An Empirical Analysis of Privacy in the Lightning Network

George Kappos[1*], Haaroon Yousaf[1*], Ania Piotrowska[1,2], Sanket Kanjalkar[3], Sergi Delgado-Segura[4], Andrew Miller[3,5], 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 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, by presenting several attacks that exploit publicly available information about the network in order to learn information that is designed to be kept secret, such as how many coins a node has available or who the sender and recipient are in a payment routed through the network.

1 Introduction

Since its introduction in 2008, Bitcoin [29] has become the most widely adopted cryptocurrency. The decentralized and permissionless nature of Bitcoin allows all users to 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 of 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, however, presents two problems for the longevity of blockchain-based cryptocurrencies. First, it imposes severe scalability limitations: the Bitcoin blockchain today is over 300 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

*Both authors contributed equally.

shown most decisively for Bitcoin [4, 27, 35, 39, 41], but this type of analysis extends even to cryptocurrencies that were explicitly designed with privacy in mind [6, 17, 19, 23, 28, 34, 48].

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. 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 route payments by a fee they can charge for payments they forward.

In addition to the promise it shows in improving 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. The individual *balances* associated with the channel, however, are kept secret. Furthermore, payments are not broadcast to all peers and are not stored in a public ledger. 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. In particular, we consider four main privacy properties promised by LN [2, 24]:

Private channels should allow two nodes to share a channel but keep its existence, along with all of its information (capacity, participants, etc.), hidden from the rest of the network. We explore this property in Section 3.2 by presenting a heuristic that identifies on-chain funding of private channels and one or even both of the participants.

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 4 by presenting and evaluating a generic method by which an active attacker (i.e., one opening channels with nodes in the network) can discover channel balances.

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. We explore this property in Section 5, where we leverage an LN simulator we developed (described in Section 5.1) to evaluate the ability of an intermediate node to infer the sender and recipient in payments that it routes.

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 pay-

ment 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.

1.1 Ethical considerations

The attacks presented in Sections 5 and 6 are evaluated on a simulated network rather than the live one, but our attack in Section 4 is evaluated on the live test network. As in related active attacks on Bitcoin [7, 8, 22], 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 disclosed the results of this paper to the developers of the three main LN clients and the liquidity provider Bitrefill in February 2020, and have discussed the paper with the Lightning developers since then.

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, 45], or on its graph properties [26, 40]. Rohrer et al. [37] study the susceptibility of LN to topology-based attacks, and Tochner et al. [44] 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. Pérez-Solà et al. [32] present an attack that diminishes the capacity of a node's channels, preventing it from participating in the network. Tikhomirov et al. [42] show how a wormhole attack prevents honest intermediaries from participating in routing payments.

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. Nowatowski and Tøn [31] study various heuristics in order to identify Lightning transactions on the Bitcoin blockchain. Concurrently to our work, Romiti et al. [38] developed several heuristics to link Bitcoin wallets to Lightning entities. One of their heuristics is similar to the *tracing heuristic* we develop in Section 3.2, but their goal is to create augmented Bitcoin clustering methods rather than identify private channels. As we describe further in Section 4, others have performed balance discovery attacks [16, 30, 43]. The main limitation of these attacks is that they rely on specifics of the error messages the attacker receives, so may easily become irrelevant as the network evolves. We

overcome this limitation by presenting a *generic* attack (in Section 4), 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. Their work considers only single-hop payments, however, 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, 49]. Several of these simulators were used to perform economic analysis of the Lightning network [9, 13, 49], 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

In order to open a Lightning *channel*, two parties deposit 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 funds represent the channel *capacity*; i.e., the maximum amount of coins that can be transferred via this channel. Once a channel is established, its participants can use it to exchange arbitrarily many payments, as long as either has a positive balance. They can also close the channel using a Bitcoin transaction that sends them their respective balances from the 2-of-2 multi-signature address.

Most users, however, are not connected directly, so instead need to *route* their payments through the global Lightning Network. Here, nodes are identified by public keys, and edges represent channels, which are publicly associated with a *channel identifier* cid, the 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. Except for *private* channels, which are revealed only at the time of routing, the topology of this network and its public labels are known to every peer. When routing a payment, the sender (Alice) uses *onion routing* to hide her relationship with the recipient (Bob). Alice selects the entire path to Bob (*source routing*), based on the capacities and fees of the channels between them. The eventual goal is that each intermediate node on this path 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 it may be the case that the payment fails. To thus create an intermediate state, LN uses *hashed timelock contracts* (*HTLCs*), which allow for time-bound conditional payments. In summary, the protocol follow five basic steps to have Alice pay Bob:

1. **Invoicing** Bob generates a secret x and computes the hash h of it. He issues an *invoice* containing h and some payment amount amt, and sends it to Alice.

2. Onion routing Alice picks a path $A \to U_1 \to \cdots \to U_n \to B$. Alice then 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} *prepare* their channel by updating their intermediate state using an HTLC, which 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 this is done, U_i can send 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-1} in order to redeem his payment of amt. This value is in turn sent backwards along the path.

5. Channel settlement At every step on the path, U_i and U_{i+1} use x to *settle* their channel; i.e., to confirm the updated state reflecting the fact that amt_i was sent from U_i to U_{i+1} and thus that amt was sent from Alice to Bob.

3 Blockchain Analysis

3.1 Data and measurements

The Lightning network can be captured over time by periodic snapshots of the public network graph, which provide ground-truth data about nodes (identifiers, network addresses, status, etc.) and their channels (identifiers, capacity, endpoints, etc.). To obtain a comprehensive set of snapshots, we used data provided to us by (1) our own lnd client, (2) one of the main c-lightning developers, and (3) scraped user-submitted (and validated) data from 1ML[6] and LN Bigsun.[7] To analyze on-chain transactions, we also ran a full Bitcoin node, using the BlockSci tool [18] to parse and analyze the raw blockchain data.

Our LN dataset included the hash of the Bitcoin transaction used to open each channel. By combining this with our blockchain data, we were thus able to identify when channels closed and how their funds were distributed. In total, we identified 174,378 channels, of which 135,850 had closed with a total capacity of 3315.18 BTC. Of the channels that closed, 69.22% were claimed by a single output address (i.e., the channel was completely unbalanced at the time of closure), 29.01% by two output addresses, and 1.76% by more than two outputs.

3.2 Private channels

Private channels provide a way for two Lightning nodes to create a channel but not announce it to the rest of the network. In this section, we seek to understand the extent to which private channels can nevertheless be identified, to understand

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

[7] https://ln.bigsun.xyz/

their privacy limitations as well as the scope of our remaining attacks on the public network.

We first provide an upper bound on the number of private channels using a *property heuristic*, which identifies Bitcoin transactions that seem to represent the opening and closing of channels but for which we have no public channel identifier.

Property Heuristic To align with our LN dataset, we first looked for all Bitcoin transactions that (1) occurred after January 12, 2018 and (2) before September 7, 2020, and (3) where one of the outputs was a P2WSH address (which LN channels have to be, according to the specification). We identified 3,500,312 transactions meeting these criteria, as compared with the 174,378 public channels opened during this period. We then identified several common features of the known opening transactions identified from our dataset: (i) 99.91% had at most two outputs, which likely represents the funder creating the channel and sending themselves change; (ii) 99.91% had a single P2WSH output address; (iii) 99.85% had a P2WSH output address that received at most 16,777,215 satoshis, which at the time of our analysis is the maximum capacity of an LN channel; (iv) 99.99% 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; and (v) 99.99% were funded with either a WitnessPubHeyHash address or ScriptHash address.

By requiring our collected transactions to also have these features and excluding any transactions involved in opening or closing public channels, we were left with 267,674 potential transactions representing the opening of private channels. 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 the Lightning Network. Again, we identified the following features based on known closing transactions we had from our network data: (i) 100% had a non-zero sequence number, as required by the Lightning specification [2]; (ii) 100% had a single input that was a 2-of-2 multisig address, again as required by the Lightning design; and (iii) 98.24% 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 three features, we were left with 77,245 pairs of transactions that were potentially involved in opening and closing private channels. Again, this is just an upper bound, since there are other reasons to use 2-of-2 multisigs in this way that have nothing to do with Lightning.

We identified 77,245 pairs of transactions that were potentially involved in opening and closing private channels, but likely has a high false positive rate. We thus developed a *tracing heuristic*, which follows the "peeling chain" [27] initiated at the opening and closing of public channels to identify any associated private channels.

Tracing heuristic. We next look not just at the properties of individual transactions, but also at the flow of bitcoins between transactions. In particular, we observed that it was common for users opening channels to do so in a "peeling chain" pattern [27]. This meant they would (1) use the change in a channel opening transaction to continue to create channels and (2) use the outputs in a channel closing transaction to open new channels. Furthermore, they would often (3) co-spend change with closing outputs; i.e., create a channel opening transaction in which the input addresses were the change from a previous opening transaction and the output from a previous closing one.

By systematically identifying these operations, we were able to link together channels that were opened or closed by the same Lightning node by following the peeling chain both forwards and backwards. Going backwards, we followed each input until we hit a transaction that did not seem to represent a channel opening or closure, according to our property heuristic. Going forwards, we identified the change address in a transaction, again using the property heuristic to identify the channel creation address and thus isolate the change address as the other output, and continued until we hit one of the following: (1) a transaction with no change output or one that was unspent, meaning we could not move forwards, or (2) a transaction that did not satisfy the property heuristic. We also did this for all of the outputs in a known channel closing transaction, to reflect the second pattern identified above.

We started with the 174,378 public channels identified in our LN dataset. By applying our tracing heuristic, we ended up with 27,386 additional channel opening transactions. Of these, there were 27,183 that fell within the same range of blocks as the transactions identified by our property heuristic.

Using the tracing heuristic, however, not only identified private channels but also allowed us to cluster together different channels (both public and private), according to the shared ownership of transactions within a peeling chain [27]. To this end, we first clustered together different channels according to their presence in the same peeling chain, and then looked at the public channels within each cluster and calculated the common participant, if any, across their endpoints. If there was a single common participant, then we could confidently tag them as the node responsible for opening all of these channels.

In order to find the other endpoint of each private channel, we followed the closing outputs of the channel's closing transaction, whenever applicable, leveraging the second and third observed patterns in the tracing heuristic. In particular, when a closing output was spent in order to open a new channel, we performed the same clustering operation as earlier. We failed to identify the second participant in each channel only when the channel was still open, the channel was closed but the closing output was still unspent, or the closing output was used for something other than Lightning.

Out of the 27,183 transactions we identified as representing the opening of private channels, we were able to identify both participants in 2,035 (7.5%), one participant in 21,557 (79.3%), and no participants in 3,591 (13.2%). Our identification method applies equally well, however, to public channels. We were able

to identify the opening participant for 155,202 (89.0%) public channels. Similarly, for the public channels that were already closed, which represent 185,860 closing outputs, we were able to associate 143,577 (77.25%) closing outputs with a specific participant.

4 Balance Discovery

Previous attacks designed to discover the balances associated with individual channels (as opposed to just their capacity) [16, 30, 43] exploited debug information as an *oracle*. In these attacks, an attacker opens a channel with a node and routes a fake payment hash, with some associated amount amt, through its other channels. Based on the error messages received and performing a binary search on amt (i.e., increasing amt if the payment went through and decreasing it if it failed), the attacker efficiently determines the exact balance of one side of the channel. In this section we perform a new *generic* attack on the LN testnet. As compared to previous attacks, our attacker must run two nodes rather than one. If error messages are removed or made generic in the future, however, our attack would continue to work whereas previous attacks would not.

The attack. In our attack, an attacker running nodes A and D needs to form a path $A \to B \to C \to D$, with the goal of finding the balance of the channel $B \to C$. This means our attacker needs to run two nodes, one with a channel with outgoing balance (A), and one with a channel with incoming balance (D). Creating 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 two main options for doing this. First, the attacker can open the channel $C \to D$ and fund it themselves, but assign the balance to C rather than to D (this is called funding the "remote balance"). This presents the risk, however, that C will immediately close the channel and take all of its funds. We call this approach unassisted channel opening. The second option is to use a *liquidity provider* (e.g., Bitrefill[8] or LNBIG[9]), which is a service that sells channels with incoming balance.[10] We call this assisted channel opening.

Once the attacker has created the channels $A \to B$ and $C \to D$, they route a random payment hash H to D, via B and C, with some associated amount amt. If D receives H, this means the channel from B to C had sufficient balance to route a payment of amount amt. If D did not receive H after some timeout, the attacker can assume the payment failed, meaning amt exceeded the balance from B to C. Either way, the attacker can (as in previous attacks) repeat the process using a binary search on amt. Eventually, the attacker discovers the balance of the channel as the maximum value for which D successfully receives H.

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

[9] http://lnbig.com/

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

To a certain extent, this attack generalizes 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. In the event of a payment failure though, the current C-lightning and LND clients return an *error index*, which is the position of the node at which the payment failed. This means that an attacker would know exactly where a payment failed along a longer path. We chose not to use this index when implementing our attack, in order to keep it fully generic and to test just the basic version, but leave an attack that does use this index as interesting future research.

Attack results. We performed this attack on `testnet` on September 3 2020. We ran two LN nodes and funded all our channels (unassisted), both locally and remotely, which required a slight modification of the client (as fully funding a remote channel is restricted by default). We opened channels with every accessible node in the network. At the time of the attack there were 3,159 nodes and 9,136 channels, of which we were able to connect to 103 nodes and attack 1,017 channels. We were not able to connect to a majority of the overall nodes, which 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 refused to open a channel.

Of these 1,017 channels, we determined the balance of 568. Many (65%) of the channels were fairly one-sided, meaning the balance of the attacked party was 70% or more of the total capacity. We received a variety of errors for the channels where we were unsuccessful, such as *TemporaryChannelFailure*, or we timed out as the client took more than 30 seconds to return a response.

We did not carry out the attack on mainnet due to cost and ethical considerations, but believe it likely that the attack would perform better there. This is because there is no cost for forgetting to close open channels on testnet or maintain a node, whereas on mainnet a user is incentivized by an opportunity cost (from fees) to ensure a node is maintained and its channels are active.

Attacker cost. In our experiment we used testnet coins, which are of essentially no value, so the monetary cost for us to perform this attack was negligible. To understand the practical limitations of this attack, however, we estimate the minimum cost on mainnet. When creating the outgoing channel $A \to B$, the attacker must pay for the opening and closing transaction fees on the Bitcoin blockchain. At the time of our attack, this was 0.00043 BTC per transaction. They must also remotely fund the recipient node with enough *reserve satoshis* to allow the forwarding of high payments, which at present are 1% of the channel capacity. To create the incoming channel $C \to D$, the attacker can use liquidity providers like Bitrefill, who at the time of writing allow users to buy channels with 0.16 BTC incoming capacity for 0.002604 BTC.

Purchasing the cheapest incoming liquidity available today would cost the attacker 0.00086 BTC and 0.005 BTC on hold, enabling routes to 4,811 channels

(with a total capacity of 45 BTC). This would require opening 2,191 channels with a maximum channel capacity of 0.04998769 BTC. In total, this would require the attacker to spend 1.097 BTC and put 109.53 BTC on hold.

5 Path Discovery

We now describe how an *honest-but-curious* intermediate node involved in routing the payment can infer information regarding its path, and in particular can identify the sender and recipient of the payment. Our strategy is similar to a passive variant of a predecessor attack [46] proposed against the Crowds [36] anonymous communication network. Our strategy can be further extended by analyzing the sparse network connectivity and limited number of potential paths due to channel capacity.

In contrast to previous work [5], we consider not only single-hop routes but also routes with multiple intermediate nodes. The only assumption we make about the adversary's intermediate node is that it keeps its channel balanced, which can be easily done in practice.

We define $\Pr_S$ and $\Pr_R$ as the probability that the adversary successfully discovers, respectively, the sender and recipient in a payment. Following our notation, Béres et al. claim, based on their own simulated results, that $\Pr_S = \Pr_R$ ranges from 0.17 to 0.37 depending on parameters used in their 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).

The strategy of our honest-but-curious adversary is simple: they always guess that their immediate predecessor is the sender. In other words, if we define H as the adversary's position along the path, they always assume that $H = 1$. Similarly, they always guess that their immediate successor is the recipient. We focus on the probability of successfully guessing the sender; the probability of succesfully guessing the recipient can be computed in an analogous way.

Successful payments. We start by analyzing the success probability of this adversary 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 adversary'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 adversary is the only intermediate node in this case. Hence, $\Pr[L = 3 \mid \mathsf{succ}]$ is a lower bound on $\Pr_S$.

To consider the overall probability, we focus on the conditional probabilities $\Pr[H = 1 \mid L = \ell, \mathsf{succ}]$. If all nodes form a clique,[11] 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 an adversary could choose fees to match its neighbors as closely as possible.) In this case then, the probability that $H = 1$ is just $1/(\ell - 2)$.

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 adversary 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 adversary 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

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

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 adversary'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

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

5.1 Lightning Network simulator

In order to investigate the success of this on-path adversary, we need measurements that it would require significant resources to obtain from the live network, such as the average path length for a payment. Given the financial and

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

ethical concerns this would raise, we make the same decision as in previous work [5, 9, 10, 13, 49] to develop a Lightning network simulator to perform our analysis. We implemented our simulator in 2,624 lines of Python 3 and will release it as open-source software.

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 3.1, 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. Nodes may publish an IPv4, IPv6 or .onion address, or some combination of these. 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.[12] 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.[13]

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. [44], 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 [47] and the networkx Dijkstra's SPF algorithm.[14]

Software versions. 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.

Payment parameters Our first parameter, $\mathsf{t}_{\mathsf{pay}}$, represents the total daily number of payments happening in LN. For this, we use an estimate from 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

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

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

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

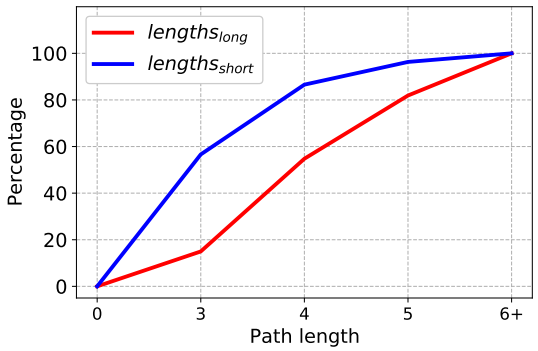

Fig. 1: Average path length.

going through the network is 1000-1500 per day, but this does not take into account the payments performed via direct channels. Given this estimation, we use two values for $\mathsf{t_{pay}}$: 1000, representing a slight underestimate of today's volume, and 10,000, representing a potential estimate for the future of LN.

We also define as endpoints the parameter that determines the sender and the recipient of a payment. We define two values for this parameter: *uniform*, which means that the payment participants are chosen uniformly at random, and *weighted*, which means the participants are chosen randomly according to a weighted distribution that takes into account their number of direct channels (i.e., their degree). Similarly, we use values to determine the values of payments. When values is *cheap*, the payment value is the smallest value the sender can perform, given its current balances. When values is *expensive*, the payment value is the biggest value the sender can send.

5.2 Simulation results

Given the parameters $\mathsf{t_{pay}}$, endpoints, and values, we ran two simulation instances, with the goal of finding the worst-case and best-case scenarios for the on-path adversary. Based on the respective probabilities for $\Pr_S^{\mathsf{succ}}$ and $\Pr_S^{\mathsf{fail}}$, we can see that the worst case is when the path is long and the payment is likely to succeed, while the best case is when the path is short and the payment is likely to fail. Since the total volume $\mathsf{t_{pay}}$ does not affect the path length, we use $\mathsf{t_{pay}} = 1000$ for both instances. Each simulation instance was run using the network and node parameters scraped on September 1, 2020.

In our first simulation, $\mathsf{lengths_{long}}$, our goal was to capture the adversary's worst case. This meant we chose endpoints = *uniform*, so that the choice of sender and receiver was not biased by connectivity, and thus paths were not short due to their potentially high connectivity. Similarly, we chose values = *cheap* to minimize the probability of having a payment fail. For our second simulation, $\mathsf{lengths_{short}}$, our goal was to capture the adversary's best case, so we

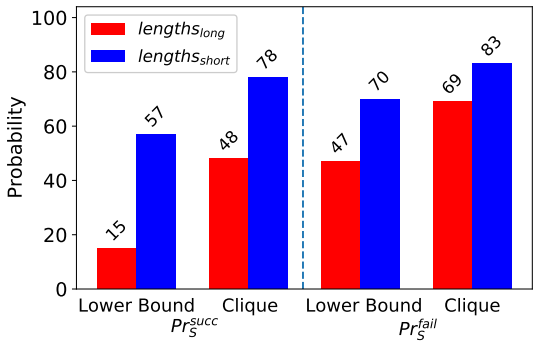

Fig. 2: Probability of correctly identifying the sender given successful and failed payment. For detailed simulation settings of $\mathsf{lengths}_{\mathsf{long}}$ and $\mathsf{lengths}_{\mathsf{short}}$ see Section 5.2

chose endpoints = *weighted* to ensure highly connected nodes were picked more often and thus paths were shorter. We also chose values = *expensive*, leading to many balance failures.

As shown in Figure 1, even when we attempted to maximize the path length in $\mathsf{lengths}_{\mathsf{long}}$, 14.98% of paths still consist of only one hop. In $\mathsf{lengths}_{\mathsf{short}}$, 56.65% of paths consisted of a single hop. This interval agrees with recent research, which argues that 17-37% of paths have only one intermediate node [5]. The main reason the paths are short even in $\mathsf{lengths}_{\mathsf{long}}$ is that the network topology and the client path finding algorithm have a much larger effect on the path length than endpoints or values.

Beyond the results in Figure 1, running our simulator enabled us to estimate the probabilities $\Pr[L = \ell]$ for $3 \leq \ell \leq 20$ for both the best- and worst-case scenario for the adversary. We now use those results to compute the probabilities $\Pr_S^{\mathsf{succ}}$ and $\Pr_S^{\mathsf{fail}}$ for the case where our adversary is succesful only when it is impossible to be wrong (LowerBound) as well as in the case of a clique topology (clique), as shown in Figure 2. Here the clique topology is the worst possible topology for the adversary, since a less complete topology would allow the adversary to rule out nodes that cannot be involved in the payment and thus increase their confidence.

$\Pr_S^{\mathsf{succ}}$ is bounded from below when $L = 3$, since in that case the adversary can never be wrong. Similarly, $\Pr_S^{\mathsf{fail}}$ is bounded from below when $L = 4$. In the case of a successful payment, the lower bound on $\Pr_S^{\mathsf{succ}}$ ranges from 15% ($\mathsf{lengths}_{\mathsf{long}}$) to 57% ($\mathsf{lengths}_{\mathsf{short}}$). On the other hand, the lower bound of $\Pr_S^{\mathsf{fail}}$ increases with the percentage of unsuccessful attempts, up to 83% ($\mathsf{lengths}_{\mathsf{short}}$), which is significantly higher than any previously recorded experiment. This is also not just a theoretical result: according to recent measurements, 34% of payments fail on the first try [1].

Our measurements show that even an adversary following an extremely simple strategy can have a high probability of inferring the sender of a payment routed through their node, especially in the case in which the payment fails. This is likely due to LN's highly centralized topology, which means paths are short and often involve the same intermediate nodes, as well as the fact that clients are designed to find the cheapest—and thus shortest—paths. Without changes in the client or the network topology, it is thus likely that intermediate nodes will continue to be able to violate on-path relationship anonymity.

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 balance discovery attack described in Section 4, 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 $G_{\mathsf{diff}} = S_{t+\tau} - S_t$ 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 C along the path $A \to B \to C$, we do not want to count it as two (equal-sized) payments of the form A to B and B to 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 often holds when the interval between snapshots is relatively short. 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

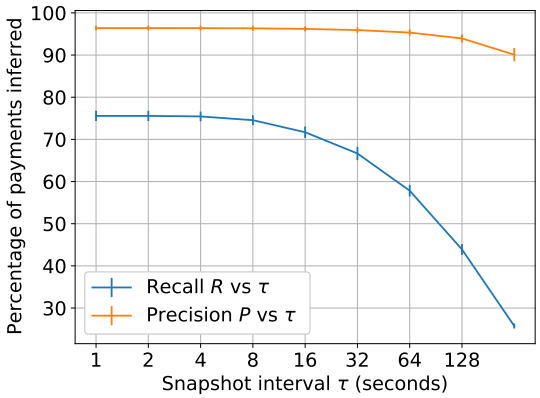

Fig. 3: The precision and recall of our payment discovery attack, based on the snapshot interval τ (in log scale). The error bars show 95% confidence intervals over five simulation runs.

weight amt can be merged.

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

We then remove this path from G_{diff} and continue with another edge. Asymptotically the running time 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 to B and B to C occurring end-to-end. Second, our algorithm does not attempt to 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 4, balance discovery may fail for some (or many) channels in the network. 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 two 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 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), although we also present in Appendix C an analysis of the attacker's recall based on the number of channels it opens (n).

To answer these questions we leverage the simulator we developed in Section 5.1, and extend it to include the balance discovery attack from Section 4. Due to the fact that 98% of the errors in this attack were because a node was not online or did not participate in any payments, we set a 0.05 probability of it failing on a functional channel in which both nodes are online. In keeping with the discussion in Section 5.1, we use $\mathsf{t}_{\mathsf{pay}} = 2000$ as the total number of payments per day and sample the senders and recipients randomly from all nodes in the network. In terms of payment value, our simulation is a pessimistic scenario where the payment amounts are very small (1000 satoshis on average) but fluctuate uniformly within a small range around this average (± 10 satoshis). This is pessimistic because it is close to the worst-case scenario, in which all payments have identical amounts, but two factors make it likely that real-world payments would have much greater variation: (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 is volatile.

6.3 Simulated attack results

In order to figure out the effect of the snapshot interval τ on P and R, we take balance inference snapshots of the entire network for varying time intervals, ranging from $\tau = 1$ second to $\tau = 2^8$ seconds. Each time, we run the simulator for a period of 30 days, amounting to 60,000 payments in total. Figure 3 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 $\tau = 32$ seconds, the attacker has a recall R of 66%, which increases slowly to 74.1% for $\tau = 1$ second. With a realistic minimum of $\tau = 30$ seconds, which is the time it took us and others to run the balance discovery attack on a single channel [16]), the attacker has a recall of more than 67%. Because of our final filtering step in our discovery algorithm, we have a precision P very close to 95% for smaller values of τ.

7 Conclusions

In this paper, we systematically explored the main privacy properties of the Lightning Network and showed that, at least in its existing state, each property is susceptible to attack. Unlike previous work that demonstrated 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.

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 anoymity 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, 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.
28. 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.
29. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.
30. Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks, 2020.
31. Mariusz Nowostawski and Jardar Tøn. Evaluating methods for the identification of off-chain transactions in the Lightning Network. *Applied Sciences*, 2019.
32. Cristina Pérez-Solà, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning network channels. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 245–263, Cham, 2020. Springer International Publishing.

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. 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. Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.
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. Matteo Romiti, Friedhelm Victor, Pedro Moreno-Sanchez, Bernhard Haslhofer, and Matteo Maffei. Cross-layer deanonymization methods in the lightning protocol, 2020.
39. Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*. Springer, 2013.
40. István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. Topological analysis of Bitcoin's Lightning Network. *arXiv:1901.04972*, 2019.
41. 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.
42. Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. A quantitative analysis of security, anonymity and scalability for the lightning network. Cryptology ePrint Archive, Report 2020/303, 2020. https://eprint.iacr.org/2020/303.
43. Sergei Tikhomirov, Rene Pickhardt, Alex Biryukov, and Mariusz Nowostawski. Probing channel balances in the lightning network, 2020.
44. Saar Tochner, Stefan Schmid, and Aviv Zohar. Hijacking routes in payment channel networks: A predictability tradeoff. *arXiv:1909.06890*, 2019.
45. Shira Werman and Aviv Zohar. Avoiding deadlocks in payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 175–187. Springer, 2018.
46. Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):489–522, 2004.
47. 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.
48. 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.
49. 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.

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.[15] 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, 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.

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

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.

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.

B Node Throughput

Given the parameters $\mathsf{t}_{\mathsf{pay}}$, $\mathsf{endpoints}$, and values, we ran two simulation instances, with the goal of finding the upper and lower bound for the number of payments a node forwards per day. Each simulation instance was run using the network and node parameters scraped on September 1, 2020. We define the throughput T_x as the number of payments a node x forwards per day.

In our first simulation instance, $\mathsf{throughput}_{\mathsf{big}}$, we aim to find the upper bound of T_x. For this, we use $\mathsf{t}_{\mathsf{pay}} = 10000$ since the throughput per node increases if the total number of payments increases. We also use $\mathsf{endpoints} = \textit{weighted}$, which means that well-connected nodes will be picked more frequently as payment

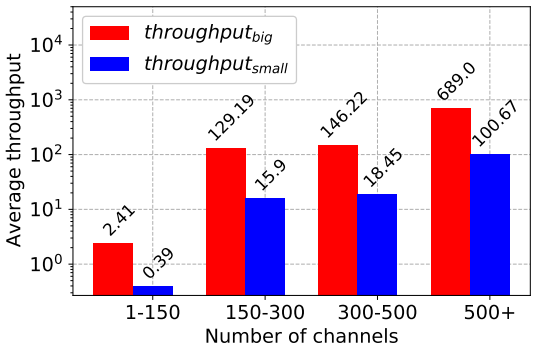

Fig. 4: Average throughput

participants. This will increase T_x for that set of nodes. Similarly, we use $\mathsf{values} = \textit{expensive}$, as this again increases T_x for nodes that are part of channels with high capacities. Using these parameters thus maximizes T_x for nodes with high degree and high capacity, which we refer to as *routers*.

In contrast, for $\mathsf{throughput}_{\mathsf{small}}$, we aim to find the lower bound of T_x. We thus use $\mathsf{t}_{\mathsf{pay}} = 1,000$ and $\mathsf{endpoints} = \textit{uniform}$, as the throughput per node is lowest when the (low) traffic is spread evenly across them. Similarly, we use $\mathsf{values} = \textit{cheap}$ to evenly distribute the load across all channels, even those with low capacity.

After running the simulations, 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. The results are in Figure 4, and show that—in both scenarios—the number of channels a node has directly influences the number of payments it forwards.

In particular, for $\mathsf{t}_{\mathsf{pay}} = 2000$ 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 $\mathsf{t}_{\mathsf{pay}} = 4500$, the node with the maximum number of channels forwarded 1,485 payments, which translates to a throughput of 0.0172 payments per second.

On-path discovery observation. As mentioned in Section 1, a node in a path should not learn anything about the other participating nodes, and even two colluding nodes should not be able to infer that they belong in the same path. Malavolta et al. [25] observed that there is a common identifier in a path (the hash of the secret $\mathbf{x}$) that would make this trivially possible, but overcame this by blinding the common identifier in each hop.

In practice, however, as we saw above, even in $\mathsf{throughput}_{\mathsf{big}}$, where we maximized the throughput, it still remained low; i.e., 0.48 payments per minute for the busiest nodes. Given this, if two colluding nodes are asked to forward a payment of approximately the same value at approximately the same time (depending on the TCP delays of the nodes in between), it trivial to infer that they

are forwarding the same payment, even with blinded identifiers. Anonymous communication systems have tackled the same problem of low bandwidth by adding cover traffic, so LN could similarly add dummy payments to make such correlation attacks harder to perform. We leave a more thorough exploration of this countermeasure as future work.

C Effect of Attacker Budget on Recall R

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 4, we believe to be a feasible minimum). In the following discussion, we refer to our balance discovery attack as a *generic* attack, and to previous attacks [16, 30, 43] that rely on error messages as *oracle-aided* attacks.

Figure 5 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, and 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 56.3% 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.

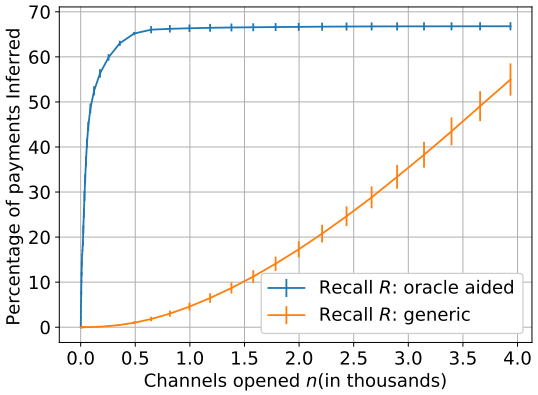

Fig. 5: Recall R as a function of the number of channels required. The error bars indicate a 95% confidence interval over five simulation runs.