An Empirical Analysis of Privacy in the Lightning Network

George Kappos [*1], Haaroon Yousaf [*1], Ania M. Piotrowska[1,2], Sanket Kanjalkar[3], Sergi Delgado-Segura[4], Andrew Miller[3,5], and Sarah Meiklejohn[1]

[1]*University College London*
[2]*Nym Technologies*
[3]*University of Illinois Urbana-Champaign*
[4]*PISA Research*
[5]*IC3*

Abstract

Payment channel networks, and the Lightning Network in particular, seem to offer a solution to the lack of scalability and privacy offered by Bitcoin and other blockchain-based cryptocurrencies. Previous research has already focused on the scalability, availability, and crypto-economics of the Lightning Network, but relatively little attention has been paid to exploring the level of privacy it achieves in practice. This paper presents a thorough analysis of the privacy offered by the Lightning Network. We present three main attacks that exploit publicly available information about the network topology and its active nodes and channels in order to learn information that is designed to be kept secret, such as how many coins a node has available to spend or who the sender and recipient are in a payment routed through the network. We evaluate one of our attacks on the live network and, due to cost and ethical considerations, evaluate our other two attacks on a simulated Lightning network that faithfully mimics the real one.

1 Introduction

Since its introduction in 2008, Bitcoin [31] has become the most widely adopted cryptocurrency, with it and the thousands of alternative cryptocurrencies it has inspired being touted as an alternative to the traditional financial ecosystem. The decentralized and permissionless nature of Bitcoin allows users to easily join the network and avoids the need for intermediaries and authorities who control the flow of money between them. Instead, the validity of each transaction is verified by a consensus decision made by the network participants themselves; valid transactions are then recorded in the public blockchain. The blockchain thus acts as a ledger containing all transactions that have ever taken place.

The need to broadcast transactions to all peers in the network and store them in a permanent ledger presents two problems for the longevity of blockchain-based cryptocurrencies. First, it imposes severe scalability limitations: the Bitcoin blockchain today is over 283 GB, and Bitcoin can achieve a throughput of only ten transactions per second. Other cryptocurrencies achieve somewhat higher throughputs, but there is an inherent tradeoff in these broadcast-based systems between throughput and security [11, 14]. Second, the transparent nature of the ledger means anyone can observe the flow of coins, identify the counterparties to a transaction, and link different transactions. This has been shown most decisively for Bitcoin [4,28,35,38,40], but this type of analysis extends even to cryptocurrencies that were explicitly designed with privacy in mind, such as Dash [27, 29], Monero [17, 23, 30, 45], and Zcash [6, 19, 34], or even across cryptocurrency ledgers [44].

The most promising solutions that have been deployed today to address the issue of scalability are so-called "layer-two" protocols [15], with the Lightning Network (LN) [33] emerging as the most popular one since its launch in March 2018. In Lightning, pairs of participants use the Bitcoin blockchain to open and close *payment channels* between themselves. Within a channel, these two users can make arbitrarily many *off-chain* payments between themselves, without having to use the blockchain in any way. Beyond a single channel, Lightning supports multi-hop payment routing, meaning even participants who are not connected directly can still route payments through a broader *payment channel network* (PCN). Nodes in the network are incentivized to help route payments by a fee they can charge for each payment they forward. Altogether, the use of channels avoids the latency and cost (in terms of transactions fees) associated with getting a transaction confirmed on the Bitcoin blockchain.

In addition to the promise it shows in providing better scalability, Lightning also seems to address the issue of privacy. As we elaborate on in Section 2, the nodes in the network and most of the channels in the network are publicly known in order to build up the PCN (although some channels may be kept private), as is the *capacity* of a given channel, meaning the maximum payment value that can be routed through it. On the other hand, the individual *balances* associated with the channel, meaning the actual amount that each participant can pay the other, are kept secret. Furthermore, payments

[*]Both authors contributed equally

are not broadcast to all peers and are not stored in a public ledger; indeed, if a payment is done within a single channel then only its counterparties need ever know it exists. Even if a payment passes through multiple channels, onion routing is used to ensure that each node on the path can identify only its immediate predecessor and successor.

As is the case with ledger-based cryptocurrencies, however, the gap in Lightning between the potential for privacy and the reality is significant, as we show in this work. While some of the works that de-anonymize Bitcoin and other cryptocurrencies do so using network data (e.g., IP addresses) [7, 8, 22], most of them rely instead on attacks that can be conducted offline and in a passive manner using the blockchain data. There is no ledger here, so we instead must actively monitor and participate in the network. We describe in Section 3 how we run our own Lightning node, as well as the other data sources we collect. While running a node is itself inexpensive, some of the attacks we develop are too expensive or time-consuming for us to carry out on a relatively modest budget. To nevertheless evaluate these attacks, we developed a simulator for the Lightning network. In comparison to other recent LN simulators [5, 9, 10, 13, 46], we believe ours to be the most realistic in terms of its assumptions and the distribution of payments it produces, as we argue in Section 4.

Concretely, we consider the following three privacy properties that LN promises [2, 24]:

Third-party balance secrecy says that although the capacity of the channel is public, the respective balances of the participants should remain secret. We explore this property in Section 5 by presenting and evaluating two methods by which an active attacker (i.e., one opening channels with nodes in the network) can discover channel balances.

Off-path payment privacy says that any node not involved in routing a payment should not infer any information regarding the routing nodes or the payment value. We explore this property in Section 6 by presenting and evaluating a method by which an active attacker can use the ability to discover balances to form *network snapshots*. By comparing consecutive network snapshots, the attacker can infer payments by identifying where and by how much the balances of channels changed.

On-path relationship anonymity says that intermediate nodes routing the payment should not learn which other nodes, besides their immediate predecessor or successor, are part of the payment's route. They also should not learn the length of the route or their position within it, or link this payment to previous payments involving the same nodes. We explore this property in Section 7, where we leverage our LN simulator (described in Section 4) to evaluate the ability of an intermediate node to infer the sender and recipient in payments that it routes.

1.1 Ethical considerations

The attacks presented in Sections 6 and 7 are evaluated on a simulated network rather than the live one, so do not raise any ethical concerns. We evaluate our attacks in Section 5, however, on the live test network and on parts of the main network. We made every effort to ensure that our attacks did not interfere with the normal functioning of the network: the messages sent during the attack have no abnormal effect and do not cost any money to process, and their volume is relatively modest (we sent at most 24 messages per node we attacked). We thus believe that they did not have any long- or short-term destructive effect on the nodes that processed them. We will also disclose the results of this paper to the Lightning developers at the time of its submission.

1.2 Related work

We consider as related all research that focuses on the Lightning Network, particularly as it relates to privacy. Most of the previous research has focused on the scalability, utility and crypto-economic aspects of LN [9, 20, 21, 24, 42], or on its graph properties [26, 39]. Rohrer et al. [37] study the susceptibility of LN to topology-based attacks, and Tochner et al. [41] present a DoS attack that exploits how multi-hop payments are routed. Among other findings, they show that the ten most central nodes can disrupt roughly 80% of all paths using their attack. Nowatowski and Tøn. [32] study various heuristics in order to identify Lightning transactions from the underlying Bitcoin blockchain.

In terms of privacy, Malavolta et al. [25] identify a new attack exploiting the locking mechanism, which allows dishonest users to steal payment fees from honest intermediaries along the path. They propose anonymous multi-hop locks as a more secure option. Herrera-Joancomartí et al. [16] investigate a balance discovery attack, which we re-implement in this work as our "oracle-aided" attack, and carry it out on part of the test network. The main limitation of this attack is its reliance on the specifics of the error messages it receives. We overcome this limitation by presenting a new generic attack (in Section 5), as well as investigating the implications of such an attack more broadly (in Section 6).

Béres et al. [5] look briefly at the question of finding the sender and recipient of a payment. Similarly to our work, they develop an LN traffic simulator based on publicly available network snapshots and information published by certain node owners. They still make several simplifying assumptions, however, such as assuming all payments carry the same value and all nodes run the same software. We thus believe our simulator more faithfully mimics the actual behavior in LN, as we argue further in Section 4. Furthermore, their work examines the question of anonymity (finding the sender and recipient) but considers only single hop payments and does not look at other privacy properties.

There are a number of other Lightning network studies that use a network simulator [9, 10, 13, 46]. Several of these simulators were used to perform economic analysis of the Lightning network [9,13,46], while the CLoTH simulator [10] provides only performance statistics (e.g., time to complete a payment, probability of payment failure, etc.). However, all of those simulators make several simplifying assumptions about the topology, path selection algorithm, and distribution of payments. As such, they are not suitable for an analysis of its privacy properties.

2 Background

At the most fundamental level, the Lightning Network (LN) allows its participants to pay each other. Unlike Bitcoin and other cryptocurrencies, the key idea is that individual transactions do not have to be published on a blockchain.

In order to open a *channel*, two parties, Alice and Bob, deposit a certain amount of bitcoins into a 2-of-2 *multi-signature* address, meaning any transaction spending these coins would need to be signed by both of them. These committed funds represent the channel *capacity*; i.e., the maximum amount of coins that can be transferred via this channel. The payment channel is associated with *inward* and *outward balances*, representing how the funds are distributed between Alice and Bob. This information is captured in the channel's *state*.

Once a channel is established, Alice and Bob announce it to the rest of the Lightning Network. They can also start exchanging arbitrarily many payments, as long as either party has a positive balance. In order to settle a payment, Alice and Bob update the balances and sign a new *commitment transaction* representing these balances. These transactions alter the channel state but are not broadcast to the main ledger; for this reason, they are said to happen *off-chain*. Either of them can close the channel at any time by broadcasting the latest mutually signed commitment transaction. This means only two transactions are ever submitted to the blockchain: one when the channel is open, and one when it is closed.

This process describes how two users can pay each other if they have an open channel between them. What is more likely, however, is that Alice and Bob are not connected directly, so instead need to *route* the payment through the global Lightning Network. This can be represented as an undirected multigraph $G = (V, E)$, where nodes represent users and edges represent channels. Nodes are publicly associated with some uid and a public key $\mathsf{pk}_{\mathsf{uid}}$. Edges are publicly associated with a *channel identifier* cid, the overall channel capacity C, and a *fee* fee that is charged for routing payments via this channel. Privately, edges are also implicitly associated with the inward and outward balances of the channel. The topology of this network, along with all public labels, is known to every peer. There exist also *private* channels that are not included in this public topology and revealed only at the time of routing; we discuss these further in Section 3.2.3.

To perform this *multi-hop routing*, Alice additionally uses *onion routing* to hide her relationship with Bob. This means each intermediate node knows only its immediate predecessor and successor, and none of the nodes can reconstitute the whole route. Alice selects the entire path to the destination (*source routing*), based on the public capacities and fees associated with the channels between her and Bob. Once a path is selected, the eventual goal is that each intermediate node forwards the payment to its successor, expecting that its predecessor will do the same so its balance will not change. The nodes cannot send the money right away, however, because if one channel has sufficient capacity for the payment but not a sufficient outward balance, the payment fails. Equally, the payment could fail due to a malicious intermediate node simply deciding not to forward it.

To thus create an intermediate state, and to unite payments across an entire path of channels, the Lightning Network uses *hashed time-lock contracts* (*HTLCs* for short), which allow for time-bound conditional payments. While it is not necessary in order to understand the rest of the paper, we provide more details about HTLCs (and how channels work) in Appendix A.

To have Alice (sender) pay Bob (recipient) using multi-hop routing, the protocol thus follows five basic steps:

1. Invoicing Bob generates a secret x and computes the hash h of it. He issues an *invoice* containing h and some amount amt that he wants to be paid, and sends it to Alice. The pre-image x will later be used to settle the channels along the route Alice uses.

2. Onion routing Alice picks a path $A \to U_1 \to \cdots \to U_n \to B$, using the publicly available information about the network topology and fees. To route her payment along this path, Alice forms a Sphinx [12] packet destined for Bob and routed via the U_i nodes. Alice then sends the outermost onion packet onion_1 to U_1.

3. Channel preparation Upon receiving onion_i from U_{i-1}, U_i decodes it to reveal: cid, which identifies the next node U_{i+1}, the amount amt_i to send them, a timeout t_i, and the packet onion_{i+1} to forward to U_{i+1}. Before sending onion_{i+1} to U_{i+1}, U_i and U_{i-1} need to *prepare* their channel by updating their intermediate state using an HTLC based on the values h and t_i. The HTLC ensures that if U_{i-1} does not provide U_i with the pre-image of h before the timeout t_i, U_i can claim a refund of their payment. After the state is updated, U_i can send the packet onion_{i+1} to U_{i+1}.

4. Invoice settlement Eventually, Bob receives onion_{n+1} from U_n and decodes it to find (amt, t, h). If amt and h match what he put in his invoice, he sends the invoice pre-image x to U_n in order to redeem his payment of amt. This value is in turn forwarded backwards along the path.

5. Channel settlement At every step on the path, U_i and U_{i+1} use the value x to *settle* their channel, i.e., to confirm

the updated state, reflecting the fact that $\texttt{amt}_i$ was sent from U_i to U_{i+1} and thus that amt was sent from Alice to Bob.

3 Data Collection & Measurements

In this section, we describe our data sources and present some preliminary measurements about the collected data.

3.1 Data collection

Unlike blockchain data, gathering data about the Lightning Network (LN) cannot be done after the fact and instead requires us to be an active network participant. To do this for both the test network (testnet) and main network (mainnet), we ran the lnd client from April 13, 2019 to February 14, 2020 and scraped the results of the `describegraph` command every hour in order to form snapshots of the public network graph.

The gathered snapshots provide us with ground-truth data about the network graph, including information about the nodes (identifiers, network addresses, status, etc.) and their channels (identifiers, capacity, endpoints, etc.). To augment this dataset, we also scraped user-submitted (and validated) data from 1ML, which is an LN search engine.[1] This included data such as node names, client versions, and operating system. We describe this data further in Section 4.

To analyze on-chain transactions, we also ran a full Bitcoin mainnet and testnet node, using the BlockSci tool [18] to parse and analyze the raw blockchain data. As of February 14, 2020 the mainnet dataset was 283 GB, containing 503 million transactions, and the testnet dataset was 33 GB, containing 54 million transactions.

3.2 Measurements

As of February 14, 2020, the LN testnet had 3,003 nodes and 8,612 open channels, with a total capacity of 607.74 BTC. The mainnet had 6,372 nodes and 35,987 open channels, with a total capacity of 880.02 BTC.

3.2.1 Network statistics

Lightning nodes have the option of choosing to broadcast (1) no address or some combination of (2) an IPv4 address, (3) an IPv6 address, and (4) a .onion address. To broadcast no address, the node must keep the IP address field of the configuration file blank (which it is by default). This means that other nodes are unable to initiate a connection with this node, but it is still able to open connections with others.

For each node in the scraped dataset, we recorded the type of address it reported (if any). In testnet we found that 83.78% reported only an IPv4 address, 1.64% only an IPv6 address, 4.42% only a .onion address, 5.17% reported a mixture of addresses, and 4.98% were incorrectly set to broadcast a localhost address. In mainnet we found that: 55.45% reported only an IPv4 address, 0.56% only an IPv6 address, 22.18% only a .onion address, 21.29% reported a mixture of addresses, and 0.51% were incorrectly set. The main difference is that there is a much higher percentage of nodes in mainnet running with only a .onion address, as well as reporting mixture of addresses.

[1] https://1ml.com/

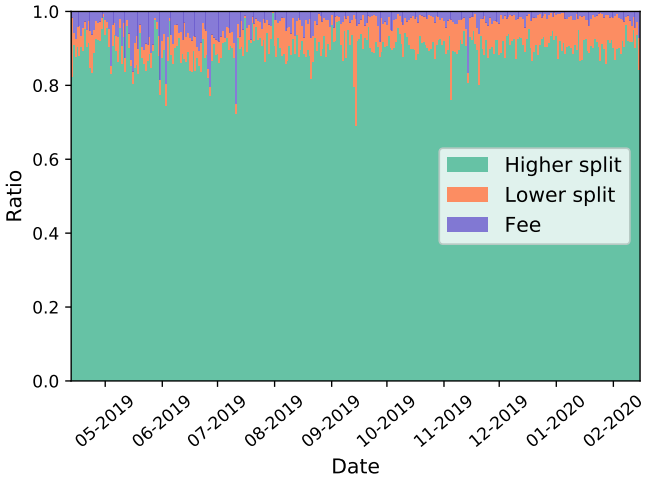


Figure 1: The split of funds in channels closed with two outputs, averaged across a mainnet Bitcoin block.

3.2.2 Blockchain statistics

The data we collected from LN included the hash of the Bitcoin transaction used to open each channel. By combining this with Bitcoin blockchain data, we were thus able to identify when channels closed and how their funds were distributed.

Across our entire collection period, we identified 84,080 channels, of which 47,347 had closed with a total capacity of 1372.47 BTC. For each channel that was closed, we analyzed the scripts used in the Bitcoin transaction, which were all (according to the LN specification) a P2WSH ("pay-to-witness-script-hash") address. 67.20% of channels were claimed by a single output address (i.e., the channel was completely unbalanced at the time of closure), 31.41% by two output addresses, and 1.39% by more than two outputs.

Of those channels claimed by a single output, over 88.23% closed with at least 90% of their capacity. However, 1.90% closed with less than 50% of their original capacity, meaning they paid a significant amount in transaction fees. For those channels claimed by two outputs, Figure 1 shows the split of coins between the two parties, as well as the amount lost to fees. Over 45% of these channels had 90% or more of the capacity claimed by a single output, meaning they were still fairly one-sided.

3.2.3 Private channels

Our attacks are all applicable only to public channels; i.e., channels that have been announced and are included in the public network graph. It is also possible to use private channels, however, and even to route via these private channels by including them in the route_hints field of a payment invoice. We thus seek to determine an upper bound for the number of private channels, in order to understand the scope of our attacks on the public network.

To obtain anecdotal evidence for the existence of private channels, we first gathered 122 payment invoices by interacting with merchants (i.e., making a fake purchase without proceeding with the payment) identified on 1ML and Lightning Network Stores.[2] We then scanned the QR codes contained in these invoices and found that 62 were invoices for unique destination addresses (as merchants often use payment gateway providers), and 15 of these (24%) contained information about private channels. Once we knew that private channels were thus used relatively frequently, we aimed to find them in a systematic way by looking at the opening and closing of channels represented in the Bitcoin blockchain. In this we were guided by previous work on channel identification [32] and by our own ground-truth data.

To align with our network data collection period, we first looked for all Bitcoin transactions that (1) occurred after April 13, 2019, (2) before February 15, 2020, and (3) where one of the outputs was a P2WSH address. We identified 1,219,541 transactions matching this pattern. This was compared with the 44,587 public channels opened during this period, which suggests the need to refine this heuristic. To create these refinements, we used our ground-truth data about the opening transactions of these public channels. From these known opening transactions, we identified features that we then used to develop a heuristic for identifying other potential opening transactions. These additional features were as follows:

- 99.99% of our known opening transactions had at most two outputs, which likely represents the funder of the channel (1) creating the channel and (2) sending themselves change.
- 99.93% had a P2WSH output address that received at most 16,777,215 satoshis. This is the current maximum capacity of a channel in lnd.
- 100% had a P2WSH output that appeared at most once as both an input and output, which reflects its "one-time" usage as a payment channel and not as a reusable script.

By requiring our collected transactions to also have these three features, we were left with 216,651 potential transactions representing the opening of private channels. Again, this suggested the need for further refinement.

If the outputs in these transactions had spent their contents (i.e., the channel had been closed), then we were further able to see how they did so, which would provide better evidence of whether or not they were associated with Lightning. Again, we identified the following features based on known closing transactions we had from our network data.

- 100% had a non-zero sequence number, as required by the Lightning specification [2].
- 100% had an input that was a 2-of-2 multisig address, again as required by the Lightning design (see Appendix A).
- 100% had a single input (which was the 2-of-2 multisig address).
- 98% had at most two outputs, which reflects the two participants in the channel.

By requiring our collected opening transactions to also have a closing transaction with these four features, we were left with 27,313 pairs of transactions that were potentially involved in opening and closing private channels, 37% of the channels we observed. Again, this is just an upper bound; there are other reasons to use 2-of-2 multisigs in this way that have nothing to do with Lightning. Nevertheless, it broadly matches our anecdotal evidence, in which private channels were involved in 24% of the invoices, as well as a recent blog post from the BitMEX research team that estimates 27.8% of all channels are private [36].

4 Lightning Network Simulator

In this section, we describe the LN simulator we developed in order to measure the feasibility of our attack on the on-path relationship anonymity in Section 7. We rely on the simulator to evaluate our attacks because performing them in the live network would require significant resources (e.g., running many nodes), which is financially impractical and raises ethical concerns.

In order to simulate the Lightning Network as closely as possible and minimize the assumptions made about it, we parameterize our simulation according to data we collected from our own interactions with the network or other public sources, as described in Section 3.1. We implemented our simulator in 2,624 lines of Python 3.

4.1 Node and network parameters

Network topology. As mentioned in Section 2, we represent the network as a graph $G = (V, E)$. We obtain the information regarding V and E from the snapshots we collected, as described in Section 3.2, which also include additional information such as capacities and fees. The network topology view of our simulator is thus an exact representation of the actual public network.

Geolocation. Recall from Section 3.2 that nodes may publish an IPv4, IPv6 or .onion address, or some combination of these.

[2] https://lightningnetworkstores.com/

If a node advertised an IPv4 or IPv6 address then we used it to assign this node a corresponding geolocation. This enabled us to accurately simulate TCP delays on the packets routed between nodes, based on their distance and following previous studies [14] in using the global IP latency from Verizon.[3] For nodes that published only a .onion, we assign delays according to the statistics published by Tor metrics, given the higher latency associated with the Tor network.[4]

Path selection. As discussed in Section 2, the route to the destination in LN is constructed solely by the payment sender. All clients generally aim to find the shortest path in the network, meaning the path with the lowest amount of fees. As shown by Tochner et al. [41], however, both the routing algorithm and the fee calculation differ across the three main choices of client software: lnd, c-lightning, and eclair. We could not easily extract or isolate the routing algorithms from these different implementations, so chose to implement all three versions of the path finding algorithm ourselves. We did this using Yen's *k*-shortest path algorithm [43] and the networkx Dijkstra's SPF algorithm.[5]

Our collected snapshots did not include information about software versions, so we scraped the Owner Info field for each node listed on the 1ML website. Although in 91% of the cases this field is empty, the results allow us to at least estimate the distribution of the client software. We obtained information about 370 nodes and found that 292 were lnd, 54 were c-lightning, and 24 were eclair. We randomly assign software versions to the remaining nodes in the network according to this distribution, and then modify the weight function in the path finding algorithm according to the software version.

Channel balances. In order to simulate realistic payments, it is important to know the individual balances associated with a channel, and not just its capacity. To do this, we leverage the results of the balance discovery attack presented in Section 5. As we will see there, we learn not only the concrete balances for a number of channels, but also identify patterns in terms of how the balances tend to be split. We thus assign channel balances according to the ground truth data and heuristics described in detail in Section 5.

4.2 Payment parameters

In this section, we discuss the parameters that we use to simulate payments; e.g., the frequency at which payments are sent or received.

Payment distribution. Different nodes will have different volumes of sent and received payments according to their usage of the network. To model this distribution of payments as accurately as possible, we distinguish between three different categories: Merchant, Router and User. We denote as S_x and R_x the probability of a payment being respectively sent or received by a node belonging to category x for $x \in \{m, r, u\}$.

The intuition behind these parameters is that users buy goods or services, meaning they initiate more payments than merchants or routers so S_u is higher than S_r or S_m. Similarly, merchants receive more payments than users or routers, so R_m is higher than R_u or R_r. We also use small S_r, R_r since the majority of a router's activity involves forwarding payments on behalf of other nodes, rather than sending or receiving payments itself. Within each category, we further group nodes as having *high*, *medium* or *low* connectivity according to the number of channels they have. This follows the assumption that people who have more money are willing to transact more regularly. We flag a node as having high connectivity if it belongs in the top 25th percentile of its category, low if it belongs in the bottom 25th percentile, and medium otherwise.

In order to identify Merchants we first scraped the Directory tab from 1ML, which gave us 398 public keys of nodes labelled by 1ML as providing various services. We augmented this with another publicly available data source, the Lightning Networks Store[6], which contains a list of the online merchants that support Lightning payments. In total, we collected the public keys of 525 Merchant nodes. We found no similar data sources concerning Router nodes, but intuitively this type of node routes many payments due to its strategic connectivity, and thus gains profit by earning fees. Hence, we focus on the profit-driven agenda of these nodes, which requires them to be online and available at all times in order to maximize their profit. Moreover, a node with more connectivity and large capacity is more likely to be a reliable router. Therefore, from the set of remanining nodes we collect the top 20% of nodes with the highest connectivity and capacity, and have at least 90% uptime.[7] Overall, we flag 50 nodes as Routers. Our classification results align with previous studies [39, 41] which study nodes with the highest betweenness centrality. We then labelled all the remaining 4044 nodes as Users.

Payment volume. In order to estimate the total number of payments t_pay happening in the Lightning Network per day, we use as a starting point the estimates published by LNBIG [3], the largest node that holds more than 40% of the network's total capacity at the time of writing. According to LNBIG, the total number of routed transactions going through the network is 1000-1500 per day. However, this estimation does not capture the payments performed via direct channels. Since it is not possible to know the exact number of payments happening via direct channels, we estimate the percentage of such payments using our simulator. We simulate 10,000 payments for a span of a day[8] for 10 different choices

[3] https://enterprise.verizon.com/terms/latency/

[4] https://metrics.torproject.org/onionperf-latencies.html

[5] https://networkx.github.io/documentation/stable/reference/algorithms/shortest_paths.html

[6] https://lightningnetworkstores.com/

[7] A node might go offline for a short period of time due to a software upgrade.

[8] This is far more than the actual number of LN payments per day, but such a large number allows us to minimize the randomness of the results.

of S_x, R_x. We observed that the number of direct payments between users, compared to the total number of payments, fluctuated from 6% to 14% with an average of 9%. Given this result, our estimation for the total payments per day is $t_{pay} = 1250 + 1250 \cdot 9\% = 1362$. However, since this cannot be taken as ground truth knowledge, for the rest of the paper we continue treating t_{pay} as a parameter and try different values for it, while of course keeping our choices realistic compared to this estimation.

Payment value. For payments with a merchant as a recipient, we chose the value randomly based on the prices we gathered from scraping products from the websites of 89 different merchants.

More generally, we assume that nodes with more open channels and higher capacities make bigger payments, as indicated by the observation of the values our node was asked to forward. We thus built a *value predictor* using the following heuristic. We first compute for every node the average balance across its channels (B_{avg}) and the maximum balance across its channels (B_{max}) in order to know the average and maximum amount the node can forward. We further define the minimum payment a node can forward (B_{min}) as the minimum HTLC it has across all its channels. Since LN users do not tend to spend all of their balance at once, as that would require rebalancing or opening a new channel (both of which are expensive operations), we also define a parameter $V_x \in [0,1]$ that represents the percentage of a node's balance that it is willing to spend on any given payment. The value of each payment is then chosen randomly from $[B_{min}, B_{max}]$ following the Poisson distribution with a mean value of $B_{avg} \cdot V_x$.[9]

4.3 Simulation results

In this section, we demonstrate the effect that different choices of t_{pay}, V_x, S_x, and R_x have on the behavior of the simulated network. In particular, we focus on three main features of the simulation that determine the effectiveness of our attack described in Section 7. All the instances we run simulate the Lightning Network for the duration of a day, using the data collected on December 19, 2019.

Node throughput. We first define the throughput T_x as the number of payments a node x forwards per day. We use fixed values for the parameters S_x and R_x since our goal is to measure how different values of t_{pay} affect T_x.

In particular, we group nodes into four categories, according to the number of channels they possess (1-150, 150-300, 300-500, 500+). We next measure the average number of payments each of these categories forwards per day, given different values of t_{pay}. These values range from a realistic estimate of the volume of payments today (as discussed earlier) to a dramatic increase that may reflect a broader adoption of LN in the future. The results are in Figure 2a. We observe that the number of channels a node has influences the number of payments it forwards, as might perhaps be expected. In particular, for $t_{pay} = 2000$ we see that a node with over 500 channels forwards around 276 payments per day, whereas a node with 150-300 channels forwards slightly above 130. When $t_{pay} = 4500$, the node with the maximum number of channels forwarded 1,485 payments, which translates to a throughput of 0.0172 payments per second.

Paths tried per payment. We next focus on the number of paths each initiated payment attempts in order to successfully reach its destination. Again, we fix the parameters S_x and R_x; we also fix t_{pay}, as we argue that increasing the number of payments will inherently increase the number of failed paths, so this does not reveal any insight into the network.

In particular, we group the payment attempts into four categories representing the number of paths attempted (1, 2, 3, and 4+). We then run the simulation for three different values of the parameter V_x (which determines the value of payments): 0.005, 0.2, and 0.5. The results are in Figure 2b. The main insight here is that payments with higher values have an increased probability of failing due to insufficient balances of the nodes along the path. For example, when $V_x = 0.05$ a payment has a 78.85% chance of succeeding on the first attempt, whereas when $V_x = 0.5$ only 42.18% of the payments succeed on the first attempt.

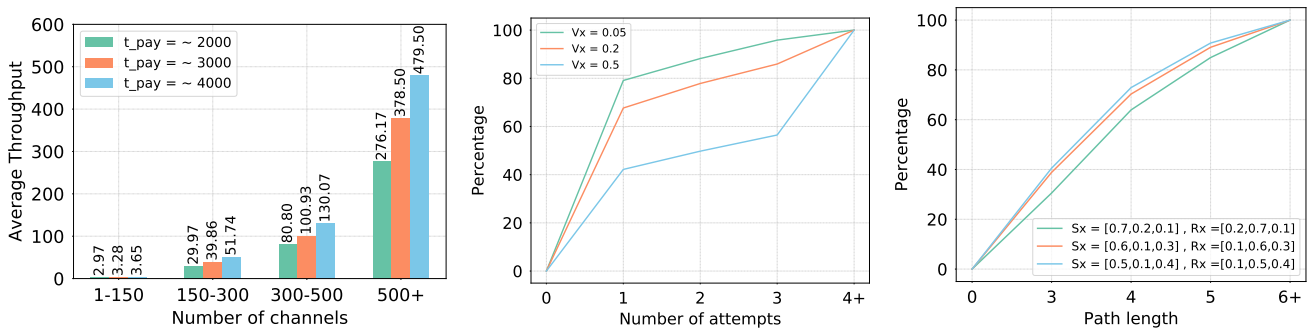
Path length. Finally, we focus on the length of the paths chosen by senders, and how it is affected by the category and connectivity of the sender. As such, we fix both t_{pay} and V_x, and instead choose different values of $S_x = [S_u, S_m, S_r]$ and $R_x = [R_u, R_m, R_r]$, as shown in Figure 2c. In particular, by definition Routers have higher connectivity compared to the other two categories, so we aimed to measure how the presence of well-connected nodes on the endpoint of a payment affected the number of intermediate hops.

Indeed, we see in Figure 2c that when gradually increasing S_r and R_r (the sending and receiving ratios of a Router), the attempted paths become smaller. For example, when $S_r = R_r = 0.1$, the probability of a path being of length three is 30.08% but when $S_r = R_r = 0.4$, the probability becomes 40.52%. The difference between different parameter choices here, however, is noticeably smaller than with our other measurements, which suggests that the network topology and the client path finding algorithm have a much larger effect on the path length than the number of Routers. This agrees with recent research (also based on a simulated network) showing that paths with one intermediate node are more prevalent than all other path lengths [5].

5 Balance Discovery

In this section we perform two attacks on the LN testnet and mainnet in order to discover the actual balances associated

[9] The Poisson distribution is not bounded, so we resample all the values that happen to be outside this range.



(a) Average daily throughput per category.

(b) Number of paths tried.

(c) Average path length.

Figure 2: The effect of different parameters on different aspects of our simulated Lightning network. The detailed simulation settings are presented in Appendix, Table 1.

with individual channels, as opposed to just their overall capacity. The first technique, which we call an *oracle-aided* attack, has been discussed in the literature [16] and even carried out for a part of testnet, by targeting the top 11 nodes that account for 50% of the total number of open channels.

This attack requires the attacker to run only a single node, but it must exploit debug information returned by error messages. As such, it would no longer be possible if error messages were generic. We run this attack on all nodes in testnet and run a partial attack on mainnet.

The second technique, which we call a *generic* attack, is new. Here, the attacker must run two nodes, and the attack exploits messages in the log files of the attacker's nodes. Due to the extra cost associated with running two nodes, we run this attack on testnet but run only a restricted version of it on mainnet.

5.1 Oracle-aided balance inference

As outlined in Section 2, a number of conditions must be satisfied in order for a node to successfully claim a payment.

If these conditions are met and a payment is successfully routed, the sender will eventually receive a hash pre-image x. If unsuccessful, then according to the current LN specification they will receive an error with some debug information that allows them to identify the problem. This information acts as an *oracle* and can be exploited to determine the exact balance of the channel at the time at which the attack is performed.

To carry out this attack, we open a channel with each node in the network and forward a fake payment hash, with some associated payment amount amt, through each of its channels. If we receive an `unknownpaymenthasherror` then we know that the channel is able to forward payments this big; i.e., that this side of the channel has a balance of at least amt. We then repeat the attack using a binary search on the payment amount, reducing it if we receive an `InsufficientFunds` or `TemporaryChannelFailure` error message and increasing it if we receive an `unknownpaymenthasherror` or `IncorrectOrUnknownPaymentDetails` message. We run this until we identify the exact balance with 1-satoshi-level precision.

The cost of this attack is the cost of opening and closing the channels, which requires on-chain transaction fees, and the opportunity cost of investing the coins elsewhere.

5.1.1 Testnet results

We attacked the testnet most recently from January 7, 2020 to January 9, 2020. The attack took a number of days because we allow the node to open a new channel, wait for confirmation, run the attack, close the channel, and then repeat. Alternatively, one could parallelise the process, opening a large number of channels at once, and then attack them simultaneously. In particular, opening and closing a channel is a time-consuming process due to the latency involved in having a transaction confirmed on the Bitcoin blockchain. Testnet coins are of essentially no value, so the monetary cost for us to perform this attack was negligible. Due to our local configurations, we attempted to connect only to nodes with public IPv4 addresses.

At the start of the attack the network had 3,035 nodes sharing 8,665 channels with a total capacity of 592.22 BTC. Of these, we were able to connect to and open direct channels with 132 nodes, attempting to attack 4,585 channels. We discovered the balances of 619 channels, worth a collective 47.68 BTC.

Given that the majority (90.86%) of the channels we attempted to attack failed, we looked into the reasons why. On many of them, we received errors such as `ChannelDisabled` (meaning the channel was disabled due to an issue with one of the nodes), `PermanentChannelFailure` (there was an issue along the route), `UnknownNextPeer` (the target node could not be found despite having a channel, e.g. zombie channels) or we timed out as the client took more than 30 seconds to return a response. Others returned errors due to our route not containing correct channel properties, an incorrect fee calcu-

lation, or incorrect expiry times. Moreover, we were not able to connect to a majority of the overall nodes, which again happened for a variety of reasons: some nodes did not publish an IPv4 address, some were not online, some had their advertised LN ports closed, and some even refused to open a channel.

Of the channels where we were able to discover the balance, we considered how one-sided the channels were by exploring what percentage of the overall channel capacity was held by the attacked party, as shown in Figure 3a. As we can see, many (45%) of the channels were fairly one-sided, with the attacked party holding 90% or more of the total capacity.

5.1.2 Mainnet results

Given our limited supply of bitcoins and their high price, we targeted only a limited set of nodes on mainnet to keep our attack within a relatively modest budget, in total opening 0.131124 BTC worth of channels (976.78 USD at the time), paying 3.92 USD to open and close channels.

We carried out the attack on mainnet most recently from December 18, 2019 to December 19, 2019. At this point in time the network had 6,107 nodes and 35,069 channels, with a total collective capacity of 859.53 BTC.

We attempted to open channels with the top ten nodes by capacity, but only successfully opened channels with six, as the remaining four refused the connection, timed out, or did not broadcast an IPv4 address. We attacked $1,293$ channels, and were able to infer the balance of 678 channels with a total capacity of 6.06 BTC.

Similarly to our attack on testnet, 47.56% of our channel attacks failed due to a variety of errors. Of those that were successful, we again looked at how one-sided the channels were; the results are in Figure 3b. The result here is ultimately similar to the one in testnet, with 50% of the channels having at least 80% of the funds controlled by the attacked party.

5.2 Generic balance inference

As discussed above, the oracle-aided attack relies on the debug information available inside error messages in the current implementation of LN. This still works as of version 0.9.0-beta of the lnd client, but its reliance on implementation details raises the question of how else balances could be discovered, and in particular how to do it in a generic way.

Our generic attack requires running two nodes: one with a channel with outgoing balance, and one with a channel with incoming balance. This is because the goal for an attacker controlling nodes A and D is to form a path $A \to B \to C \to D$ in order to find the balance of the channel $B \to C$. Obtaining the channel $A \to B$ is easy, as the attacker can just open a channel with B and fund it themselves. Opening the channel $C \to D$ is harder though, given that the attacker must create incoming balance.

Today, there are essentially two options for doing this. In the first, the attacker can open the channel $C \to D$ and fund it themselves, but choose to assign the balance to C rather than to D (this is called funding the "remote balance" due to the structure of transactions in Lightning). This presents the risk, however, that C will immediately close the channel and take all of its funds. The second option is thus to use a *liquidity provider* (e.g., Bitrefill[10] or LNBIG[11]), which is a service that sells channels with incoming balance.[12]

5.2.1 The attack

Once the attacker sets up the configuration $A \to B \to C \to D$ described above, their goal is to learn the balance of the intermediate channel $B \to C$. To do this, they route a payment hash H through B and C with the destination set to D and with some associated amount amt. They then wait until the payment hash arrives at D. If D does receive H, this means the channel from B to C had sufficient balance to route a payment of size amt. If D did not receive H after some timeout, the attacker can assume the payment failed, meaning the value exceeded the balance from B to C. Either way, the attacker can (as in the oracle-aided attack) repeat the process using a binary search on the value. Eventually, the attacker can discover the balance of the channel as the maximum value for which D successfully receives H.

This attack generalizes to a certain extent even to the case in which there is more than one intermediate channel between the two attacker nodes. In this more general case, however, the above method identifies the bottleneck balance in the entire path, rather than the balance of an individual channel.

5.2.2 Attack results

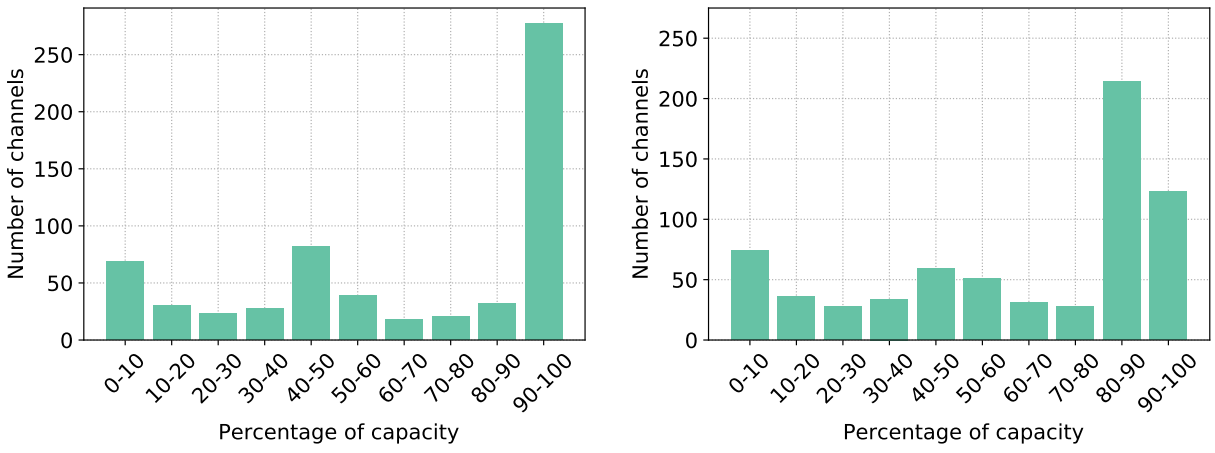
We attempted to perform this attack on testnet multiple times, as recently as February 14, 2020. Every channel in the testnet we attacked returned a balance of 739 satoshis, however, giving only `TemporaryChannelFailure` as error messages. Given the results from the oracle-aided attack, we knew this was incorrect. We are unsure of the exact issue, given the myriad difficulties we encountered working with Lightning; this again highlights the difficulty of working with live networks as compared to blockchain data.

We also performed this attack on mainnet on February 14, 2020. As we did not want to give away our bitcoin to the funder, we paid a service to open and fund a remote channel to our node. More precisely, we paid for one channel (25 USD) from liquidity provider Bitrefill, with a remote capacity of 0.165 BTC. We then analyzed the channels of the Bitrefill node, randomly selected two target channels and connected

[10] https://www.bitrefill.com/

[11] http://lnbig.com/

[12] Bitrefill, for example, sells a channel with an incoming balance of 5000000 satoshis (the equivalent as of the time of writing of 493.50 USD) for 8.48 USD.



(a) Testnet

(b) Mainnet

Figure 3: Figures for our oracle-aided balance inference attack. Each one shows the number of channels where the percentage of the capacity that was held by the attacked party (as calculated from the discovered balance) was within a given range.

to their endpoints, funded local channels with each (with a higher capacity than the target), and ran the attack. These two channels had a total capacity of 0.121639 BTC (1254.71 USD). We found that both targets had balances that were slightly less than half the total capacity of the channel. This shows that our generic attack is feasible, despite not working on testnet. In the sections that follow, we use the balances discovered by the (successful) oracle-aided attack.

6 Payment Discovery

In this section, we analyze the off-path payment privacy in Lightning, in terms of the ability of an attacker to learn information about payments it did not participate in routing.

Informally, our attack works as follows: using the balances learned in the attack described in Section 5, the attacker constructs a *network snapshot* at time t consisting of all channels and their associated balances. It then runs the attack again at some later time $t+\tau$ and uses the differences between the two snapshots to infer information about payments that took place by looking at any paths that changed. In the simplest case that only a single payment took place between t and $t+\tau$ (and assuming all fees are zero), the attacker can see a single path in which the balances changed by some amount amt and thus learn everything about this payment: the sender, the recipient, and the amount amt. More generally, two payments might overlap in the paths they use, so an attacker would need to heuristically identify such overlap and separate the payments accordingly.

6.1 Payment discovery algorithm

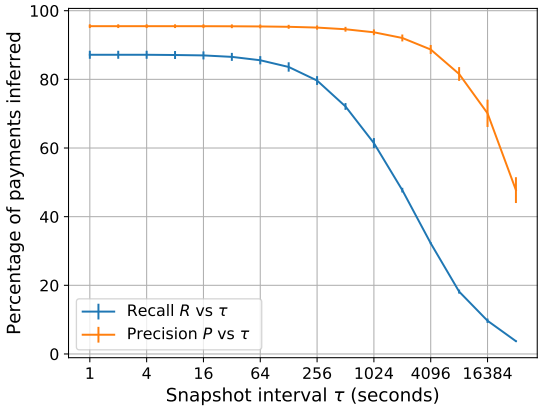
We define τ to be the interval in which an attacker is able to capture two snapshots, S_t and $S_{t+\tau}$ and let $G_{\text{diff}} = S_{t+\tau} - S_t$ be the difference in balance for each channel. Our goal is then to decompose G_{diff} into paths representing distinct payments. More specifically, we construct paths such that (1) each edge on the path has the same amount (plus fees), (2) the union of all paths results in the entire graph G_{diff}, and (3) the total number of paths is minimal. This last requirement is to avoid splitting up multi-hop payments: if there is a payment from A to D along the path $A \to B \to C$, we do not want to count it as two (equal-sized) payments of the form $A \rightsquigarrow B$ and $B \rightsquigarrow C$.

We give a simple algorithm that solves the above problem under the assumption the paths are disjoint. This assumption may not always hold, but we will see in Section 6.3 that it holds for the most part when the interval between snapshots is relatively short. As such, the algorithm heuristically performs well. Our algorithm proceeds iteratively by "merging" payment paths. We initially consider each non-zero edge in G_{diff} as a distinct payment. We then select an arbitrary edge with difference amt, and merge it with any adjacent edges with the same amount (plus the publicly known fee f) until no edge of weight amt can be merged.

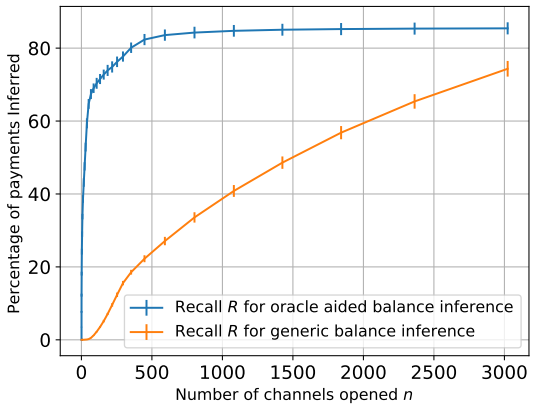
$$A \xrightarrow{\mathsf{amt}+f_{A,B}+f_{B,C}} B,\ B \xrightarrow{\mathsf{amt}+f_{B,C}} C \ \Rightarrow\ \text{infer payment } A \rightsquigarrow C$$

We then remove this path from G_{diff} and continue with another edge. Asymptotically the running of this algorithm is $O(|E|^2)$ for E edges; given the size and sparsity of Lightning Network today this means it runs in under a second.

There are several ways this algorithm can make incorrect inferences. First, it would incorrectly merge two same-valued payments $A \rightsquigarrow B$ and $B \rightsquigarrow C$ occurring end-to-end. This is unlikely to occur often because (1) payments are denominated in fine-grained units (1 satoshi = 10^{-8} BTC), and (2) wallets typically support generating payment invoices in units of fiat currency by applying the real-time Bitcoin exchange rate, which fluctuates. Second, our algorithm does not attempt to



(a) The precision and recall of our payment discovery attack, based on the snapshot interval τ (in log scale). The error bars show 95% confidence intervals over five simulation runs.



(b) Recall R as a function of the number of channels required. The error bars indicate a 95% confidence interval over five simulation runs.

Figure 4: Results of simulated payment detection attack.

resolve the case that a single channel is used for multiple payments in an interval. Looking ahead to Section 6.3, our experiments show this happens infrequently when the snapshot intervals are short enough. Finally, as we saw in Section 5, balance discovery may fail for some (or many) channels in the network. To cope with this, our algorithm takes a conservative approach designed to minimize the false positive rate: as a final filtering step, it suppresses any pairs of inferred payments with approximately the same amount (within a small threshold of 25 satoshis).

6.2 Attack simulation

We denote the attacker's precision by P (the number of correctly detected payments divided by the total number of detected payments) and the attacker's recall by R (the number of correctly detected payments divided by the number of actual payments). We are primarily interested in understanding how these performance metrics depend on the interval at which an attacker takes snapshots (τ) and the number of channels (n) opened by the attacker.

To answer these questions we leverage the simulator we developed in Section 4, and extend it to include the balance discovery attack from Section 5. Of the multiple reasons that our balance discovery attacks failed, we observe that 98% of the errors were because the peer was not online or did not participate in any payments. Therefore we set a 0.05 probability for balance inference failing on a channel in which both peers are online. In keeping with the discussion in Section 4.2, we use $t_{pay} = 2000$ as the total number of payments per day, $V_x = 0.2$, $S_x = [0.7, 0.2, 0.1]$ and $R_x = [0.2, 0.7, 0.1]$.

To run our experiments, we use the simulator to generate a sequence of payment and channel balance change events. We perform five such runs, each simulating a period of 30 days. This amounts to 60,000 total payments per simulation. When reporting confidence intervals, we consider these five independent simulation run samples.

6.3 Simulated attack results

6.3.1 Effect of snapshot interval (τ) on P and R

For this experiment, we take balance inference snapshots of the entire network for varying time intervals, ranging from $\tau = 1$ second to $\tau = 2^{15}$ seconds. Figure 4a shows the relationship between τ and the number of payments inferred and confirms the intuition that the attack is less effective the longer the attacker waits between snapshots, as this causes overlap between multiple payments. At some point, however, sampling faster and faster offers diminishing returns; e.g., for $\tau = 128$ seconds, the attacker has a recall R of 82.6%, and for $\tau = 64$ seconds the recall R is 84.6% and increases slowly to 86.1% for $\tau = 1$ second. With a realistic minimum of $\tau = 30$ seconds, which is the time it took us to run the balance discovery attack on a single channel (the same number has been reported by Herrera-Joancomartí et al. [16]), the attacker has a recall of more than 85.5%. Because of our final filtering step in our discovery algorithm, we have a precision P very close to 94% for smaller values of τ.

6.3.2 Effect of attacker budget on Recall R

Next, we consider a budget-constrained attacker that can only open channels with a subset of n nodes rather than the entire network. We consider an attacker that chooses the top n nodes with the greatest number of channels. Here we fix the snapshot

interval of $\tau = 30$ seconds (which, based on our experiment in Section 5, we believe to be a feasible minimum).

Figure 4b shows how the number of nodes (n) to which the attacker opens a channel affects the number of payments inferred. For the same number of nodes attacked, we see that the oracle-aided attack performs significantly better than the generic attack. This is because the generic attack requires successfully connecting to both nodes involved in a channel, whereas either node can suffice for the oracle-aided attack to succeed. This is also why we see the recall for the oracle-aided attack taper off whereas it continues to increase gradually (but with a decreasing slope) for the generic attack.

As we observe in the oracle-aided attack especially, creating more channels provides diminishing returns after a certain point. This is expected given that most payments are routed through a small subset of nodes (Figure 2a). This suggests that if the attacker is operating with a relatively modest budget, they can choose to attack a small subset of nodes rather than the entire network. In our simulation, the oracle-aided attacker achieved a recall of 70.3% after opening only 100 channels.

Additionally, an attacker may choose to target a subset of nodes, or even a single target node, for other reasons; e.g., if it is particularly interested in their payment activities and less interested in the broader network. By connecting to and then continuously probing the target node to observe changes in its balances, the attacker can identify its payments: if the node was an intermediate routing node, then the channels around it should have similar incoming and outgoing balances. If instead only a single channel changed by a particular amount, that node must have been either the sender or the recipient of a payment of that amount.

7 Path Discovery

In this section, we describe an attack on the on-path relationship anonymity of Lightning, in which an *honest-but-curious* intermediate node involved in routing the payment can infer information regarding its path, and in particular its sender and recipient.

In contrast to previous work by Béres et al. [5], we consider not only single-hop routes but also routes with multiple intermediate nodes. In particular, we leverage our simulator to define a probability distribution over path lengths and attempted paths per payment, as already explored in Section 4.3. We then combine this distribution with the topology of the Lightning Network to define the probability of a successful attack; i.e., of an intermediate node correctly guessing the sender and recipient of a payment it was involved in routing. The only assumption we make about the attacker's intermediate node is that it has enough balance to forward every payment. This can be easily done in practice by an attacker rebalancing their channels or running multiple nodes.

We define as $\Pr_S$ the probability that the attacker successfully discovers the sender, and as $\Pr_R$ the probability that they successfully discover the recipient. Following our notation, Béres et al. claimed, based on their own simulated results, that $\Pr_S = \Pr_R$ ranges from 0.17 to 0.37 depending on parameters used in the simulation. We show that this probability is actually a lower bound, as it does not take into account multiple possible path lengths or the chance that a payment fails (their simulation assumes that all payments succeed on the first try). In particular, we discuss below the ways in which failed payments leak additional information.

We start by defining the strategy of our attacker. In particular, whenever they have to guess the sender, they always guess their immediate predecessor. In other words, if we define H as the attacker's position along the path, they always assume that $H = 1$. Similarly, whenever they have to guess the recipient, they always guess their immediate successor. Next, we define the probability of attacker's success. We focus on the probability of succesfully guessing the sender, however, the probability of succesfully guessing the recipient can be computed likewise.

Successful payments. We start by analyzing the success probability of this attacker in the case of a successful payment, which we denote as $\Pr_S^{\mathsf{succ}}$. We define as $\Pr[L = \ell]$ the probability of a path being of length ℓ, and as $\Pr[H = h \mid L = \ell]$ the probability that the attacker's node is at position h given that the path length is ℓ. According to the Lightning specification [2], the maximum path length is 20. By following the strategy defined above, we have that

$$\begin{aligned}\Pr_S^{\mathsf{succ}} &= \sum_{n=3}^{20} \Pr[L = \ell \mid \mathsf{succ}] \cdot \Pr[H = 1 \mid L = \ell, \mathsf{succ}] \\ &= \Pr[L = 3 \mid \mathsf{succ}] \\ &\quad + \sum_{n=4}^{20} \Pr[L = \ell \mid \mathsf{succ}] \cdot \Pr[H = 1 \mid L = \ell, \mathsf{succ}]\end{aligned}$$

since $\Pr[H = 1 \mid L = 3, \mathsf{succ}] = 1$ given that the attacker is the only intermediate node in this case. Hence, $\Pr[L = 3 \mid \mathsf{succ}]$ is a lower bound on the attacker's success probability.

To consider the overall probability, we focus on the conditional probabilities $\Pr[H = 1 \mid L = \ell, \mathsf{succ}]$. If all nodes involved in the payment form a clique,[13] then it would be almost equally probable for any node to be in any hop position $H = h$. The only reason the distribution is not entirely uniform is that some channels may be chosen more often than others, depending on the relative fees they charge, but we assume the attacker would choose fees to match its neighbors as closely as possible. In this case then, the probability that $H = 1$ is just $1/(\ell - 2)$; i.e., the attacker is equally likely to be in any intermediate position. The success probability thus

[13] This would rather be a clique excluding a link between the sender and recipient, since otherwise they would presumably use their channel directly.

becomes

$$\Pr_S^{\mathsf{succ}} = \Pr[L = 3 \mid \mathsf{succ}] + \sum_{n=4}^{20} \Pr[L = \ell \mid \mathsf{succ}] \cdot \frac{1}{\ell - 2}.$$

Failed payments. Similarly, in case the payment fails, we define the probability $\Pr_S^{\mathsf{fail}}$ as

$$\Pr_S^{\mathsf{fail}} = \sum_{\ell=3}^{20} \Pr[L = \ell \mid \mathsf{fail}] \cdot \Pr[H = 1 \mid L = \ell, \mathsf{fail}].$$

This is the same formula as for $\Pr_S^{\mathsf{succ}}$ so far, but we know that $\Pr[L = 3 \mid \mathsf{fail}] = 0$, since if the attacker is the only intermediate node the payment cannot fail. Furthermore, the conditional probability $\Pr[H = 1 \mid L = \ell, \mathsf{fail}]$ is different from the probability $\Pr[H = 1 \mid L = \ell, \mathsf{succ}]$, as the fact that a payment failed reveals information to the attacker about their role as an intermediate node. In particular, if an intermediate node successfully forwards the payment to their successor but the payment eventually fails, the node learns that their immediate successor was not the recipient and thus that the failed path was of length $L \geq 4$ and their position is not $L-1$. This means that $\Pr[L = \ell \mid \mathsf{fail}]$ becomes $\Pr[L = \ell \mid \mathsf{fail}, \ell \geq 4]$. We thus get

$$\Pr_S^{\mathsf{fail}} = \Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4] + \sum_{\ell=5}^{20} \Pr[L = \ell \mid \mathsf{fail}] \cdot \Pr[H = 1 \mid L = \ell, \mathsf{fail}].$$

This gives $\Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4]$ as a lower bound in the case of a failed payment. As we did in the case of successful payments, we assume a clique topology as the best case for this attacker's strategy, in which their chance of guessing their position is $1/(\ell - 3)$ (since they know they are not the last position). We thus obtain

$$\Pr_S^{\mathsf{fail}} = \Pr[L = 4 \mid \mathsf{fail}, \ell \geq 4] + \sum_{\ell=5}^{20} \Pr[L = \ell \mid \mathsf{fail}] \cdot \frac{1}{\ell - 3}.$$

Attack measurements. In Section 4.3, we empirically estimated the probabilities $\Pr[L = \ell]$ for $3 \leq \ell \leq 20$ given different parameters S_x and R_x. We now use those results to compute the probabilities defined above. The results are depicted in Figure 5.

$\Pr_S^{\mathsf{succ}}$ is bounded from below by the probability of the scenario in which $L = 3$, since in that case the attacker can never be wrong. Similarly, the probability of $L = 4$ gives us the lower bound on $\Pr_S^{\mathsf{fail}}$. In case of a successful payment, lower bound probability varies from 30.08% ($\mathsf{Sim}_1^{\mathsf{S}}$) to 40.52% ($\mathsf{Sim}_3^{\mathsf{S}}$). On the other hand, the lower bound of $\Pr_S^{\mathsf{fail}}$ increases with the percentage of unsuccessful attempts, even up to 74.14% ($\mathsf{Sim}_3^{\mathsf{F}}$), which is significantly higher than any recorded experiment in previous works. Such high probability of attacker's success is not just a corner case. According to the recent measurements presented in [1] 34% payments failed at the first try. Furthermore, our simulations showed that for an increased value of payments (Sim_6 with $V_X = 0.5$), 58.53% of the simulated payments had to try at least two different paths in order to successfully go through (see Figure 2b). This translates to 8,016 failed attempts in total, in which an adversarial intermediate node had 71.87% probability of knowing who the sender is.

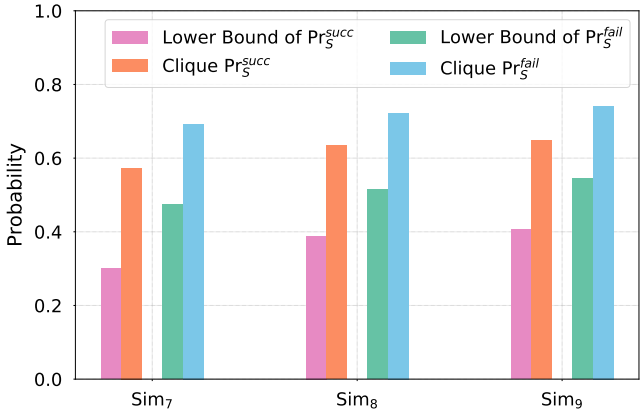


Figure 5: Probability of correctly identifying the sender given successful and failed payment. For detailed simulation settings of $\mathsf{Sim}_7, \mathsf{Sim}_8, \mathsf{Sim}_9$ see Table 1 in Appendix B.

We computed $\Pr_S^{\mathsf{succ}}$ and $\Pr_S^{\mathsf{fail}}$ given our strategy and a clique topology. We note, however, that these probabilities might be even higher if the network graph leaks additional information (e.g., if the preceding node has only one channel with enough capacity to support the payment etc.), or the payment fails repeated number of times and hence we can look for intersection between the neighbour's of involved nodes.

8 Conclusions

In this paper, we systematically explored the three main privacy properties of the Lightning Network and showed that, at least in its existing state, each property is susceptible to attacks by nodes who are either active (in the case of our balance and payment discovery attacks) or passive (in the case of our path attack). Unlike previous works that demonstrate similar gaps between theoretical and achievable privacy in cryptocurrencies, our research does not rely on patterns of usage or user behavior. Instead, the same interfaces that allow users to perform the basic functions of the network, such as connecting to peers and routing payments, can also be exploited to learn information that was meant to be kept secret. This suggests that these limitations may be somewhat inherent, or at least that avoiding them would require changes at the design level rather than at the level of individual users.

Acknowledgements

George Kappos, Haaroon Yousaf and Sarah Meiklejohn are supported in part by EPSRC Grant EP/N028104/1, and in part by the EU H2020 TITANIUM project under grant agreement number 740558. Sanket Kanjalkar and Andrew Miller are supported by the NSF under agreement numbers 1801369 and 1943499. Sergi Delgado-Segura was partially funded by EPSRC Grant EP/N028104/1.

References

[1] The lightning conference: Of channels, flows and icebergs talk by christian decker. https://www.youtube.com/watch?v=zk7hcJDQH-I.

[2] Lightning network specifications. https://github.com/lightningnetwork/lightning-rfc.

[3] Person behind 40% of LN's capacity: "I have no doubt in Bitcoin and the Lightning Network". https://www.theblockcrypto.com/post/41083/person-behind-40-of-lns-capacity-i-have-no-doubt-in-bitcoin-and-the-lightning-network.

[4] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 2013.

[5] Ferenc Béres, Istvan Andras Seres, and András A Benczúr. A cryptoeconomic traffic analysis of Bitcoins Lightning network. *arXiv:1911.09432*, 2019.

[6] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. Privacy aspects and subliminal channels in Zcash. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[7] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of ACM CCS*, 2014.

[8] Ivan Bogatyy. Linking 96% of Grin transactions. https://github.com/bogatyy/grin-linkability.

[9] Simina Brânzei, Erel Segal-Halevi, and Aviv Zohar. How to charge Lightning. *arXiv:1712.10222*, 2017.

[10] Marco Conoscenti, Antonio Vetrò, Juan Carlos De Martin, and Federico Spini. The cloth simulator for HTLC payment networks with introductory lightning network performance results. *Information*, 2018.

[11] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*. Springer, 2016.

[12] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *30th IEEE Symposium on Security and Privacy*, 2009.

[13] Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. Towards an economic analysis of routing in payment channel networks. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2017.

[14] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

[15] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. *IACR Cryptology ePrint Archive*, 2019.

[16] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. On the difficulty of hiding the balance of Lightning Network channels. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (CCS)*, 2019.

[17] Abraham Hinteregger and Bernhard Haslhofer. Short paper: An empirical analysis of Monero cross-chain traceability. In *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC)*, 2019.

[18] Harry A. Kalodner, Steven Goldfeder, Alishah Chator, Malte Möser, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. *arXiv:1709.02489*, 2017.

[19] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in Zcash. In *27th USENIX Security Symposium '18*.

[20] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453, 2017.

[21] Nida Khan et al. Lightning network: A comparative review of transaction fees and data analysis. In *International Congress on Blockchain and Applications*, pages 11–18. Springer, 2019.

[22] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in Bitcoin using P2P network traffic. In *International Conference on Financial Cryptography and Data Security (FC)*, 2014.

[23] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of Monero's blockchain. In *European Symposium on Research in Computer Security*. Springer, 2017.

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

[25] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous Multi-Hop Locks for blockchain scalability and interoperability. In *Proceedings of NDSS*, 2018.

[26] Stefano Martinazzi. The evolution of lightning network's topology during its first year and the influence over its core values. *arXiv preprint arXiv:1902.07307*, 2019.

[27] Sarah Meiklejohn and Claudio Orlandi. Privacy-enhancing overlays in Bitcoin. In *Proceedings of the 2nd Workshop on Bitcoin Research*, 2015.

[28] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of Bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM.

[29] Malte Möser and Rainer Böhme. Anonymous alone? measuring Bitcoin's second-generation anonymization techniques. In *IEEE Security & Privacy on the Blockchain (S&P)*, 2017.

[30] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the Monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018.

[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[32] Mariusz Nowostawski and Jardar Tøn. Evaluating methods for the identification of off-chain transactions in the Lightning Network. *Applied Sciences*, 2019.

[33] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable off-chain instant payments, 2016.

[34] Jeffrey Quesnelle. On the linkability of Zcash transactions. *arXiv:1712.01210*, 2017.

[35] Fergal Reid and Martin Harrigan. An analysis of anonymity in the Bitcoin system. In *Security and Privacy in Social Networks*. Springer, 2013.

[36] BitMEX Research. Lightning Network (Part 7) – Proportion of Public vs Private Channels. https://blog.bitmex.com/lightning-network-part-7-proportion-of-public-vs-private-channels/.

[37] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 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)*, pages 347–356. IEEE, 2019.

[38] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*. Springer, 2013.

[39] István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. Topological analysis of Bitcoin's Lightning Network. *arXiv:1901.04972*, 2019.

[40] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitiodine: Extracting intelligence from the Bitcoin network. In *International Conference on Financial Cryptography and Data Security*. Springer, 2014.

[41] Saar Tochner, Stefan Schmid, and Aviv Zohar. Hijacking routes in payment channel networks: A predictability tradeoff. *arXiv:1909.06890*, 2019.

[42] Shira Werman and Aviv Zohar. Avoiding deadlocks in payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 175–187. Springer, 2018.

[43] Jin Y Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 1970.

[44] Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 837–850, 2019.

[45] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupen Yang, Quiliang Xu, and Wang Fat Lau. New empirical traceability analysis of CryptoNote-style blockchains. In *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC)*, 2019.

[46] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *IEEE International Conference on Communications (ICC) 2019*.

A More Details on How LN Works

A.1 Channel management

In this section, we describe in detail the channel management operations, including its opening and closing, and channel state updates.

A.1.1 Opening a channel

For Alice and Bob to create a channel between them, they must fund the channel by forming a *funding transaction* $\mathsf{tx}_{\mathsf{fund}}$. Let's assume that Alice is the one funding the channel.[14] The funding transaction consists of one input, which is a UTXO associated with one of Alice's addresses, and one output, which is a 2-of-2 multisig address representing both Alice and Bob's public keys. This means that both Alice and Bob must provide their signatures in order to release the funds. The amount sent to the multisig address is the initial capacity C of the channel.

It is important that Alice does not just put this transaction on the Bitcoin blockchain; otherwise, her coins may be locked up if Bob refuses to provide a signature. Instead, she and Bob first go on to form two *commitment transactions*, one for each of them, which represent their agreement on the current state of the channel. Each commitment transaction has the 2-of-2 multisig as input, and has two outputs: local and remote. The transaction $\mathsf{tx}_{A,i}$, representing Alice's view of state i, essentially sends her current balance to local and sends Bob's current balance to remote. Likewise, $\mathsf{tx}_{B,i}$ sends Bob's balance to local and Alice's balance to remote.

The remote output is simply the address of the other involved party (so addr_B in $\mathsf{tx}_{A,i}$ and addr_A in $\mathsf{tx}_{B,i}$), but the local output is more complicated. Instead, the local output in $\mathsf{tx}_{B,i}$ is encoded with some timeout t, and awaits some potential input σ. If the timeout t has passed, then the funds go to addr_B as planned. Otherwise, if a valid *revocation signature* σ is provided before time t, the funds go to addr_A. This means that the revocation signature allows one party to claim the entire capacity of the channel for themselves. As we explore fully in Section A.1.2, this is used to disincentive bad behavior in the form of publishing old states.

To create a channel, Alice thus creates the transaction $\mathsf{tx}_{B,0}$, sending C to remote and 0 to local (since so far she has supplied all the funds for the channel). She signs this transaction and sends her signature to Bob, who signs it as well. Bob then forms $\mathsf{tx}_{A,0}$, sending 0 to remote and C to local, and sends his signature on this to Alice. Alice then signs it and publishes $\mathsf{tx}_{\mathsf{fund}}$ to the Bitcoin blockchain.

At this point, Alice and Bob both have valid transactions, meaning transactions that are signed by both parties in the input multisig, which is itself the output of a transaction on the blockchain (i.e., the funding transaction). Either of them could thus publish their transaction to the blockchain to close the channel. Alternatively, if they mutually agree to close the channel and want to avoid having one of them wait until the timeout to claim their funds, they could update to a new state without the timeout in local and publish that.

[14] In general, either one of the parties funds the channel, or both of them.

A.1.2 Updating a channel state

Once both Alice and Bob have signed $\mathsf{tx}_{A,0}$ and $\mathsf{tx}_{B,0}$, they have agreed on the *state* of the channel, which represents their respective balances B_A and B_B. In particular, the amount sent to local in $\mathsf{tx}_{A,0}$ and to remote in $\mathsf{tx}_{B,0}$ represents Alice's balance, and similarly the amount sent to local in $\mathsf{tx}_{B,0}$ and to remote in $\mathsf{tx}_{A,0}$ represents Bob's balance. The question is now how these transactions are updated when one of them wants to pay the other one, and thus these balances change.

In theory, this should be simple: Alice and Bob can repeat the same process as for the creation of these initial commitment transactions, but with the new balances produced by the payment. The complicating factor, however, is if the old commitment transactions are still valid after the creation of the new ones, then one of them might try to revert to an earlier state by broadcasting an old commitment transaction to the Bitcoin blockchain. For example, if Alice pays Bob in exchange for some service, then it is important that once the service has been provided, Alice cannot publish an old commitment transaction claiming her (higher) balance before she made the payment.

This is exactly the role of the *revocation signature* introduced earlier. In addition to exchanging signatures on the new transactions $\mathsf{tx}_{A,i+1}$ and $\mathsf{tx}_{B,i+1}$, Alice and Bob also exchange the revocation secret keys $\mathsf{sk}_{A,i}$ and $\mathsf{sk}_{B,i}$ that correspond to the public keys $\mathsf{pk}_{A,i}$ and $\mathsf{pk}_{B,i}$ used in $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$. These public keys are derived from base public keys pk_A and pk_B such that (1) both Alice and Bob can compute $\mathsf{pk}_{A,i}$ and $\mathsf{pk}_{B,i}$ for any state i, thus can independently form $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$ as long as they know the right amounts, and (2) the corresponding secret keys $\mathsf{sk}_{A,i+1}$ and $\mathsf{sk}_{B,i+1}$ are completely unknown until one party reveals them to the other. Thus, up until they exchange revocation keys, they can broadcast the transactions for state i to the Bitcoin blockchain as a way to close the channel. Once they exchange the secret keys, however, Bob can form a valid revocation signature and claim all of Alice's funds in local if she broadcasts $\mathsf{tx}_{A,i}$ (it is indeed only Alice who can broadcast a valid $\mathsf{tx}_{A,i}$ as only she has both her and Bob's signatures on it). At this point the new channel state $i+1$ is confirmed and Bob can safely perform his service for Alice.

A.1.3 Closing a channel

As long as either Alice or Bob has a positive balance, they can send payments to the other, knowing that if there is a disagreement they can settle their previously agreed channel state onto the blockchain, as described in the previous section. If there is no dispute, and both Alice and Bob agree to close the channel, they perform a *mutual close* of the channel. This means providing a signature that authorizes a *settlement transaction*.

A.2 Hashed time-lock contracts (HTLCs)

The previous section describes how the state of a two-party channel can be updated, in which both Alice and Bob are sure that they want a payment to go through, and can thus transition to the new state immediately. In a broader network of channels, however, it may very well be the case that two parties need to prepare their channel for an update that does not happen. To see why, remember from Section 2 that when Alice is picking a path from herself to Bob, the information she has about the channels along the way is their capacity C, which is $\mathsf{C} = \mathsf{C}_{\mathsf{in}} + \mathsf{C}_{\mathsf{out}}$. If $\mathsf{cid}(U_{n-1} \leftrightarrow U_n)$ has capacity $\mathsf{C} \geq \mathsf{amt}$ but $\mathsf{C}_{\mathsf{out}} < \mathsf{amt}$ (i.e., U_{n-1} does not have enough to pay U_n the amount Alice is asking), then the payment fails at this point, so no one along the path should have actually sent any money. Equally, the payment could fail due to a malicious intermediary simply deciding not to forward the onion packet or otherwise follow the protocol.

To thus create an intermediate state, and to unite payments across an entire path of channels, the Lightning network uses *hashed time-lock contracts*, or *HTLCs* for short. In using HTLCs, Alice and Bob can still transition from $\mathsf{tx}_{A,i}$ and $\mathsf{tx}_{B,i}$ to new transactions $\mathsf{tx}_{A,i+1}$ and $\mathsf{tx}_{B,i+1}$ representing a payment of n coins from Alice to Bob, but the first message that Alice sends includes the hash h and timeout t. This signals to Bob to add an additional output htlc to $\mathsf{tx}_{A,i+1}$, which sends n coins to Alice if the time is greater than t (as a refund) and sends them to him if he provides as input a value x such that $H(x) = h$. Alice and Bob then proceed as usual in exchanging signatures and revocation keys for their respective transactions.

If at some point before t Bob sends such an x to Alice, then it is clear to both parties that Bob could claim the htlc output of $\mathsf{tx}_{A,i+1}$ if it were posted to the blockchain. They thus transition to a new state, $i+2$, in which they go back to two-output commitment transactions, with the additional n coins now added to Bob's balance.

Going back to the third step in the description provided in Section 2 then, U_{i-1} and U_i prepare their channel by updating to this intermediate state $j+1$, using the h and t_i provided in the packet onion_i to form the htlc output. In the fifth and final step, they settle their channel by updating to state $j+2$ by removing the htlc output, once U_i has sent the pre-image x to U_{i-1} and thus demonstrated their ability to claim it. If Bob never provides the pre-image x, then by the pre-image resistance of the hash function no party along the path is able to claim the htlc output, so the payment simply does not happen. If some malicious U_i along the path decides not to continue forwarding x, then all previous U_k, $1 \leq k \leq i$ can still claim the htlc output in the intermediate state.

	$\mathsf{t}_{\mathsf{pay}}$	V_x	S_u	S_m	S_r	R_u	R_m	R_r
Sim_1	2000	0.2	0.7	0.2	0.1	0.2	0.7	0.1
Sim_2	3000	0.2	0.7	0.2	0.1	0.2	0.7	0.1
Sim_3	4000	0.2	0.7	0.2	0.1	0.2	0.7	0.1
Sim_4	2000	0.05	0.7	0.2	0.1	0.2	0.7	0.1
Sim_5	2000	0.2	0.7	0.2	0.1	0.2	0.7	0.1
Sim_6	2000	0.5	0.7	0.2	0.1	0.2	0.7	0.1
Sim_7	2000	0.2	0.7	0.2	0.1	0.2	0.7	0.1
Sim_8	2000	0.2	0.6	0.1	0.3	0.1	0.6	0.3
Sim_9	2000	0.2	0.5	0.1	0.4	0.1	0.5	0.4

Table 1: Parameters used in each simulation of the Lightning network presented in Section 4.

B Summary of Simulated Parameters

Table 1 summarizes the configuration of all parameters used in the simulations presented in Section 4 and Section 7.