Analysis and Probing of Parallel Channels in the Lightning Network

Alex Biryukov
University of Luxembourg
alex.biryukov@uni.lu

Gleb Naumenko
thelab31.xyz
naumenko.gs@gmail.com

Sergei Tikhomirov
University of Luxembourg
sergey.s.tikhomirov@gmail.com

Abstract

The Lightning Network (LN) is a prominent scalability solution for Bitcoin that allows for low-latency off-chain payments through a network of payment channels. LN users lock bitcoins into collaboratively owned addresses and redistribute the ownership of these funds without confirming each transfer on-chain. The LN introduces new privacy challenges.

In this paper, we focus on channel balance probing. We propose a new model of the LN that accounts for parallel and unidirectional channels, which has not been done in prior work. We describe a probing algorithm that accurately updates the attacker's balance estimates without the need to directly connect to victims. We introduce an uncertainty-based metric to measure the attacker's information gain.

We implement the first probing-focused LN simulator and suggest several countermeasures against general probing (implemented considering parallel channels). We evaluate these techniques using the simulator, as well as experiments on the real network.

According to our simulations, an attacker can infer up to 80% information regarding channel balances spending ≈ 20 seconds per channel. The suggested countermeasures limit the attacker's gain at 30%, while also increasing the attack time by 2-4x.

In addition, we describe sophisticated attack techniques that combine fee-probing and channel jamming to get precise access to individual channel balances inside a hop, and test them against the real network.

Finally, we discuss payment flows and their concealment.

1 Introduction

Bitcoin is a peer-to-peer electronic cash system [15]. The requirement for being fully functional on commodity hardware limits Bitcoin's transaction throughput to approximately 7 transactions per second. Among various scaling approaches, second-layer (L2) protocols [7] are one of the most promising. The Lightning Network [20], launched in 2018, is an L2 protocol on top of Bitcoin. It is a network of payment channels. A payment channel is a trust-minimized two-party protocol that allows for low-latency and low-cost payments [10] with minimal interaction with the base layer (Bitcoin).

Privacy is a crucial requirement for cryptocurrency protocols. Bitcoin's privacy has been shown to have weaknesses [1, 13]. One of the main privacy challenges of Bitcoin is a consequence of its strength – public verifiability. Since any user must be able to verify all transactions that have ever happened, all transactions are in the plain sight of an attacker.

In the LN, the full payment details are only known to the sender and the receiver. This makes the LN a good candidate for improving not only Bitcoin's scalability but also privacy. In practice, however, the LN introduces new privacy attack vectors [2, 16, 24, 25]. In particular, an attacker can estimate channel balances of honest users in a probing attack [11, 12, 33, 36]. The key element of probing is sending fake payments (probes) and narrowing balance estimates via binary search depending on the error type. The attacker bears only minimal cost (i.e., on-chain fees for opening channels) and does not pay LN fees, as failed payment attempts are free. Probing can be used as a building block to spy on payments or node balances, or to deanonymize LN users.

Previous work has demonstrated that probing attacks are cheap, fast and rather non-sophisticated. At the same time, the prior art has introduced a number of impractical assumptions:

- *The attacker connects directly to a victim channel*. The victim may not allow this, moreover, this requirement makes the attack poorly scalable.
- *There is at most one channel between each pair of nodes*. The LN specification explicitly allows multiple channels between the same pair of nodes. This feature is widely used in practice (45% of capacity is held in such channels) and causes problems in channel probing[1].

[1] The paper [16] writes: "Our tool failed to produce accurate results in this scenario [...] It is however perfectly reasonable to have multiple channels between two nodes [...] We expect this to be the predominant form of retrospectively increasing potential payment flow between nodes and further research on how to deal with this complication would be highly appreciated."

- *Channels forward payments in both directions.* In practice, channel counterparties may disallow forwarding from their side of the channel, thus preventing probing from that direction.

Building on top of the most efficient probing technique [33], our approach expands it with a notion of parallel channels and channel directions. In particular, we carefully model **multi-hop paths**, **parallel channels**, and **channel directions** by representing a network as a directed multigraph. We track where each error comes from and update the balance estimates accordingly. We do not assume that nodes can only be connected with one channel, and develop a new hop-based (as opposed to single-channel-based) model of the network. Finally, we probe each channel from both directions if possible, increasing efficiency; unlike previous works, and reflecting the properties of the real network, we do not assume that channels are always bidirectional.

To assess the effectiveness of our probing method, we propose a novel way to measure the attacker's information gain. We develop the first probing-based LN simulator that implements the hop model, and measure probing effectiveness in a simulated environment under different conditions. Our simulations are based on a real network snapshot.

We describe how two additional attacks (channel jamming and privacy leakage through channel policies) allow for probing parallel channels, which would enable a nearly full extraction of channel balance information. We verify these ideas in controlled experiments against real LN implementations.

Finally, we propose three types of countermeasures and demonstrate how their combinations make the attacks 2-4x longer in time (i.e., reduce their efficiency) and also bound the attacker's knowledge to 30% of the initial uncertainty (instead of 80% without countermeasures).

The rest of this paper is organized as follows. In Section 2, we provide the necessary background about Bitcoin, Lightning, and relevant attacks. Section 3 introduces a novel hop-based model of the LN and a probing algorithm that accurately updates balance estimates to fully reflect probing outcomes. In Section 4, we define the uncertainty-based metrics to assess the attacker's information gain and attack cost (in terms of time). In Section 5, we evaluate our probing technique in a simulated network based on a real LN snapshot. We also assess three countermeasures: reporting delay, error spoofing, and deliberate forwarding failure. Section 6 describes how the attacker can use channel jamming to improve probing of multi-channel hops. Section 7 briefly describes how to adjust probing to spy on payments. Section 8 suggests advanced countermeasures. We discuss usability-privacy trade-offs and future work in Section 9, review related work in Section 10, and conclude in Section 11.

2 Background

2.1 Bitcoin

Bitcoin is a distributed system for transactions without a trusted third party. Every transaction changes ownership over a certain amount of coins, or satoshis – the minimal unit of payment (one bitcoin equals 100 million satoshis).

The current ownership of coins is encoded in unspent transaction outputs (UTXOs). The most simple ownership rule is a possession of a private key for a corresponding public key. To spend a UTXO, its owner signs a transaction that proves their ownership of the private key in question and transfers the ownership to the receiver's public key. All users can then verify that the sender indeed had access to the private key. To make this verification practical even on commodity hardware, the Bitcoin protocol limits the block size. Considering the constant average time between blocks (10 minutes), Bitcoin can confirm around 7 transactions per second.

2.2 Lightning architecture

To scale Bitcoin beyond this limit, the LN was introduced. The LN is a peer-to-peer network of nodes, some of which share one or multiple payment channels (referred to simply as channels throughout this work). Nodes and channels are uniquely identified by their IDs. Compared to Bitcoin, the LN allows for low-latency payments at the cost of relaxing certain security assumptions. In particular, LN users must monitor the Bitcoin's blockchain and dispute fraud attempts. It is sufficient to re-connect once every few hours or days, depending on channel parameters. The monitoring may be outsourced to specialized services called watchtowers.

Besides payments between channel parties, the LN supports routing payments across multiple hops. This allows any LN user to pay any other user without establishing a channel with them (assuming the receiving user has sufficient incoming balance). In a multi-hop payment, channel balances along the path are shifted simultaneously. The atomicity of this shift is enforced cryptographically.

The development of the LN is guided by a set of standards [3] followed by multiple development teams. Major LN implementations include LND, C-LIGHTNING, and ECLAIR.

2.2.1 Channels in the LN

A channel is a trust-minimized cryptographic protocol that allows two parties to maintain the distribution of coins initially committed to a jointly controlled UTXO. The channel state is encoded in a set of pre-signed commitment transactions. A channel operates in three stages: opening, normal operation, and closing. To open a channel, Alice and Bob lock coins in a cooperatively owned UTXO, establishing the initial balance distribution. To make a payment, the parties negotiate and exchange a new set of commitment transactions. Critically, this

action makes the new channel state enforceable by provably invalidating the old state [7]. At any time, any party can close the channel on-chain and withdraw their coins as defined by the latest channel state.

The total number of coins committed to a channel is called *capacity* and is constant throughout the life of a channel. The amount of coins currently owned by each party is called a *balance* and changes as payments are made. The two balances sum up to the capacity (without accounting for in-flight payments and fees, see also Section 6.2.2).

According to the LN specification [3], a pair of LN nodes may share multiple channels. We refer to such channels as *parallel*. They constitute 21% of all channels and hold 45% of total network capacity (see Section 5.2 for more details).

2.2.2 Multi-hop payments

As mentioned earlier, LN payments may be *routed* through a path of channels connecting the sender to the receiver. Routing is atomic: a payment cannot *partially* fail along the way.

The sender determines the route based on a local view of the network graph[2]. The graph information is distributed in gossip messages and describes the availability, capacities, and fee policies of channels. Users may also establish unannounced channels, hidden from the rest of the network. To compensate for resource usage, routing nodes normally charge a fee.

To hide payment details from routing nodes, LN payments are onion-routed. Intermediary nodes know the payment amount but are only aware of the previous and the next node in the route.

Note that the sender only knows the capacities of remote channels, whereas their forwarding ability is determined by their balances. Therefore, a sender cannot reliably choose a suitable route, and LN payments often fail due to insufficient balance at an intermediary hop. Normally, a sender makes multiple attempts before the payment eventually succeeds.

Upon receiving a payment, the receiver finalizes it by propagating the *payment secret* along the path back to the sender. The payment secret is the preimage of the *payment hash* that identifies the payment[3]. If an error occurs, the erring node sends an error message back to the sender along the path. Therefore, the sender always knows whether an error occurred and if so – at which node it happened.

2.2.3 Parallel channels and non-strict forwarding

Even though the sender is solely responsible for path-finding, the protocol only enforces an ordered list of nodes through which a payment must flow.

In practice, two nodes in the LN might be connected by multiple parallel channel. Parallel channels might be useful for security flexibility: a routing node might have a smaller channel with relaxed security parameters (e.g., `cltv_expiry_delta`) for risky payments and a large channel with strict parameters. These different channels might also have different fee policies.

If several (parallel) channels are suitable for forwarding a payment, the protocol does not enforce which one should be used. A routing node may choose a channel using different strategies. Currently, all LN implementations use the *simple best-effort* approach, i.e., they do forward the payment if at least one of their channels can do so. No matter which channel is chosen, a routing node takes all the fees allocated for it. We discuss how alternative approaches to channel selection might prevent probing in Section 8.

The three major LN implementations use different channel selection strategies. ECLAIR uses the channel with the smallest capacity (among the channels with the same capacity, it prefers the one with a smaller balance)[4]. LND chooses a random channel[5]. C-LIGHTNING has no channel selection as it does not support parallel channels.

2.2.4 Unidirectional channels

All channels in the Lightning Network are bidirectional, meaning they might facilitate payments (both initiated by the channel counterparties and other nodes in the network via multi-hop) in both directions.

At the same time, nodes sometimes disable one of the channel directions for forwarding multi-hop payments. The responsible channel co-owner may do so by signing a `channel_update` message and gossiping it across the network.

Forwarding is usually disabled due to the loss of connectivity, or prior to channel settlement, although other scenarios are also possible. Note that for privacy reasons, private channels (see Section 2.2.5) might be preferred, because they achieve the same non-forwarding behavior, but are also not even known to the network.

In our work, we call these channels *unidirectional* (not to be confused with the unidirectional channel construction, which is not used by the LN). Unidirectional channels constitute 9% of the channel count and control 7% of the LN's total capacity.

2.2.5 Private channels

Private channels are not announced to the network. These channels constitute a non-negligible part of the network. Mobile clients often exclusively use private channels to LN hubs

[2] Unless trampoline or rendezvous routing is used [30, 39].

[3] We refer the reader to [3] for details on hash time-locked contracts (HTLCs) that are the basis of the LN protocol. The protocol does not forbid using the same hash for different payments, which is used in multi-part payments (MPP).

[4] https://github.com/ACINQ/eclair/blob/5f9d0d/eclair-core/src/main/scala/fr/acinq/eclair/payment/relay/ChannelRelay.scala#L199

[5] https://github.com/lightningnetwork/lnd/blob/f98a3c/htlcswitch/switch.go#L1091

to preserve their privacy.

If a node does not announce its channels, only its channel counterparties would know it exists. However, it can still receive payments from anywhere if it assists the payee with so-called routing hints.

Private channels are of interest for us because they can be used to protect public channels from being probed. We discuss this in more detail in Section 8.5.

2.3 Attacks on Lightning

The attacker can gain knowledge regarding the activities of other users using several attack vectors. In this work, we focus on third party channel / node balance inference via probing. Balance inference may be interesting for an attacker in several contexts. In particular, it allows for estimating the victim's capital and is a building block for other attacks (payment inference, payment flow inference), encompassing several scenarios. For instance, Mallory may want to learn how much Alice pays to Bob or, more broadly, detect all Alice's payments, including their amount, time, and destination.

2.3.1 Probing

The possibility of probing is a consequence of payment channel networks design. An LN payment can only be forwarded through a channel if its balance in the required direction is greater or equal to the payment amount. This leads to information leakage: a channel reacts to forwarding requests differently depending on its balance. An attacker exploits this leak by forwarding fake payments (probes) and observing resulting errors.

The attack proceeds as follows. First, the attacker defines the target channel and prepares a route that terminates at this channel. Since payments may fail along the route, shorter routes are better. Then, the attacker sends a probe through the route. The probe either reaches the destination or fails due to insufficient balance at an intermediary channel[6]. To avoid paying fees in the former case, the attacker uses a random value instead of a payment hash that is normally generated by the receiver. The receiver does not know the preimage of a random hash and responds with a specific error, according to the protocol. The attacker thus learns whether the balance of the failing channel is above or below the probe amount, and that all channels before it have succeeded. Repeating this process, the attacker can in principle infer the target channel balance with arbitrarily high accuracy. In practice, the attacker might terminate the probing after reducing the window between the balance estimates to an acceptable size. The initial estimates for an unknown balance b are $[0, c]$, where c is the channel capacity. The optimal choice of probe amounts (to minimize the number of probes) depends on the distribution of channel balances. Assuming uniform distribution, binary search is the optimal strategy.

2.3.2 Channel jamming

As we will show in subsequent chapters, the attacker can improve the effectiveness of probing by combining it with another type of attack – channel jamming. Jamming is a family of denial-of-service attacks on LN channels. An attacker initiates a payment along a route that goes through a target channel and terminates at another attacker's node. Then, an attacker refuses to finalize the payment, locking the funds along the entire route until the timelocks expire. There is no simple solution to this problem. An intuitive timeout would not work, because then an attacker just repeats the payment over and over again. After the attack is done, an attacker fails the payment (to release their coins without paying fees).

There are two types of jamming. In capacity-based jamming, an attacker jams a channel (fully or partially) by initiating a payment of a given (presumably high) value[7]. In slot-based jamming, an attacker blocks a channel completely by sending a series of small payments and reaching the limit of in-flight payments (by default, it is 483 in each direction).

3 Network model for probing

Parallel channels account for 45% of network capacity and are common between highly-connected and well-capitalized nodes ("hubs"). We expect them to be heavily used in payment forwarding, so they should not be ignored in the context of channel probing.

Parallel channels prevent the attacker from learning channel balances with arbitrary precision using simple binary search. With a trivial approach to probing and current payment forwarding protocols, the attacker can only reliably find the *maximum* balance among a set of parallel channels, but cannot argue about the exact balance of each channel. To better understand these limitations and suggest attack strategies and countermeasures, we propose an alternative model of the Lightning Network, which better reflects the reality. Our model includes two graphs: the LN graph that models the LN itself, and the hop graph that models the attacker's knowledge about the network.

[6] Other errors are possible, but we omit them for simplicity. It is also possible that the same error we are looking at is caused by limits on in-flight payments, which either resolves to the same case (if those in-flight payments succeed), or mislead an attacker (if those payments "roll back"). We leave the latter considerations for future work.

[7] An attacker might have also just sent this payment to themselves without withholding and achieve the same effect, but this would cost paying fees. Sending instead of withholding may make sense because it eases the requirement of the overall capital to be locked, in the case of jamming multiple parallel channels simultaneously.

3.1 LN graph model

We represent the LN as a directed multigraph (i.e., parallel edges are allowed), in which vertices represent nodes and edges represent channels between them.

We define the *direction* of a channel to take two values: direction `dir0` goes from the node with the alphanumerically smaller ID, and `dir1` is the opposite. Analogously, we refer to the balance b at the alphanumerically smaller node ID as balance in direction `dir0`. This notation is consistent with the LN specifications. The balance in direction `dir1` can be derived as $\tilde{b} = c - b$, where c is the channel capacity.

3.2 Hop graph

We model the attacker's view of the LN as a directed multigraph H. Each edge of H represents a *hop* – a set of parallel channels between two nodes. The channels in the hop fully determine its properties. In particular, a hop with n channels can forward at most $b_{max} = max_{i\in[1,n]}(b_i)$ in `dir0` and $\tilde{b}_{max} = max_{i\in[1,n]}(\tilde{b}_i)$ in `dir1`.

In some cases (for example, in a single-channel hop), probes reveal information about individual channels. In general, however, probes only reveal information about a hop as a whole. To accurately reflect this distinction, the attacker maintains upper and lower bounds for each channel, as well as bounds for the hop as a whole. We refer to channel-level bounds for the i-th channel in a hop as b_i^l and b_i^h, and to hop-level bounds as b_{max}^l and b_{max}^h.

The upper bound b_{max}^h shows how much a hop can forward in the best case. It can be calculated from the channel bounds as $max_{i\in[1,n]}(b_i^h)$. The lower bound b_{max}^l shows how much a hop is guaranteed to be able to forward. It cannot be calculated from balance bounds alone: a successful probe of amount a tells the attacker that at least *one of the channels* can forward a, but does not tell which channels can do it. As a result, the attacker can only update the hop-level bounds. In contrast, if a probe of amount a fails, the attacker can update the upper bounds for individual channels, as no channel can forward a. Therefore, b_{max}^l is the only field assigned per-hop and not per individual channels.

The following invariants hold for both directions for any hop with n channels (only the channels enabled in the required direction are considered):

- $\forall i \in [1,n], b_i^l \le b_i \le b_i^h \le c_i$;
- $b_{max}^l \ge max_{i\in[1,n]}(b_i^l)$;
- $b_{max}^h = max_{i\in[1,n]}(b_i^h)$.

Consider an example of a multi-channel hop in Table 1. Taking channel directions into account, the hop described in Table 1 can forward $b_{max} = max(42,81) = 81$ in `dir0` and $\tilde{b}_{max} = max(119,126) = 126$ in `dir1`.

cid	c	b	$\tilde{b}$	dir0 enabled	dir1 enabled
1	100	42	58	yes	no
2	200	81	119	yes	yes
3	300	174	126	no	yes

Table 1: Example of three parallel channels.

The attacker defines channel- and hop-level bounds for each hop and direction in its local hop graph. In the hop defined above, a priori channel bounds are: $\forall i \in [1,n], b_i^l = 0, b_i^h = c_i$. The attacker only maintains channel bounds in `dir0`, as the bounds in `dir1` are automatically derived as $\tilde{b}^l = c - b^h$ and $\tilde{b}^h = c - b^l$. A priori hop-level bounds are $b_{max}^l = \tilde{b}_{max}^l = 0$ and $b_{max}^h = max(c_i)$ for channels enabled in `dir0`, that is, $b_{max}^h = max(100,200) = 200$. Analogously, $\tilde{b}_{max}^h = max(200,300) = 300$.

3.3 Updating the bounds

Starting from the initial bounds, the attacker performs a series of probes toward each target hop with the goal to shrink the interval between the balance bounds. Recall that the attacker always knows where each probe fails. A probe may gain extra knowledge for a target hop as a whole or for individual channels. If a probe has not reached the target hop, the attacker tries another route. The attacker stops probing as soon as the precision is achieved, as defined using information-theoretic metrics (see Section 4).

We now describe the update rules that allow the attacker to precisely reflect the information from probes. Consider a probe in direction `dir0` with amount a being forwarded through an n-channel hop. The case of `dir1` is analogous. The channels have capacities $c_1,\dots,c_n$ and balances $b_1,\dots,b_n$. The current upper and lower bounds for b_i are b_i^l and b_i^h, and for b_{max} the bounds are b_{max}^l and b_{max}^h. There are two possible outcomes: the probe went through ("success") or it did not ("failure").

The attacker only updates the bounds if the new estimate is more precise, i.e., the attacker never decreases lower bounds and never increases upper bounds. For simplicity, the algorithm description omits this implementation detail: one should read $h^0 := a$ as $h^0 := max(h^0, a)$.

3.3.1 Success case

In the success case, the probe has been forwarded. This means that the hop can forward at least a, therefore, $b_{max}^l := a$.

Additional information may be derived from this outcome. If the attacker knows which channels the probe could have been forwarded through, the bounds of these channels may be updated. For instance, in a single-channel hop case, the attacker updates both hop bounds and channel bounds for its only channel.

To capture this information in the multi-channel case, the attacker orders the channels in the hop by the upper bounds of their balances b_i^h. If $b_i^h \geq a > b_{i+1}^h$, the probe could only have been forwarded through channels $1,\ldots,i$. To avoid tracking bounds for each channel subset, we use a simplified version of this rule. If $b_1^h \geq a > b_2^h$, this means that only the channel with the highest estimated balance could have forwarded the probe. Therefore, the attacker updates the first channel's balance lower bound: $b_1^l := a$.

This, in turn, allows the attacker to also improve channel estimates in the opposite direction. Indeed, if the balance of the first channel $b_1 \geq a$, then $\tilde{b}_1 = c_1 - b_1 \leq c_1 - a$. Therefore, the attacker updates $\tilde{b}_{max}^h := max_{i\in[1,n]}(c_i - b_i)$ accounting for the new bound on b_1. This updated upper bound implies that all channel-level bounds in `dir1` can also be updated: $\forall i \in [1,n] : \tilde{b}_i^h := max(c_1 - a, c_2 - b_2, \ldots, c_n - b_n)$.

3.3.2 Failure case

In the failure case, the probe has not been forwarded. This means that all channel balances in the hop are strictly lower than a. The attacker updates: $\forall i \in [1,n] : b_i^h := a - 1$. This also allows for updating the hop-level lower bound in the opposite direction. Indeed, $\forall i \in [1,n] : b_i < a \Rightarrow \tilde{b}_i = c_i - b_i > c_i - a$. Note that $\tilde{b}_{max} = max_{i\in[1,n]}(c_i - b_i) > max_{i\in[1,n]}(c_i - a) = max_{i\in[1,n]}(c_i) - a = \tilde{c_1} - a$, where $\tilde{c_1}$ is the highest capacity among the channels enabled in direction `dir1`. Therefore, we may apply the new *lower* bound for the opposite direction: $\tilde{b}_{max}^l := \tilde{c_1} - a + 1$.

3.4 Countermeasures

In this Section, we broadly discuss countermeasure design for channel probing and suggest three ideas. This is the first proper evaluation of these novel countermeasures. While they are not related to parallel channels, we used our advanced parallel-aware model while evaluating these countermeasures in our simulator. Now, let us discuss the countermeasures.

If reliably distinguishing between probes and genuine payments were possible, routing nodes could just ban the attackers without the need for sophisticated countermeasures. However, certain architectural aspects of the LN make such differentiation difficult. First, routing errors are a part of the normal protocol operation. Honest payments may and in fact do fail due to insufficient balances at intermediary hops. Second, the LN uses onion routing to preserve users' privacy. Therefore, even if it were possible to reliably detect probing, routing nodes would not know where the probes are coming from and therefore could not easily ban attackers.

Since channel probing may be applied to spy on both node balances and payments, the incentives of implementing countermeasures are unclear. In the former case, the hop being probed always belongs to the victim. Thus, the privacy solely depends on the victim, although per some countermeasures, routing nodes may agree to help a victim preserve privacy. In both cases, the countermeasures that we suggest might not be not incentive-compatible, as they require sharing failures or delays across the payment path. Applying them might not be in the best interest of economically rational forwarding nodes: failures and delays would reduce their reliability score as calculated by other users, who would then route fewer payments through those nodes. The burden on the user experience of honest payers should be also considered. We leave this evaluation for further discussion, and now present the three countermeasures.

3.4.1 Channel selection: random failure and single random channel

Here we describe two similar strategies. The first one is to "fail with probability p" payments that would otherwise pass. This strategy may be applied when a probing attack is detected. In such case, the attacker needs to issue at most k more probes to be sure that the failure was caused by the strategy rather than an actual lack of balance:

$$P(\text{success } |b_{max} < a) \geq 1 - p^k \tag{1}$$

For example, assuming $p = 0.1$, if we set an error probability $P(\text{probe failure } |b_{max} < a) \leq 0.001$, $k = 3$ would be sufficient. However, this countermeasure might be more difficult to overcome if the node adaptively increases p during the attack.

The second approach is to use the "single random channel" selection strategy. For a hop with n channels, a channel with the maximal balance will be selected with probability $1/n$. So in the case of a probe failure with amount a, the attacker needs to repeat the probing at most cn times for some constant c to make the probing error arbitrarily small:

$$P(\text{ success } |b_{max} < a) \geq 1 - \left(1 - \frac{1}{n}\right)^{cn} \approx 1 - \left(\frac{1}{e}\right)^{c}. \tag{2}$$

Comparing the two approaches, we note that in the first case the binary-search prober needs to make c times more probes, while in the second case the number of extra probes is proportional to the number n of channels in a hop. Since n is typically not large, both cases are similar to the best-effort payment selection strategy.

The "single random channel" strategy leaks information that is otherwise unavailable to the attacker, namely, it reveals channel balances beyond b_{max} in the general case. Having ordered the channels by balance, we can consider the *balance distribution function of a hop* as a monotonic function $f : \{1,2,\ldots,n\} \to \{0,\ldots,b_{max}\}$. The attacker can efficiently recover this function (or, equivalently, the balance distribution of a hop up to a permutation of its channels) by measuring the frequency of probe successes and failures.

3.4.2 Adding artificial delays

Significant time requirements make large-scale probing challenging (we discuss the time-cost in Section 4.2). To leverage this factor against the attacker, honest nodes may add an artificial delay when forwarding payments[8].

The effect of additional delays is twofold. At the very least, this countermeasure slows down the information gathering by the attacker. Moreover, if the network handles heavy payment traffic, the probing results may be outdated by the time the attack is finished. The exact delay might be tuned considering how much obfuscation honest payments can provide. One way to limit the impact of this countermeasure on honest users could be to introduce extra delay only when an attack is detected (e.g., a series of failed payments), though the attacker could overcome this by sending more probes.

3.4.3 Replacing errors

As mentioned earlier, probing is based on the distinction between payment failure and success, which an attacker observes by looking at *what kind of error* is returned *by whom* along the payment path. Normally, routing nodes propagate a forwarding failure back to the sender. However, if a routing node receives an error from the next node in the path, it can pretend the error happened at its own channel (*spoof* the error). The sender has no way of distinguishing genuine errors from spoofed ones. This countermeasure would obfuscate the picture for the attacker. The side-effect of this countermeasure is a decreased precision of node scoring models based on failure history.

4 Metrics

The attacker aims to obtain the highest amount of information at the lowest possible cost. We introduce metrics to quantify how successful the attacker is in achieving this goal.

4.1 Success metrics

We assess the quality of a probing by the amount of information the attacker obtains. Consider a target hop with n channels. The initial uncertainty about channel balances is the number of bits required to store vector B of balances: $\sum_{i=1}^{n} \log_2(c_i+1)$. To reflect the fact that the attacker may be interested in this information up to a certain granularity (i.e., $u = 1000$ satoshis), we use the formula:

$$U_{before} = \sum_{i=1}^{n} \log_2(c_i+1) - n\log_2(u) \tag{3}$$

This formula assumes uniform distribution of balance b_i in the range $[0,\ldots,c_i]$ and independence of balances among the channels of a hop – the worst-case assumption for the attacker.

As a result of probing, the uncertainty decreases to a range between the lower and upper bounds $b_i^l \le b_i \le b_i^h$. The remaining uncertainty can be expressed as[9]:

$$U_{after} = \sum_{i=1}^{n} max(0, \log_2(b_i^h - b_i^l + 1) - \log_2(u)) \tag{4}$$

The information gain thus is:

$$I = U_{before} - U_{after} \tag{5}$$

Full information extraction from a hop is often impossible due to symmetries. For example, in a hop with all $c_i = c$ (channels with equal capacities), it is possible to extract information only up to a permutation of b_i's, and thus up to $\log_2(n!)$ bits of uncertainty would remain (at least without resorting to advanced probing techniques, which we discuss in Section 6.2.2). On the other hand, full information extraction is not always needed. If the attacker is only interested in whether a hop can pass the payment of size a, it is sufficient to find b_{max}. To learn the total balance in a target hop, the precise information about all channels is also unnecessary: the sum $\sum_{i=1}^{n} b_i$ is invariant under permutations.

However, in our evaluation (Section 5), we assume that the attacker wants to know the exact channel balance for every channel in the target hop and assess the attacker's gains with Formula 5, summing up all channels in the target hop. Additionally, we account for the case when the attacker's bounds are misleading (the true balance is not between them). Such a scenario is possible if nodes apply countermeasures, i.e., fail probes that would otherwise succeed. In that case, we set the information gain for this channel to zero.

An alternative way to measure the attacker's gain would be to use hop-based uncertainty (as opposed to channel-based uncertainty defined in Formula 5). After finding b_{max} and $\tilde{b}_{max}$, the only thing the attacker can tell is that:

$$\forall i, \max(c_i - b_{max}^1, 0) \le b_i \le \min(b_{max}^0, c_i). \tag{6}$$

Given the public capacity vector C of the hop and the measured bounds on b_{max} and $\tilde{b}_{max}$, the attacker can compute the hop-based information gain by using Formula (5) and plugging into it the bounds from inequality (6). This also implies that for any hop $c_i \le b_{max} + \tilde{b}_{max}$. This inequality may save a bit of probing compared to naive binary search for each direction separately (but at most a factor of two).

Finally, we note that a hop may resist information extraction. This is especially relevant if the attacker wants to learn

[8] The delay cannot be applied just on the probe receiver node, because then an attacker would expand the route to one more hop, so it is applied (with some probability) at every hop while propagating an error back to the sender.

[9] We take *max* to account for the case when the bounds are closer than the required granularity: the uncertainty is then set to zero instead of a negative value.

the flow of payments that has passed through the hop in a certain time interval. For example, even after passing several payments using best effort strategy, the hop's b_{max} in either direction might not change, if the hop "absorbs" these payments inside its minor channel balances. We discuss this in Section 7.

4.2 Cost and efficiency metrics

Channel probing is a non-blocking activity that takes time, so during the attack the target channel might be used for honest payments. This means that the state of the target channel might change during probing, resulting in outdated or incorrect balance bounds. If an attacker is interested in comparing two snapshots of the same channel to track payments, balance changes during probing might make the attacker's conclusions inaccurate. This issue becomes even more apparent in case of several target channels.

Thus, **making channel probing fast is crucial** for an attacker. We use the time it takes to probe one target hop as our efficiency metric. We also note that probing can be parallelized (if target hops are not included into paths to other target hops).

Capital cost metric should also be considered. There are two components to the capital cost of probing: 1) on-chain fees for opening and closing the attacker's channels; 2) the time value (i.e., interest rate) of capital locked up in those channels for the duration of the attack. The trade-offs between the time-efficiency and capital costs are as follows.

- In regular probing (no jamming), making direct channels[10] with the victims makes probing much faster (because there are no routing failures), but every channel costs an on-chain fee and locks up a certain amount of capital (a function of the target channel capacity).
- Capacity-based jamming has a cost of opening one large channel. It requires paying one on-chain fee (it does not depend on the amount), but the funds locked are close to the aggregate amount of all parallel target channels (e.g., in a single target hop).
- Slot-based jamming has a cost of opening many small channels (and paying an on-chain fee for each of them), although the capital locked in them can be minimal. The exact number of channels is equal to the number of target channels to be jammed, because the attacker's path is limited by the same number of slots (including the first component of their own channels)[11].

The attacker never pays LN routing fees, because probing payments never succeed.

[10] In practice, nodes may forbid opening ad-hoc channels to them.

[11] Assuming all channels have the same number of slots, although in practice an attacker can sometimes find a more "permissive" path.

5 Evaluation

In this Section, we evaluate the attacker's information gain and time cost.

5.1 Simulator

We develop a simulator to model probing in the LN. The simulator takes as input the LN topology graph that contains public information from gossip. To establish the ground truth for the probing evaluation, we populate the graph with channel balances generated uniformly at random.

The attacker's node opens large channels to selected well-connected hubs. It then generates a random list of target hops and probes them one by one as per the algorithm described in Section 3. The information gain obtained during probing, including intermediate hops, is saved and reused. The attacker only considers paths that may succeed, according to the current bounds.

To simulate probing time, we increment a counter at every hop of every probe. We pick a single-hop delay from a list of values from 0.25 to 2.75 seconds with a step of 0.25. The weights are distributed symmetrically around 1.5 seconds, which is in line with prior work [11, 33] and our own experiments on the real network.

The simulator mimics the channel choice behavior of LND and ECLAIR as seen in their source code (C-LIGHTNING does not support parallel channels). In our simulations, we choose the channel choice strategy for each forward randomly, weighted by relative implementation prevalence: 91% LND, 7% C-LIGHTNING, and 2% ECLAIR [14].

We intend to release the simulator as open-source software.

5.2 Dataset

Our experiments are based on a snapshot of public LN channels obtained on 2020-12-28 from our own C-LIGHTNING node running on mainnet[12]. The snapshot contains 3672 nodes and 23026 channels. Compared to a popular Lightning explorer[13] ran by ACINQ (the company behind ECLAIR), our snapshot contains approximately 47% fewer nodes and 30% fewer channels. We explain it by the fact that our node has been launched shortly before capturing the snapshot, while popular public nodes run continuously and accumulate gossip information for a much longer period of time (but then may include non-active or dead nodes).

We proceed with its largest connected component, which consists of 3614 nodes and 22975 channels in 20156 hops. The total network capacity is approximately 927 BTC. As per our snapshot, 4769 channels (21% of total) belong to

[12] Note that here we are only aware of publicly announced channels and nodes. Detecting unannounced channels, while possible in principle [12, 21], is outside the scope of this work.

[13] https://explorer.acinq.co/.

multi-channel hops and hold 416 BTC (45% of total capacity). We thus see that parallel channels hold on average more coins than non-parallel ones and presumably play a significant role in routing. The LN contains 2173 unidirectional channels (9% of total), which hold 67 BTC (7% of total capacity). Compared to parallel channels, unidirectional channels represent a smaller but still non-negligible share of the network. We conclude that accurately modeling parallel and unidirectional channels is important for a more precise analysis of probing, countermeasures and other potential applications. We also note that the set of channels that can only be probed in one direction includes, besides unidirectional channel, leaf channels (i.e., channels where one of the endpoints has a degree of one). The LN contains 1270 leaf channels (6% of total) that hold 8 BTC (1% of total capacity).

5.3 Evaluation

We explore the effectiveness of the probing algorithm described earlier with and without each of the countermeasures: deliberate failures, artificial delays, and error spoofing. We use the information gain defined in Section 4 and the simulated probing time.

In Figure 1, we observe how the attacker's information gain grows if various countermeasures are applied. The results are averaged across 10 experiments with 100 random target hops each. The probability of each countermeasure being applied is set to $p = 0.2$. In the case of combined countermeasures, each forwarding node chooses independently whether to apply any of the selected countermeasures (each with probability p).

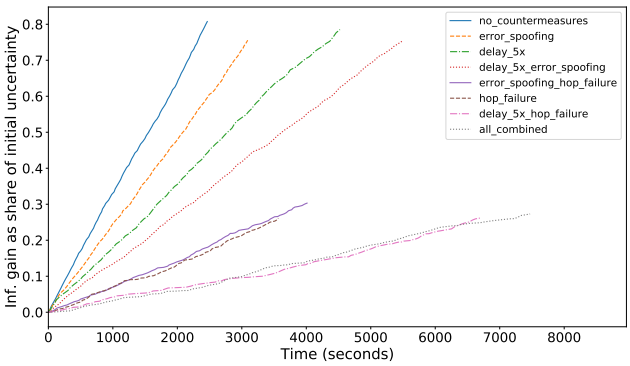

Figure 1: Attacker's information gain growing over time while probing 100 target hops. If done in parallel, attacks are around 100 times faster, but the correspondence between attacks holds.

In the absence of countermeasures, an attacker is able to reach 0.8 information gain over all target hops after approximately 2000 seconds. Note that an attack can be parallelized to approach all targets simultaneously, allowing to cut the time significantly.

The attacker does not gain perfect knowledge due to parallel channels: without additional enhancements discussed in Section 6, the attacker does not know which channels the probes are forwarded through and therefore can, in the general case, obtain only b_{max} but not individual channel balances.

Applying both `error_spoofing` (each forwarding node with probability p spoofs the error message as if it happened at this node) and `delay_5x` (each forwarding node delays a response by a factor of five with probability p for both failed and successful forwards) separately does not prevent an attacker from gaining the same 0.8 information, but makes it 10% and 100% longer, respectively. Error spoofing makes the attack longer because the attacker can no longer rely on the balance information accumulated during previous probings. However, since the attacker still manages to find paths, they get the same 0.8 information. Combining these countermeasures adds another 100% in terms of time.

Unlike the other two countermeasures, applying `hop_failure` (forwarding nodes deliberately fail payments that could have succeed with a probability p) separately makes an attacker stop the attack just around reaching 0.25 information. This is not surprising, since occasional errors are effectively false positives leading an attacker to the wrong results[14]. Additionally, an attacker might stop earlier because they think there are no suitable path to the target, after getting fake errors from many paths.

Adding `error_spoofing` only improves `hop_failure` by making it 10% longer in terms of time, probably because false positives from `hop_failure` are triggered by intermediate routing nodes even before `error_spoofing` comes into play.

Finally, a combination of `delay_5x` and `hop_failure` achieves the best result: it simultaneously limits the attacker's information gain to around 0.25 and makes it significantly longer to achieve it (three times longer comparing to gaining 0.8 knowledge in the case of no countermeasures). Combining all three countermeasures does not improve further: again, because `hop_failure` augments `error_spoofing`.

6 Distinguishing between parallel channels for improved probing efficiency

Our probing method (Section 3) accounts for parallel channels but makes no active attempts to distinguish between them. However, precisely probing a multi-channel hop in the general case is impossible unless the attacker knows which channel the probes go through. The attacker may achieve this goal by blocking all channels except for one. We now describe multiple ways to block channels via jamming.

[14] A more sophisticated attacker could be aware of these measures and adjust the attack (e.g., repeat failed probes). Still, an attacker is bounded by time (see Section 4.2).

6.1 Theory

The attacker may manipulate the channels in a target hop to make sure that probes go through the desired channel. There are multiple methods to manipulate the channels: policy-aware probing (fees), blocking by capacity, and blocking by slots.

In the next Section, we demonstrate how distinctions in fee policies on parallel channels of the same hop allow access to individual channels. In practice, an attacker might also **exploit other channel policies** (see Section 2.2.3). Note that if all policies are the same across all parallel channels, an attacker cannot exploit this.

6.1.1 Probing with fee policy

In fee-aware probing, the attacker carefully chooses fees such that the probe can only be forwarded through a subset of cheaper channels, assuming the fees differ among some of the parallel channels. If implemented trivially, this technique allows for probing the cheapest channel as if it had no parallel channels. Furthermore, all channels can be grouped into increasing nested subsets according to the different fee levels in the hop. Each fee level (i.e. a set of nodes with fee less or equal than f) has its own local balance maximum b^f_{max}. If the balance of the smallest-fee channel has global b_{max} then nothing else can be learned. The other extreme is if fees and balances (f_i, b_i) both form an increasing sequence. In that case, the attacker can learn all channel balances by gradually increasing the fees. In the general case, if the channels are sorted by fees, then the attacker can learn b^f_{max} for each fee level (similar as previously via binary search, but over the fee-level subset) and thus gain much more information about balances of the hop than just a single hop-level b_{max}.

We recall that in LND the routing node picks a random channels satisfying the fee and balance, while in ECLAIR the channel is chosen deterministically. This difference is not important if we resort only to fee-probing, since it is irrelevant over which channel with sufficient balance the probe would pass. However, it may becomes relevant if we combine fee probing with jamming (as described later).

6.1.2 Probing with capacity-based jamming

An attacker might jam a certain balance in a hop by sending one large payment to their own nodes via target channels, and never resolving it on the receiving side.

After learning that hop's b_{max} via probing technique of the previous section, the attacker may choose to send a payment of size b_{max} and hold it on the receiving side. Now, the attacker faces the same hop with one channel less, and this technique can be applied iteratively by jamming channels one-by-one. Assuming there are k channels in a hop, the attacker needs k jamming probes and at most $\sum_{i=1}^{k} \log_2(c_i)$ binary search probes (this is worst case when all $b_i = c_i$) to completely discover all k channel balances in the hop.

6.1.3 Probing with slot-based jamming

An attacker may exploit a limit of in-flight payments to completely disable one of the channels by sending several hundreds of small payments and not resolving them for the time of probing. To optimize the use of time and payment slots, an attacker may modify the attack depending on the victim's payment forwarding strategy (similarly to fee-aware probing).

In the ECLAIR case, an attacker may be able to direct jamming towards a certain channel. Then, it would require sending 483 payments to jam channels one by one, which is considerably easier. Since this can be done in parallel, it takes up to 3 seconds (the time of a single payment).

In the case of LND, for N channels, an attacker would need to send and hold $483 \cdot N - 1$ tiny payments in-flight, to leave one slot for probing. Then, after binary-search probing the only available channel left, an attacker will block the last slot, blocking that channel, and will release a new random slot. With a good chance it will be a slot in another channel which can again be probed. This process requires about $N \cdot log(N) \cdot \log_2(max(c_i))$ probes.

6.1.4 Jamming challenges

In some cases, jamming might be challenging or impossible. For example, if there are several channels between Alice and Bob, and there is only one channel to reach Alice, it would be impossible to slot-jam more than one Alice-Bob channel, since slots for all hops on the path would also be jammed. The same applies for capacity jamming[15]. These limits apply on "both sides" of the victim, although in some cases an attacker can negotiate expanding these limits. Thus, it would help if the attacker is directly connected to the victim or if there are several disjoint paths to the victim.

It would be difficult to control the in-flight limits, if other payments are also flowing across these channels. The positive side-effect of this is that this could be a *new method to track payments*, since they would temporarily block the only available slot.

Jamming is detectable on the victim's side, so they might take action to obfuscate the state of their channels. Finally, a node might decide to use one of its higher fee channels even if sender specified lower fees. This can happen if the payment helps node to rebalance its channels, which has economic value for the node, or as a countermeasure against jamming.

6.1.5 Combining fees and jamming

Fee-based probing can be enhanced with jamming: after probing the cheapest channel (or a subset of channels), they can be

[15] Note that these channels are not "useless": e.g., they might be bottlenecked by slots, but available by capacity.

jammed, so that the next-by-fee channel becomes effectively the cheapest, and may be now probed by fees. Moreover, jamming gives access to individual channel balances inside the fee-level subset of nodes. The only downside of this approach is that fee-probing the $(N+1)$-st channel would require keeping N channels jammed (meaning locked liquidity if jamming by capacity).

On the other hand, jamming can in turn be enhanced with fee considerations: to jam specific channels, an attacker might tune fees, as described previously.

6.2 Experiments on the real network

To confirm that the proposed attack optimizations indeed work, we test them against the real Lightning Network.

6.2.1 Setting

Our setting consisted of five nodes located on the same machine and running on different ports (see Figure 2)[16].

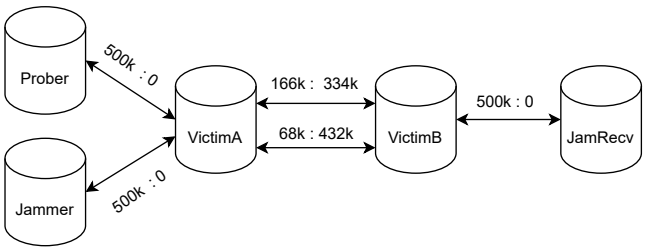

Figure 2: Experimental setup

For Prober and Jammer, we used the C-LIGHTNING implementation to build on the previous work on channel probing. For JamRecv, we modified the C-LIGHTNING implementation to `sleep(120)` after receiving a payment for a non-intended invoice, so that the channels on the route from the Jammer are indeed jammed. For VictimA and VictimB, we used the ECLAIR implementation, since C-LIGHTNING does not support parallel channels (and that is what we wanted to test).

We did not verify these techniques against the "real" (non-our) victims, and focused on the highly-controlled experiment instead. All the remaining logic was implemented in the simulator.

For both experiments, we start by opening two channels between VictimA and VictimB, allocating the balances as 68k:432k and 166k:334k. We also open supplementary channels from the attacker to the victims. The setting is summarized in Figure 2.

Since the victims run ECLAIR, they inherit the capacity-balance channel choice strategy, although the experiments we describe in this section are agnostic to it and would work for other implementations as well.

[16]Note that Prober and Jammer can be the same node with two channels for each activity, but we make them distinct for simplicity.

Probing in these conditions without optimizations provides an upper bound of 167k satoshis for both channels, and the lower bound for the entire hop is 165k satoshis.

6.2.2 Jamming

The jamming-enhanced probing proceeds as follows.

1. To infer the balance of the smaller channel, an attacker sends a payment of 150k satoshis from the Jammer node to the JamRecv node, and holds it for two minutes on the receiving side.

2. The available balances are 68k:432k and 16k:334k (note that the in-flight balance is not available here for either direction).

3. Since the 67k channel has the largest balance in this hop, an attacker can actually probe it.

4. Probing in these conditions yields an estimate of $[66k : 68k]$, which indeed represents the balance of the second channel.

5. After the probing is done, JamRecv notifies VictimB that the payment has failed, thus an attacker does not have to pay routing fees.

Thus, we confirmed that channel jamming might indeed be used to improve channel probing.

6.2.3 Probing by fee

For these experiments, only three nodes were relevant: Prober, VictimA, and VictimB. To demonstrate how an attacker can use this technique, we update the larger channel from the previous experiment to require non-zero fees.

The attacker configures the Prober node to send probes while allocating fees sufficient to cover the more expensive channel. Then, by sending such probes, an attacker yields the same result, inferring the balance of the larger (non-zero-fee channel). Then, to probe the smaller channel, an attacker configures the Prober node to send only zero-fee probes. Since the routing node would now allow forwarding over the smaller zero-fee channel, the attacker successfully infers balance of the smaller channel: $[66k : 68k]$.

7 Payment and flow inference

In this section, we show how to expand our balance knowledge into a knowledge of payment flows through a hop or sequence of hops. This method is mostly agnostic to having parallel channels.

Given a set of sufficiently frequent balance snapshots, the attacker can construct a balance difference graph, in which edges with non-zero difference correspond to payments. This

will work well even for several disjoint payments, since in a simple single-channel hop scenario the balances along the route will be shifted by the same amount (modulo fees), and thus it would be easy to discover the sender, the receiver and the amount.

However, if some cases payments are absorbed by parallel channels, the inference method needs to be more precise. Previous work [12] has shown that at the current level of activity in the LN is relatively low (2000 payments per day). Under these conditions, it is sufficient to make a balance snapshot every 30 seconds to achieve 66% success.

While obtaining a full network snapshot at this speed is challenging, it is much easier if our goal is to track payment flows between a given pair of nodes A and B. If they are endpoints of one hop, extracting 10 most significant bits of the hop balance requires about 13 probes in our simulations (0.77 bits per probe).

Multiple targets can be tracked in parallel. If A and B are not directly connected, the attacker needs to track balances in a set of 10 shortest paths between them (most of our probes succeed within 10 path attempts). Since diameter of the whole network is only 6 hops [28], and typical path lengths are 3-6 hops, the size of a target sub-network in such case is around 50. The worst case for the attacker is when equal amounts are sent via intersecting paths within a single time slot.

8 Advanced countermeasures

In this Section, we consider alternative forwarding strategies that might be employed by routing nodes to prevent probing. Some of these countermeasures require honest users to have parallel channels, while others are more general.

8.1 Intra-hop payment split

Due to non-strict forwarding, routing nodes can choose any channel to forward a payment to the next node (strictly specified by the sender). However, they do not perform *intra-hop payment split* among the channels inside a hop. Intra-hop split is being discussed as part of the future switch to the new type of channel construction, namely PTLC [19, 38]. This feature could be used by routing nodes to optimize hop bandwidth or as a countermeasure against probing. Intra-hop split is similar to multi-part payments (MPP) but differs in an important way. In MPP, the sender chooses how to split the payment, and forwarding nodes cannot alter this. In intra-hop payment split, a forwarding node decides how to split a payment among its adjacent channels.

In the most direct *best-effort intra-hop split*, the node splits the payment if the total sum of channel balances is lower than the payment amount. This behavior, while optimal in terms of efficiency, would make probing easier: a hop becomes equivalent to a pair of directed channels with balances equal to the sum of balances in each direction. Thus, a simple probing would be sufficient to discover the sum of balances.

8.2 Rebalancing

Any form of on-demand rebalancing (e.g., JIT routing [18]) is likely to hinder probing, similarly to *intra-hop payment split*. In fact, rebalancing is a better version of straightforward *intra-hop payment split* because it does not aggregate channels into a single pseudo-channel, thus revealing the aggregate hop balance. However, rebalancing is likely to have the same disadvantage if the aggregate hop balance is lower than the largest channel capacity: in this case, the largest channel after rebalancing would represent the aggregate hop balance.

8.3 Adversarial multiple channel selection strategy

Is it possible to route payments while minimizing changes in b_{max}? What should be the best channel structure for achieving this goal?

If a channel is actively used, then it could execute payments in batches once every time Δt (i.e., 1 second). The payments in a batch are re-ordered to minimize changes of b_{max} and $\tilde{b}_{max}$. In the best case, if payments of equal amounts are forwarded in the opposite directions, they "cancel" each other and become invisible to the attacker. The task seems more challenging if the flow through the hop is unbalanced. In that case, the hop might need to perform regular rebalancing.

The hop can also detect probing using heuristics, i.e., a sequence of payments failing due to wrong invoices and with specific binary-search amounts. In that case, future probes can be routed through special channels with artificially set b_{max}-es. However, since adversarial probing can be more stealthy, it is advisable and indeed possible (given sufficient liquidity in parallel channels) to have a more generic strategy that information-theoretically conceals the flows.

8.4 Gates

Consider a scenario when tightly-coupled clusters of nodes with high payment activity among them emerge in the LN. In that case, the *gate* channel structure may limit the capabilities of an out-of-cluster prober. In particular, node operators may limit capacities or balances of channels connecting the cluster of nodes to the rest of the network (the gate channels). Limiting by balances is preferable since the capacity is publicly visible (if channels in the cluster are publicly announced). The gate channels may only be open at short agreed upon time intervals to let the traffic in and out of the cluster. The disadvantage of gates is that they impede honest payments between the cluster and the rest of the network.

8.5 Private channels

Since an attacker has to rely on the public channel metadata (to pick probe amounts and update estimates), having some of the information unavailable indeed requires extra considerations.

In practice, the existence of private channels results in the following situations:

- an attacker might be able to discover private channels easily and even infer their balances (e.g., if balance of the private channel exceeds balances of public channels);
- an attacker cannot discover private channels easily (e.g., if private channel balance does not exceed balances of public channels).

Even in the former case (and indeed in the latter), an attacker cannot just blindly use the regular probing approach. For example, because the capacity of the private channel is unknown, an attacker will be unable to use bidirectional probing easily.

For now, we just note that private channels might be useful in protection against channel probing. We leave a more detailed exploration of private channels for future work.

9 Discussion and Future Work

9.1 The countermeasures vs. incentives

The fundamental limitation of many countermeasures against channel probing, including those discussed in Section 3.4, is that they are often not incentive-compatible for three reasons:

- the privacy of end-users (payment senders and receivers) often depends on the routing nodes, since probing the entire route is an intuitive way to infer a payment;
- the privacy of routing nodes may depend on the cooperation of other routing nodes;
- most countermeasures (both deployed for self-defense and for helping others) require selectively failing payments, delaying responses and limiting bandwidth. This makes those nodes less attractive for routing and puts them in a worse economic position than those that do not implement countermeasures.

We may also envision a future where protecting privacy is a competitive advantage for routing nodes, while they also charge extra fees for enabling privacy-preserving features. The challenge here is to make it provable that routing nodes adhere to privacy practices.

9.2 The countermeasures vs. user experience

Another shortcoming of many of the countermeasures discussed in the Section 3.4 and Section 8 (except Section 8.3) is that they put the burden not only on the attacker but also on honest users: payments fail more often and errors arrive slower. UX burden may become the cost for privacy features. Perhaps it is also possible to deploy countermeasures only during the attack, although since the attacks are currently cheap and easy, the LN could be under these attacks constantly.

We expect that advanced *flow-concealing* countermeasures proposed in Section 8.3 would be more attractive for routing nodes and users, since they may provide perfect security against probing at the cost of higher routing liquidity for routing nodes, with a limited impact on both user experience and routing nodes' economic incentives.

9.3 Non-trivial probing optimizations

To speed-up the binary search probing, an attacker might consider **behavioral patterns** of nodes. E.g., merchants are more likely to have a high inbound balance, so the first probe might be in the third quarter instead of the middle of the initial bounds interval. An attacker may also consider the historical behavior of nodes and channel management strategies of the particular LN implementation that the victim runs.

An attacker might also consider **bulk probing**: constructing a probing path such that multiple target channels are probed at once, which might be relevant for probing hubs. Deriving sophisticated probing algorithms is a future challenge.

10 Related work

Attacks on the Lightning Network can be grouped into privacy-related [2, 11, 12, 16, 24, 25, 33, 36], DoS-related [8, 14, 17, 24, 26, 32, 34] and incentive-related [35].

Prior work on channel probing introduced the general idea of probing based on binary search [11], probing channels from both ends [36], and controlling both sender and receiver of the probes [12].

We build on top of prior work, adding the notions of parallel and unidirectional channels (which was never thoroughly explored before), as well as accommodating probing from any location in the network (which was considered only in [33]).

Multiple simulators of the LN have been proposed. Existing simulators were designed to analyze honest economic activity [2, 5, 37] or the cost of opening payment channels [4, 6]. Our simulator is the first one to focus on probing. We start with the probing method presented in [33] and augment it according to our model improvements.

Among other relevant attacks on the LN, channel jamming [14, 32] and channel policy exploitation [22] are used in our work to expand attacker's capabilities and overcome

the issues with parallel channels. One of the improvements to payment forwarding [38] (namely *splitting it up via AMP to two sub-payments*) influenced our "intra-hop payment split" countermeasure.

Free channel probing was mentioned as an adjacent issue to jamming, and rate-limiting was proposed to mitigate them [31]. Currently, several rate-limiting approaches are being discussed [23, 27].

Although the current fee structure was previously found to be non-economical [2], we believe that incentive-compatibility of probing countermeasures with reliability-based routing protocols [9] should be discussed.

The tension between privacy and utility of the routing nodes found in [29] are partially confirmed in our work.

Using channel probing as a building block for payment inference was briefly discussed in [12], and our optimizations make those attacks much more feasible, while also providing ways to prevent them.

11 Conclusion

The Lightning Network is a promising technology to improve Bitcoin's scalability. While the LN has the potential to also become a privacy-preserving tool, recent work has demonstrated that the existing public payment channel network leaks private information about payments.

In this work, we advance the understanding of channel probing – an attack that allows an adversary to infer user balances by sending fake payments. Compared to state-of-the-art, we provide the most precise network model that reflects the existence of parallel and unidirectional channels. Such channels are common in the LN and obviate previous work on probing. We develop the first probing-focused LN simulator that implements our model. We describe information-gain, time, liquidity metrics and mathematical models which could make the attack more practical.

Finally, we suggest countermeasures. Even though they are not specific to parallel channels, we evaluate them in our novel model. Using our simulator, we demonstrate that the suggested countermeasures reduce the attacker's information gain from 80% to 30%, while increasing the time an attack takes by 2-4x. Although the countermeasures work well and make probing quite infeasible, they may negatively influence incentive-compatibility and user experience. Navigating these trade-offs should be discussed in the Lightning community.

12 Acknowledgments

We thank Antoine Riard for thoughtful feedback. This work is partially supported by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537). Contributions of Gleb Naumenko were supported with a grant by 100x Group, the holding structure for the BitMEX platform.

References

[1] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2013.

[2] Ferenc Béres, István András Seres, and András A. Benczúr. A cryptoeconomic traffic analysis of bitcoins lightning network. *CoRR*, abs/1911.09432, 2019.

[3] BOLT. Lightning network specifications, 2019.

[4] Simina Brânzei, Erel Segal-Halevi, and Aviv Zohar. How to charge lightning, 2017.

[5] Marco Conoscenti, Antonio Vetrò, J. Martin, and Federico Spini. The cloth simulator for htlc payment networks with introductory lightning network performance results. *Inf.*, 9:223, 2018.

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

[7] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 201–226. Springer, 2020.

[8] Jona Harris and Aviv Zohar. Flood & loot: A systemic attack on the lightning network. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 202–213. ACM, 2020.

[9] Tankred Hase and Valentine Wallace. Smarter autopilot, Apr 2019.

[10] Mike Hearn and Jeremy Spilman. Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html, 2013.

[11] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal Pedrosa, Cristina Pérez-Solà, and Joaquín García-Alfaro. On the difficulty of hiding the balance of lightning network channels. In Steven D.

Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 602–612. ACM, 2019.

[12] George Kappos, Haaroon Yousaf, Ania M. Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An empirical analysis of privacy in the lightning network. *CoRR*, abs/2003.12470, 2020.

[13] 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. *login Usenix Mag.*, 38(6), 2013.

[14] Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. *CoRR*, abs/2002.06564, 2020.

[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[16] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. In Steven Furnell, Paolo Mori, Edgar R. Weippl, and Olivier Camp, editors, *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP 2020, Valletta, Malta, February 25-27, 2020*, pages 7–14. SCITEPRESS, 2020.

[17] Cristina Pérez-Solà, Alejandro Ranchal Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquín García-Alfaro. Lockdown: Balance availability attack against lightning network channels. *IACR Cryptology ePrint Archive*, 2019:1149, 2019.

[18] René Pickhardt. Just in time routing (jit-routing) and a channel rebalancing heuristic as an add on for improved routing success in bolt 1.0, 2019.

[19] Andrew Poelstra. Lightning in scriptless scripts, Mar 2017.

[20] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning network: Scalable off-chain instant payments. Technical report, 2016.

[21] BitMEX Research. Lightning network (part 7) – proportion of public vs private channels, 2020.

[22] Antoine Riard. [achieved a new round, i think...], Oct 2020.

[23] Antoine Riard and Gleb Naumenko. Stake certificates, 2020.

[24] 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&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*, pages 347–356. IEEE, 2019.

[25] Matteo Romiti, Friedhelm Victor, Pedro Moreno-Sanchez, Bernhard Haslhofer, and Matteo Maffei. Cross-layer deanonymization methods in the lightning protocol. *CoRR*, abs/2007.00764, 2020.

[26] Rusty Russel. [lightning-dev] loop attack with onion routing.., Aug 2015.

[27] Rusty Russel. [lightning-dev] a proposal for up-front payments., Nov 2019.

[28] István András Seres, László Gulyás, Dániel A. Nagy, and Péter Burcsi. Topological analysis of bitcoin's lightning network. *CoRR*, abs/1901.04972, 2019.

[29] Weizhao Tang, Weina Wang, Giulia C. Fanti, and Sewoong Oh. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. In Edmund Yeh, Athina Markopoulou, and Y. C. Tay, editors, *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Boston, MA, USA, June, 8-12, 2020*, pages 81–82. ACM, 2020.

[30] Bastien Teinturier. Trampoline onion format (feature 24/25).

[31] Bastien Teinturier. Spamming the lightning network, Nov 2020.

[32] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. A quantitative analysis of security, anonymity and scalability for the lightning network. In *2020 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, September 7-11, 2020*. IEEE, 2020.

[33] Sergei Tikhomirov, René Pickhardt, Alex Biryukov, and Mariusz Nowostawski. Probing channel balances in the lightning network. *CoRR*, abs/2004.00333, 2020.

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

[35] Itay Tsabary, Matan Yechieli, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. *CoRR*, abs/2006.12031, 2020.

[36] Gijs van Dam, Rabiah Abdul Kadir, Puteri N. E. Nohuddin, and Halimah Badioze Zaman. Improvements of the balance discovery attack on lightning network payment channels. In Marko Hölbl, Kai Rannenberg, and Tatjana Welzer, editors, *ICT Systems Security and Privacy Protection - 35th IFIP TC 11 International Conference, SEC 2020, Maribor, Slovenia, September 21-23, 2020, Proceedings*, volume 580 of *IFIP Advances in Information and Communication Technology*, pages 313–323. Springer, 2020.

[37] Y. Zhang, D. Yang, and G. Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019.

[38] ZmnSCPxj. [lightning-dev] a payment point feature family (multisig, dlc, escrow, ...), Oct 2019.

[39] ZmnSCPxj. Outsourcing route computation with trampoline payments, 2019.

A Uncertainty metrics for unidirectional channels

Bidirectional probing may not be always available. There are three cases: (a) *leaf nodes* that cannot be connected to and thus can be only probed in one direction. For leaf nodes this is direction from the core of the network towards the leaf node; (b) unidirectional channels (similar to leaf node channels for our purposes); (c) intermediate graph nodes that are connected to the rest of the network via channels either with very low capacity or with very low balance.

In the first two cases (leaf nodes and partially active channels), the attacker can only measure b_{max} in one direction (so we will omit in the following formulas its directional index). It gives them the information that for all the channels of the hop for which $c_i \geq b_{max}$ it follows $b_{max} \geq b_i$. Lower bound information, however, may be obtained only on hop as a whole: when a payment of size a passes, the uncertainty of the hop reduces to:

$$\log_2((\prod_{\forall c_i > a}(c_i+1)) - a^t) + \sum_{\forall c_i \leq a} log_2(c_i+1), \qquad (7)$$

here $t = |\{i|c_i > a\}|$ is the number of channels with capacity strictly larger than a. The second term in this sum indicates that the attacker learns nothing about other channels of this hop. It can be proven that expression (7) is monotone in a and reaches minimal uncertainty for $a = b_{max}$. If upper and lower bounds: $b^l_{max} \leq b_{max} \leq b^h_{max}$ are known, they can be substituted in (7) instead of c_i and a, respectively. If b_{max} is known (as happens during the probing attack), the best achievable bound on the hop balance uncertainty (for only unidirectional probing) is as follows (for t defined as above $t = |\{i|c_i > b_{max}\}|$):

$$\log_2((b_{max}+1)^t - (b_{max})^t) + \sum_{\forall c_i < b_{max}} log_2(c_i+1) \approx \qquad (8)$$

$$\log_2(t) + (t-1)\log_2(b_{max}+1) + \sum_{\forall c_i < b_{max}} log_2(c_i+1). \qquad (9)$$

This formula comes directly from equation (7) by replacing c_i's by b_{max}, since all balances are lower or equal to this value. Luckily, in the real LN leaf nodes rarely create multi-channel hops.

The situation with narrow capacity or balance to a node is similar: it limits what the attacker can derive about the b_{max} bounds in one or both directions. Due to economic incentives, nodes probably do not create high-capacity hops surrounded from both sides with low-capacity ones, essentially wasting liquidity, unless they are only interested in paying each other or unless this is used as a countermeasure (see Section 8.4).