ϕ_i, $i = 0, \ldots, \ell - 1$, is the aggregated fees that nodes $u_{i+1}, u_{i+2}, \ldots, u_{\ell-1}$ charge. Since the receiver u_ℓ 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.

Network model. We assume a synchronous network model, i.e., there exists some known finite time bound Δ and the adversary cannot delay delivery of any message sent by honest parties for a larger than Δ time period. We further assume all users only have local knowledge of the topology of the payment channel network with the addition of an estimate of the degree of their neighbours, i.e., each node u only knows their set of immediate neighbours and an estimate of the degree of each neighbour in that set.

3 Ideal and Practical Privacy Notions

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

Ideal privacy for PCNs 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, assuming that users know at the very least the current balances on their own channels (as in the Lightning 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.

We only consider path finding protocols Π, which always find the cheapest path. Defining ideal privacy for protocols that do not necessarily output the cheapest path is more complex because the privacy loss depends on the discovered path. 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 (e.g., by providing wrong or inconsistent input).

Our privacy notion is inspired by the Indistinguishability under Chosen Topology Attack (IND-CTA) security definition from work on topology-hiding multiparty computation [15], and it is defined as follows: 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 between both networks 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.

3.1 The Ideal Privacy Security Game

We consider a security game involving an adversary $\mathcal{A}$ against a path-finding protocol Π. 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 $u \notin \mathsf{shortestPath}_{G^b}(u_s^b, u_r^b, x)$ 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 $f_{u,v}^i(.)$.
  2. A sender and receiver pair (u_s^i, u_r^i) and amount x.

  $\mathcal{A}$ chooses 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 Π 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 Π ϵ-private if $\mathcal{A}$ wins the above game with probability at most $1/2 + \epsilon$, and private if it is ϵ secure for some negligible ϵ.

3.2 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), the design of a private path-finding protocol is not challenging. In particular, each party in the network can 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 party.

To design a protocol without a trusted third party, we can instantiate $\mathcal{T}$ using a multi-party computation (MPC) protocol. As here the users need to share pairwise channels, they need to know about the other users, 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, where the goal is to hide the topology of the communication channels. Instead, 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 when $V^1 \neq V^2$ as nodes would only talk to their neighbours, and for the corrupted nodes, these are identical.

3.3 A Practical Notion of Privacy

Here we outline the following game as well as some metrics to measure the privacy of any route discovery protocol over an arbitrary network. The game is played with an adversary $\mathcal{A}$ over an arbitrary network $G = (V, E)$. The adversary $\mathcal{A}$ is given some budget $B \in \mathbb{N}$ and $\mathcal{A}$ can corrupt (as defined in Section 3.1) any number of nodes in the network such that the total number of edges incident to these nodes is no greater than B. Here we emphasise two pertinent aspects of the corruption process: first, when $\mathcal{A}$ corrupts a node, *all* of the node's channels must be added to the total count, and second, we do not double count channels that have already been accounted for (which is the case where $\mathcal{A}$ corrupts two neighbouring nodes). The constraint on the number of edges the adversary can create captures the notion that creating new nodes (i.e., wallets) in a PCN is cheap, however, creating new edges (i.e., channels) comes with some fixed cost.

We denote by Π a route discovery protocol run by a pair of honest nodes (a source and a sink) over the graph G which may contain outputs (henceforth called messages) for both honest and adversarial nodes in the process of running the protocol. The goal of the adversary is to correctly identify the source or sink of Π upon receiving a set of messages from Π.

Estimators. Let $\mathcal{M}$ denote the space of all messages adversarial nodes may receive in process of honest nodes running Π. An *estimator* is simply a function g that takes as input a set of messages M such that each message in the set is from $\mathcal{M}$ and outputs a pair of nodes in V. That is, given a set of messages from Π, the adversary outputs a guess of the nodes that are the source and sink of the route finding protocol Π.

Privacy metrics. We adopt a similar approach as in [28] and use *recall* to measure the privacy our of route discovery protocols. Recall is a common performance metric used in information retrieval to evaluate estimators in classification settings. Let $M_{(u,v)}$ denote the set of messages the adversary receives that originates from a pair of honest nodes $u, v \in V$ running an instance of Π. The recall of an estimator g is defined as the number of classifications for the pair (u, v) where either u is classified as the source or v is classified as the sink over the total number of instances where Π is run between u as source and v as sink. Formally,

$$Rec_{g,(u,v)} = \mathbb{1}\{g(M_{(u,v)}) = (u, \cdot) \vee g(M_{(u,v)}) = (\cdot, v)\} \tag{1}$$

We note that it is common to average as well as macro-average the recall over all honest nodes to get the macro-averaged expected recall $\mathbb{E}[Rec_g] = \frac{1}{|V \times V|} \sum_{(u,v) \in V \times V} \mathbb{E}[Rec_{g,(u,v)}]$.

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. Next, 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) 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 reason 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. This further helps to improve correctness and even privacy.

Our protocols differ only on the proportion of nodes each node forwards the message to. On one extreme, we have Forward-to-All that involves nodes sending exploratory messages to all their neighbours, where each message is delayed for some time proportional to the fees. As a result, the optimal path is always discovered, and moreover, 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 one neighbour. In this case, we expect fewer nodes to be involved in the discovery process but the optimal path may not be discovered.

As the protocols in the family are similar, we first present a generic overview of Forward-to-All, and then briefly describe how to modify it to get Degree-Proportional Random Walk. We then suggest some improvements to boost the privacy of this family of protocols. We present a more detailed description of our protocols in Appendix A.

4.1 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 adding their desired fees for routing the transaction, and only forward the message after some time period has elapsed that is proportional to their desired 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, δ, 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.

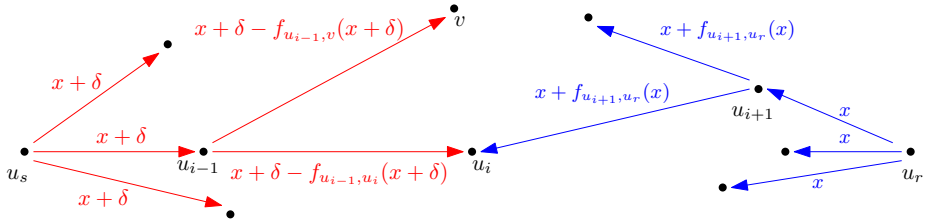


Fig. 1: Propagating exploratory messages from sender and receiver in the Forward-to-All protocol. Each directed edge (u, v) is labelled with the transaction amount in the message that u sends to v.

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 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.

4.2 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 both sender and 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 identifier and amount x_s (resp. x_r to u_{i+1}). Both u_{i-1} and u_{i+1} then identify 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.

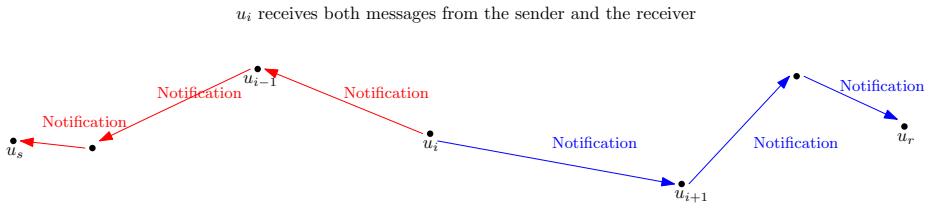


Fig. 2: Sending informative messages back to sender and receiver.

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 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.

4.3 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, stop the message propagation. 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.

4.4 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 Forward-to-All.

4.5 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 recognise when 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, however, 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 unlinkability by using a bilinear map [12] $e : G_1 \times G_2 \to G_T$ (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 $(g_2^{r'/x}, g_2^{r'})$.

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. We stress that the unlinkeability is limited as it only holds if the adversary has access to messages originating either only at the sender or only at the receiver. This is inherent as we need parties who receive tuples (a, b) and (a', b') originating at both to efficiently recognize a path is found 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 such as additive ElGamal encryption. If several protocols are running simultaneously on the network, nodes will see many messages with the same quantised transaction amount, effectively reducing linkeability.

5 Analysis and Evaluation

In this section, we empirically study the *efficiency, optimality* and *privacy* of our family of protocols described in Section 4 on a recent snapshot (September 2021) of the Lightning network [9], which comprises of 13780 nodes and

63518 channels. We parameterise our family of protocols by $\beta \in (0, 1]$ which represents the proportion (rounded up to the nearest integer) of neighbouring nodes in the Degree-Proportional Random Walk protocol that any node chooses to pass messages to. We note that $\beta = 1$ corresponds to Forward-to-All. In our experiments using the Lightning network, we chose 1000 random pairs of sender and receivers and ran our protocols for selected values of β with these random pairs. Our choices of β can be seen in Table 1 which also presents a summary of our results. We include an extension of our empirical and theoretical analysis to Barabasi-Albert random graphs to analyse the scalability of our protocols to larger networks in Appendix C.

5.1 Efficiency

We measure the efficiency of our protocols by the communication overhead. This is measured by the average number of involved nodes in one route discovery attempt, i.e., the number of nodes that receive at least one message in any given run of our protocol. As we can see from the second column of Table 1, the expected number of involved nodes in a single route discovery attempt ranges from 16 (0.1% of the network) to 459 (3% of the network) with the smallest value corresponding to sending messages to just 1 neighbour and the largest value corresponding to Forward-to-All.

5.2 Optimality

The fee-proportional delay function that we described in the Notification Phase of Section 4 guarantees that the first message that reaches the sender and the receiver corresponds to the shortest path in Forward-to-All. There is no such guarantee, however, for our protocols in the case where $\beta < 1$ due to its probabilistic nature. To measure the optimality of the discovered path for our protocols when $\beta < 1$, we look at the average ratio of the discovered path length over the length of the shortest path for various choices of β in the 1000 runs as described above. In the third column of Table 1, we observe that protocols with $\beta < 1$ return longer paths on average compared to Forward-to-All, as expected. However, even for small values of β, e.g., $\beta = 0.01$, the average ratio is no more than 2.

5.3 Privacy

We empirically measure the privacy of our protocols using the notion of recall as defined in Section 3.3. We stress that there are two integral components of our notion of recall. The first is the corruption strategy of the adversary, which is used by the adversary to choose nodes up to the corruption budget B to corrupt. The second is the choice of estimator the adversary uses to guess the source and sink of an instance of the route discovery protocol. We will first describe 3 corruption strategies, and then define the first spy estimator. We further show that the first spy estimator is in general a good guessing heuristic for the adversary.

β	# involved nodes	$\frac{len(\text{found path})}{len(\text{shortest path})}$	Recall
only 1 neighbour	16	3.36	0.129
1%	34	1.54	0.129
10%	86	1.14	0.191
20%	133	1.10	0.249
40%	277	1.07	0.29
80%	431	1.01	0.371
100%	459	1	0.43

Table 1: Summary of the efficiency, optimality and privacy of our protocols. We chose a budget of 0.15 (i.e., the adversary can corrupt nodes such that up to 15% of the total edges are incident to corrupted nodes) in our estimation of recall.

Corruption strategies. The first strategy, called *random corruption*, involves choosing nodes at random in the network to corrupt (and all their edges) until the total number of corrupted edges is less the adversarial budget B. In the second strategy, termed *well-connected corruption*, we first sort the nodes in the network based on their degree and then sequentially corrupt nodes in descending order to their degree until the budget is depleted. The intuition behind this strategy is that high-degree nodes tend to serve as hubs that route a large proportion of the transactions in the Lightning Network [9,1]. However, this strategy assumes the adversary has *full* knowledge of the degree of all nodes in the network, which is not the case in private PCNs. To achieve similar results with only local knowledge of the graph, we introduce our third strategy, called *random hub corruption*. In this strategy, the adversary starts with a random node, but instead of corrupting the node, it randomly corrupts one of its neighbours. This process continues until the budget is depleted. Intuitively, the majority of nodes in a PCN are users that are connected to high-degree hub nodes. As such, there is a high chance that a random neighbour of a selected node is a hub node. An algorithmic description of all 3 corruption strategies is given in Appendix B.1.

First spy estimator. The choice of estimator in our experiments is the *first-spy estimator* [28], which is simply guessing the first honest node that passes a message from the protocol to any adversarial node as the source. We justify our choice of estimator with the following lemma (proof in Appendix B.2) which shows that the first spy estimator is optimal in a restricted setting where messages are not re-randomised, sender and receivers are randomly chosen, and timing assumptions are not taken into account.

Lemma 1. *For Degree-Proportional Random Walk (without re-randomisation), when sender-receiver pairs are chosen uniformly and independently from honest nodes and both propagate their messages independently and randomly, the first spy estimator is the Maximum a Posteriori (MAP) estimator.*

We stress that our theoretical results hold in a restricted setting which does not mirror the realistic setting which we run our experiments on. Indeed in Appendix B.4, we highlight some limitations and counterexamples to the first spy estimator under more realistic assumptions. Nevertheless, we believe it is a good first step guessing heuristic and we leave developing stronger theoretical results in the realistic setting as an interesting direction of future work.

Average recall. Figure 3 in Appendix B.3 presents the average recall for each of the corruption strategies given the first spy estimator over 1000 random runs of each protocol. We note that the well-connected and random hub corruption strategies perform strictly better in terms of recall for all β values compared to random corruption. Moreover, random hub corruption performs almost equally well compared to well-connected corruption and thus is a good choice to use in practice when there is only partial information known about node degrees. Finally, our plots show that the adversary achieves the highest average recall with Forward-to-All and the lowest with Degree-Proportional Random Walk.

Optimality and efficiency/privacy trade-off. In Forward-to-All, nodes forward messages to all their neighbours, and as a result a large fraction of the network is involved in the route-discovery process, which as we see in Section 5.1 and Section 5.3 has negative impact on the efficiency and privacy of the protocol. 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 messages to one neighbour and thus only a very small fraction of the graph is involved. However, as we see from Section 5.1 the paths are longer on average. We note that β values in between these extremes allow users to trade off between optimality and efficiency/privacy.

6 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.

On the efficiency front, Flare [19] and SilentWhispers [13] route payments through highly connected nodes to improve the scalability of route discovery. SpeedyMurmurs [22] and VOUTE [21] 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, whereas we present a protocol that does not require any knowledge of the PCN topology. Spider Network [23] 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 route discovery process. Flash [29] uses a modified max-flow algorithm to find the optimal path, but also requires nodes to have global knowledge of the network. Perun [2] avoids routing through intermediaries altogether by introducing the notion of virtual channels. However, this does not solve the route discovery problem.

On the privacy front, MAPPCN [27] 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 [17] uses private information retrieval to perform private route discovery efficiently, but does not account for optimality in the case of private channels. In contrast, our protocols employ local probing, thus our solutions are still optimal even with private channels. Recently, [25] uses MPC to perform privacy preserving routing of transactions, however it is only limited to fixed star graph topologies.

7 Conclusion

We presented the first route discovery protocols that are suitable for private PCNs. We first formalized the ideal notion of privacy in PCNs, and showed that ideal privacy is feasible yet inefficient. We then presented a family of practical route discovery protocols which trade off between optimality and efficiency/privacy. To evaluate their privacy leakage, we introduced and leveraged a novel practical notion of privacy.

The simulation of our protocols on the Lighting Network and Barabási–Albert graphs validates our approach, unveiling the aforementioned trade-off. We also observe that our protocols involve a very small fraction of the network on average, showcasing we can indeed design efficient and private routing algorithms that rely on minimal to no assumptions on the topology of the PCN with almost optimal results. From our simulations on Lightning we further deduce that an effective strategy for an adversary is to connect with high-degree nodes, i.e., payment hubs. We also discover through our empirical simulations on large Barabási–Albert graphs that the efficiency and privacy of our algorithms perform much better on average compared to our theoretical upper bound, which demonstrates that our algorithms also scale efficiently with the size of PCNs.

References

1. Lightning network search and analysis engine. `1ml.com`
2. Perun: Virtual payment hubs over cryptocurrencies. In: IEEE S&P (2017)
3. Akavia, A., LaVigne, R., Moran, T.: Topology-hiding computation on all graphs. J. Cryptology pp. 176–227 (2020)
4. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: Secure Multi-Hop payments without Two-Phase commits. USENIX Association (Aug 2021)
5. Avarikioti, Z., Kogias, E.K., Wattenhofer, R., Zindros, D.: Brick: Asynchronous incentive-compatible payment channels. In: FC (2021)
6. 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)
7. Barabási, A.L., Pósfai, M.: Network science. Cambridge University Press, Cambridge (2016), `http://barabasi.com/networksciencebook/`
8. Bitcoin community: Bitcoin core 0.21.0-based taproot client 0.1. `https://bitcointaproot.cc/` (2021)

9. Decker, C.: Lightning network research; topology, datasets. `https://github.com/lnresearch/topology`, accessed: 2022-10-01
10. Decker, C., Russell, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. `https://blockstream.com/eltoo.pdf` (2018)
11. 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)
12. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008)
13. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in decentralized credit networks. In: NDSS (2017)
14. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019)
15. Moran, T., Orlov, I., Richelson, S.: Topology-hiding computation. In: Theory of Cryptography Conference. pp. 159–181. Springer (2015)
16. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
17. Pietrzak, K., Salem, I., Schmid, S., Yeo, M.: Lightpir: Privacy-preserving route discovery for payment channel networks. In: Proc. IFIP Networking (2021)
18. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
19. Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network. White Paper (2016)
20. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In: 2019 IEEE EuroS&P Workshop. IEEE (2019)
21. Roos, S., Beck, M., Strufe, T.: Voute-virtual overlays using tree embeddings. arXiv: 1601.06119 (2016)
22. 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)
23. 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)
24. Spilman, J.: Anti dos for tx replacement. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html`, accessed: 2020-11-22
25. Tiwari, S., Yeo, M., Avarikioti, Z., Salem, I., Pietrzak, K., Schmid, S.: Wiser: Increasing throughput in payment channel networks with transaction aggregation. CoRR **abs/2205.11597** (2022)
26. Tochner, S., Zohar, A., Schmid, S.: Route hijacking and dos in off-chain networks. In: AFT (2020)
27. Tripathy, S., Mohanty, S.K.: Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In: FC. pp. 481–495. Springer (2020)
28. Venkatakrishnan, S.B., Fanti, G., Viswanath, P.: Dandelion: Redesigning the bitcoin network for anonymity. Proc. ACM Meas. Anal. Comput. Syst. **1**(1) (2017)
29. 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)

A Protocol Details

Here, we describe our protocol Forward-to-All in detail. We leave out the details of Degree-Proportional Random Walk as it is analogous. We denote by u_s the sender of a message and u_r the receiver. Intermediate nodes are denoted by u_i. 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

$$\mathtt{PDR}_s \coloneqq (\textbf{identifier} : I, \textbf{amount} : x + \delta, \textbf{tag} : \mathsf{Sender})$$

and the PDR that u_r sends is

$$\mathtt{PDR}_r \coloneqq (\textbf{identifier} : I, \textbf{amount} : x, \textbf{tag} : \mathsf{Receiver})$$

.

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

/* Here node u_i receives a $\mathtt{PDR}_s$ from u_{i-1} */

1 *Inputs:* $\{\mathtt{PDR}_s\}$

Goal: Forwarding the $\mathtt{PDR}_s$ or finding a matching $\mathtt{PDR}_r$ for it.

The protocol:

1. u_i parses $\mathtt{PDR}_s$ as $\mathtt{PDR}_s = (\mathsf{Identifier}, amount, \mathsf{Sender})$
2. u_i searches for a path discovery data ($\mathtt{PDD}_r$) with the same Identifier. If they find it they skip the next steps and execute Protocol 3, otherwise they proceed to the next steps.
3. $\forall\ v \in \{neighbours[u_i] \mid c\,(u_i, v) \geq amount\ ,\ v\ \neq u_{i-1}\}$ u_i waits $d_{u_i,v}$ and then sends $(\mathsf{Identifier}, amount - f_{u_i,v}(amount), \mathsf{Sender})$ to v
4. u_i saves the following path discovery data $\mathtt{PDD}_s$ on storage : $\mathtt{PDD}_s \coloneqq \{\mathtt{PDR} : (\mathsf{Identifier}, amount, \mathsf{Sender})\,, \textbf{previousNode} : u_{i-1}, \textbf{nextNode} : -\}$

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

/* Here node u_i receives a $\texttt{PDR}_r$ from u_{i+1} */

2 *Inputs:* $\{\texttt{PDR}_r\}$

Goal: Forwarding the $\texttt{PDR}_r$ or finding a matching $\texttt{PDR}_s$ for it.

The protocol:

1. u_i parses $\texttt{PDR}_r$ as $\texttt{PDR}_r = (\mathsf{Identifier}, amount, \mathsf{Receiver})$
2. u_i searches for a $\texttt{PDD}_s$ with the same Identifier. If they find it they skip the next steps and executes Protocol 3, otherwise they proceed to the next steps.
3. $\forall\, v \in \{neighbours[u_i] \mid c\,(v, u_i) \geq amount\ ,\ v\ \neq u_{i+1}\}$ 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.
4. u_i saves the following path discovery data $\texttt{PDD}_r$ on storage : $\texttt{PDD}_r \coloneqq \{\texttt{PDR} : (\mathsf{Identifier}, amount, \mathsf{Receiver})\,, \mathbf{previousNode} : -, \mathbf{nextNode} : u_{i+1}\}$

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 only forward the message with the lowest fee (in an ideal network with honest parties this will never happen as the cheaper message will arrive first, however, in practice this could happen due to network delay).

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 Δ. The delay time Δ 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 $\Delta = \frac{d_{u_i,u_{i+1}} - t}{2}$.

After the delay, u_i creates and sends a path found message ($\texttt{PFM}$) to these nodes. Specifically, u_i sends $\texttt{PFM}_s = (\mathsf{Identifier}, amount_s, \mathsf{Sender}, \mathsf{Path\_found\_message})$ to u_{i-1} and $\texttt{PFM}_r = (\mathsf{Identifier}, amount_r, \mathsf{Sender}, \mathsf{Path\_found\_message})$ 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.

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

`/* Here` u_i `receives` $\texttt{PDR}_r$ `from` u_{i+1} `that has the same identifier as a` $\texttt{PDD}_s$ `that` u_i `has stored before.` `*/`

3 *Inputs:* $\{\texttt{PDR}_r, \texttt{PDD}_s, \Delta\}$

Goal: Matching the path discovery requests sent from the sender and the receiver.

The protocol:

1. u_i parses $\texttt{PDR}_r$ and $\texttt{PDD}_s$ as follows: $\texttt{PDD}_s = \{\texttt{PDR} : (\mathsf{Identifier}, amount_s, \mathsf{Sender}), \textbf{previousNode} : u_{i-1}, \textbf{nextNode} : -\}$ and $\texttt{PDR}_r = (\mathsf{Identifier}, amount_r, \mathsf{Receiver})$
2. u_i waits Δ `//` Δ `can be calculated using the timing of` $\texttt{PDR}_s$ `and` $\texttt{PDR}_r$ `by` u_i
3. u_i sends $\texttt{PFM}_s = (\mathsf{Identifier}, amount_s, \mathsf{Sender}, \mathsf{Path\_found})$ to u_{i-1} and $\texttt{PFM}_r = (\mathsf{Identifier}, amount_r, \mathsf{Sender}, \mathsf{Path\_found})$ to u_{i+1}
4. u_i completes the **nextNode** item of the $\texttt{PDD}_s$ with u_{i+1}.

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 $\texttt{SR} \coloneqq (\textbf{identifier} : I, \textbf{tag} : \mathsf{Stopping\_Request})$. 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.

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

`/* Here node` u_i `receives a` $\texttt{PFM}_s$ `from` u_{i+1} `*/`

4 *Inputs:* $\{\texttt{PFM}_s\}$

Goal: Notifying the sender and the receiver that a path exists.

The protocol:

1. u_i Parses $\texttt{PFM}_s$ as $\texttt{PFM}_s = (\mathsf{Identifier}, amount_s, \mathsf{Sender}, \mathsf{Path\_found})$
2. 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}
3. u_i forwards the $\texttt{PFM}_s$ to u_{i-1}

B Empirical Privacy Estimation Details

B.1 Corruption strategies

Here, we provide the algorithmic description of the corruption strategies described in Section 5.3.

Algorithm 1: Random corruption

Input: $\{B, G(V, E)\}$
$Rem\_B \leftarrow B \cdot |E|$
Label all the vertices in V with Honest
while $Rem\_B > 0$ **do**
  Choose a node v randomly from V
  Label v with Adversary
  Retrieve the set of v's neighbours i.e., $Ne(v)$
  for v' *in* $Ne(v)$ **do**
    if v' *is* Honest **then**
      $Rem\_B = Rem\_B - 1$
  Remove v from V

Algorithm 2: Well-connected corruption

Input: $\{B, G(V, E)\}$
$Rem\_B \leftarrow B \cdot |E|$
Label all the vertices in V with Honest
Sort the set of vertices based on their degree and call the sorted list $\bar{V}$
$indx = 0$
while $Rem\_B > 0$ **do**
  $v = \bar{V}[indx]$
  Label v with Adversary
  Retrieve the set of v's neighbours i.e., $Ne(v)$
  for v' *in* $Ne(v)$ **do**
    if v' *is* Honest **then**
      $Rem\_B = Rem\_B - 1$
  $indx = indx + 1$

Algorithm 3: Random hub corruption

Input: $\{B, G(V, E)\}$
$Rem\_B \leftarrow B \cdot |E|$
Label all the vertices in V with Honest
while $Rem\_B > 0$ **do**
  Choose a node v randomly from the set of vertices V
  Retrieve the set of v's neighbours i.e., $Ne(v)$
  Choose a node h randomly from $Ne(v)$
  Label h with Adversary
  Retrieve the set of h's neighbours i.e., $Ne(h)$
  for v' *in* $Ne(h)$ **do**
    if v' *is* Honest **then**
      $Rem\_B = Rem\_B - 1$
  Remove h from V

B.2 Proof of Lemma 1

Proof. First, let's focus only on the messages propagated by the sender. During the propagation phase, adversarial nodes might receive the same message several times from different nodes and in different timestamps. We define a random variable S as the sender of the message which takes value from the set of honest nodes V_H. Moreover, we define a set of random variables $\{I_i\}_{i\geq 1}$ where I_i is the ith honest node that forwards the message to one of the adversarial nodes, I_i takes value from the set of honest nodes, V_H, too.

First, we argue that in the Degree-Proportional Random Walk, $S - I_1 - \{I_i\}_{i\geq 2}$ form a Markov chain. Since each node only forwards the message to one of its neighbors, given I_1, we can find the probability of the message following different routes without any dependency on S therefore we can find the probability distribution of $\{I\}_{i\geq 2}$ for any given graph too. In simpler words, we can think of I_1 as an artificial sender which starts the propagation phase. Therefore the guessing strategy of the adversarial nodes will be a function of I_1 instead of $\{I_i\}_{i\geq 1}$ i.e.,

Now we argue why the guessing strategy should output I_1 to maximize the likelihood function. In the Degree-Proportional Random Walk algorithm, each arbitrary node u upon receiving a message will forward it to one of its neighbors based on a degree proportional distribution. Consider a fixed graph $G(V, E)$. Let's define $p_{u,v}$, $\forall (u, v) \in E$ as the probability that u upon receiving a message will choose v amongst its neighbors to forward the message to it. Let's denote the realization of I_1 by i_1, we write the Bayesian rule:

$$P[S = v | I_1 = i_1] \propto P[I_1 = i_1 | S = v] \cdot P[S = v] \tag{2}$$

Since the sender is chosen uniformly, $P[S = v] = \frac{1}{|V_H|}$, $\forall v \in V_H$, therefore the second term in the equation 2 is the same for all of the honest nodes and we only need to show that $v = i_1$ maximizes the first term.

Let's call the adversarial node that i_1 has sent the message to a_1, then we will have $P[I_1 = i_1|S = i_1] = p_{i_1,a_1}$. For the sake of simplicity, we assume that the probability that i_1 is the sender but does not forward the message to a_1 in the first round is negligible compared to p_{i_1,a_1}, the proof will work with minor changes if this assumption is not true. Now we find $P[I_1 = i_1|S = v]$ for an arbitrary v.

The message is originated from v and has passed a set of nodes $j_1, j_2, \ldots, j_k,\ j_k = i_1$ after it has reached a_1. Therefore $P[I_1 = i_1|S = v] = p_{v,j_1} \cdot \prod_{\ell=1}^{k-1} p_{j_\ell,j_{\ell+1}} \cdot p_{i_1,a_1} < p_{i_1,a_1}$ hence i_1 maximizes the equation 2.

So we showed that for a single source of message, the first spy estimator is the MAP estimator. For showing the same result in the case that both sender and receiver propagate messages, we note that if the sender and receiver start propagating messages in independent times unknown to the adversary, timing does not add any information to the adversary, and the adversary can treat the messages received from the sender and receiver as independent messages and guess the sender in an independent process from the receiver and using the first spy estimator. □

B.3 Relevant Figures

B.4 Limitations of the first spy estimator

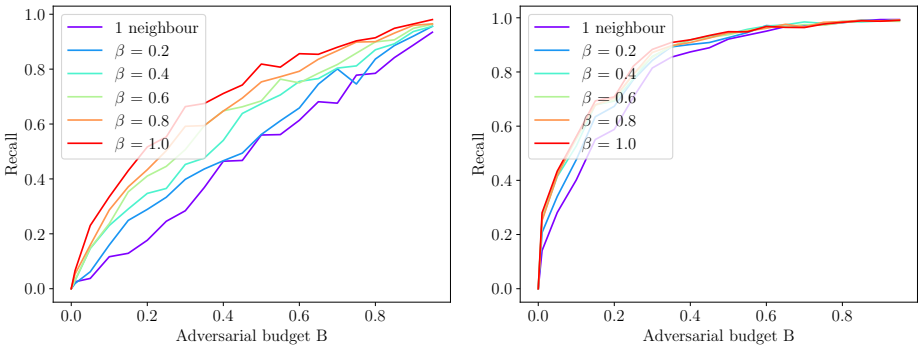
We present an example where the first spy estimator might not be optimal when we take into account timing assumptions. Consider the network in Figure 4, and assume discrete time slots whereby messages are propagated one edge at a time. That is, if we have time slots $t_1, t_2, \ldots$ a message travelling over a path (x, y, z) sent at t_1 would be sent over (x, y) at t_1 and (y, z) at t_2. In this setting, if the adversary knows that the sender (s) and receiver (r), propagate their messages at the same time, a_2 will receive the receiver message 1 time slot earlier than the slot which a_1 receives their message. a_1 can use this information to infer that node i is not the real sender and the real sender is at least in distance 2 away from a_1.

C Scalability of our protocols

We are also interested to see how our protocols perform as the Lightning Network scales in size. To do so, we perform simulations of our protocols on Barabási–Albert graphs. The Barabási–Albert model [7] is a popular algorithm to create scale free networks using a preferential attachment mechanism (see Appendix C.1 for more details). Many real world networks, including the Lightning network, are characterized as scale-free [6,20].

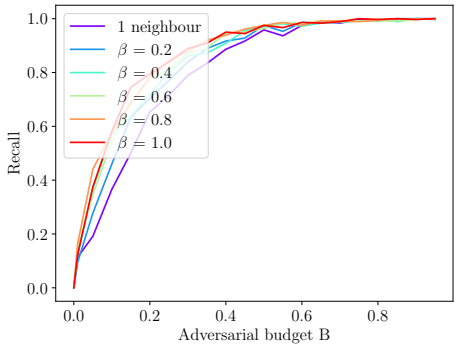
C.1 Barabási–Albert Network Creation Algorithm

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



(a) Random corruption

(b) Well-connected corruption



(c) Random hub corruption

Fig. 3: Average recall over 1000 random runs of the protocols in the Lightning network for different corruption strategies

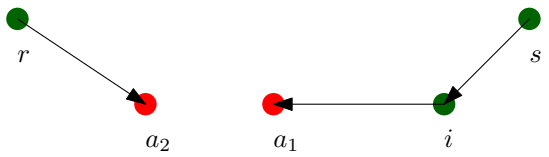


Fig. 4: Counterexample to optimality of first spy estimator that accounts for timing assumptions. Green nodes are honest and red nodes are adversarial.

1. We add the first node.
2. Each new node is connected to 1 existing node with a probability proportional to the degree that the existing node 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.

The Barabási–Albert algorithm $BA(n, m)$ to create a graph with n vertices and preferential attachment parameter m is similar but we start with m_0 connected nodes at the first step, and at each step each new node connects to $m < m_0$ nodes with probability proportional to the existing node degree.

C.2 Empirical and theoretical analysis

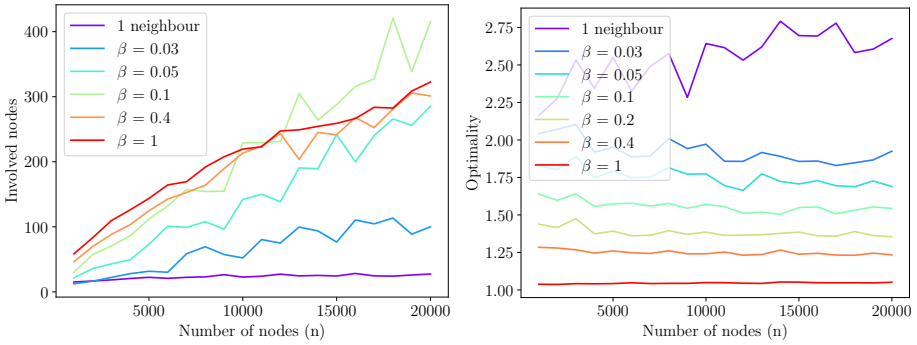
Figure 5 presents the average communication overhead as well as the average ratio of the length of the found path over the optimal path. Our empirical simulations show that the communication overhead for the Barabási–Albert graph with 20000 nodes is similarly low (2%) when compared to the Lightning Network.

We complement our empirical analysis with a theorem which states that for Barabási–Albert graphs the communication overhead of the Degree-Proportional Random Walk protocol scales sublinearly in the number of nodes n in the network.

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

Proof. We analyse the case where G is the graph formed by the $BA(n, 1)$ algorithm. For graphs formed by $BA(n, m), m > 1$ the proof is analogous.

Label the vertices in G by the order in which they join the graph according to the $BA(n, 1)$ algorithm [7]. 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



(a) Communication overhead

(b) Ratio of length of found path to length of optimal path

Fig. 5: Efficiency and optimality of our routing algorithms on Barabasi-Albert graphs

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 [7], 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,

$$\mathbb{P}[v_i \text{ passes message to } v_1] = \frac{i}{i + \sqrt{n}} \cdot \frac{1}{2\sqrt{i}} > \frac{\sqrt{i}}{2(i + \sqrt{n})} > \frac{1}{4\sqrt{n}}$$

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. □