Inferring Sensitive Information in Cryptocurrency Off-Chain Networks Using Probing and Timing Attacks

Utz Nisslmueller[1], Klaus-Tycho Foerster[1]
Stefan Schmid[1], and Christian Decker[2]

[1] Faculty of Computer Science, University of Vienna, Vienna, Austria
[2] Blockstream, Zurich, Switzerland

Abstract. Off-chain networks have recently emerged as a scalable solution for blockchains, allowing to increase the overall transaction throughput by reducing the number of transactions on the blockchain. However, off-chain networks typically require additional bootstrapping and route discovery functionality to determine viable routes. For example, the Lightning Network (LN) uses two mechanisms in conjunction: gossiping and probing. This paper shows that these mechanisms introduce novel vulnerabilities. In particular, we present two attacks. The first one, which we shall call a probing attack, enables an adversary to determine the (hidden) balance of a channel or route through active probing and differentiating the response messages from the route participants. The second one, which we shall call a timing attack, enables the adversary to determine the logical distance to the target in hops, given that geographical data of LN nodes is often publicly listed, or can be inferred from allocated IP addresses. We explore the setup and implementation of these attacks and address both the theoretical and practical limitations these attacks are subject to. Finally, we propose possible remediations and offer directions for further research on this topic.

Keywords: Lightning · Confidentiality · Probing Attack · Timing Attack.

1 Introduction

Decentralized cryptocurrencies such as Bitcoin and Ethereum revolutionized the way monetary transactions can be performed, without requiring a trusted third party, central bank or intermediaries. The underlying technology, the blockchain, allows to record transactions reliably in a public distributed ledger. A key challenge faced by today's cryptocurrencies concerns their scalability. supporting only tens of transactions per second, compared to the thousands of transactions per second supported by systems such as Visa. The bottleneck is the required global consensus algorithm.
A promising solution to mitigate the blockchain scalability problem are off-chain networks [9], a.k.a. payment channel networks (PCNs) or second-layer blockchain

networks. These networks allow participants to make payments directly through a network of *peer-to-peer* payment channels, and hence to avoid the overhead of global consensus protocols and committing transactions on-chain. Bitcoin Lightning [18], Ethereum Raiden [22], XRP Ripple [8], and other off-chain networks promise to reduce load on the underlying blockchain and hence to increase transaction throughput. PCNs may also reduce transaction fees, since only one counterparty is responsible for validating a payment initially, rather than the whole network.

PCNs can be represented as graphs, where each node represents a user and each weighted edge represents funds escrowed on a blockchain; these funds can be transacted only between the endpoints of the edge. Many payment channel networks employ source routing: the source of a payment specifies the complete route for the payment. If the global view of all nodes is accurate, source routing is highly effective because it finds all paths between pairs of nodes. Naturally, nodes are likely to prefer paths with lower per-hop fees, and are only interested in paths which support their transaction, i.e. which have sufficient channel capacity.

The fact that nodes need to be able to find routes however also requires mechanisms for nodes to learn about the payment channel network's state. The two typical mechanisms which enable nodes to find and create such paths are *gossip* and *probing*. The gossip protocol defines messages which are to be broadcast in order for participants to be able to discover new nodes and channels and keep track of currently known nodes and channels [15]. *Probing* is the mechanism which is used to construct an actual payment route based on a local network view delivered by gossip, and ultimately perform the payment. In the context of Section 4, we are going to exploit probing to discover whether a payment has occurred over a target channel. The gossip store is queried for viable routes to the destination, based on the desired route properties [24]. Because the gossip store contains global channel information, it is possible to query payment routes originating from any node on the network. Due to privacy concerns, gossip messages only include the *total* balance for any given channel rather than the balance each node is holding.

We explore whether the inherent need for nodes to discover routes in general, and the gossip and probing mechanisms in particular, can be exploited to infer sensitive information about the off-chain network and its transactions.

This paper improves upon the preliminary research in [20] in several aspects. Generally, there is a clearer distinction between design and implementation of the analyzed attacks. The implementation for both attacks now covers a lab implementation (§4.2, §5.2) and actual BTC testnet runs (§4.3, §5.3). For the probing attack in §4, we have formalized the procedure into three algorithms (Algorithms 1, 2 and 3). For the timing attack, our research has shown that the

timing attack might be more promising than initially stated in [20], which is discussed in §5.4.

1.1 Our Contributions

We uncover and analyze two novel threats for the confidentiality of off-chain networks. As a case study, we consider the Lightning Network. We present two attacks, an active one and a passive one. The active one is a *probing attack* in which the adversary wants to determine the maximum amount which can be transferred over a target channel it is directly or indirectly connected to, by active probing. The passive one is a *timing attack* in which the adversary discovers how close the destination of a routed payment actually is, by acting as a man-in-the middle and listening for / analyzing certain well-defined messages. We then analyze these attacks, identify limitations and also propose remediations for scenarios in which they are able to produce accurate results.

1.2 Organization

The remainder of this paper is organized as follows. We introduce some preliminaries in Section 2 and discuss related work in Section 3. We describe the probing attack in Section 4 and the timing attack in Section 5. We conclude in Section 6.

2 Preliminaries

While our contribution is applicable to the concept of off-chain networks in general, to be concrete, we will consider the Bitcoin Lightning Network (LN) as a case study in this paper. In the following, we will provide some specific preliminaries which are necessary to understand the remainder of this paper.

The messages which are passed from one Lightning node to another are specified in the Basics of Lightning Technology (BOLTs) [17]. Each message is divided into a subcategory, called a layer. This provides superior separation of concerns, as each layer has a specific task and, similarly to the layers found in the Internet Protocol Suite, is agnostic to the other layers.

For example in Lightning, the `channel_announce` and `channel_update` messages are especially crucial for correct payment routing by other nodes on the network. `channel_announce` signals the creation of a new channel between two LN nodes and is broadcast exactly once.

`channel_update` is propagated at least once by each endpoint, since even initially each of them may have a different fee schedule and thus, routing capacity may differ depending on the direction the payment is taking (i.e., when c is the newly created channel between A and B, whether c is used in direction AB

or BA). Once a viable route has been determined, the sending node needs to construct a message (a transaction "request") which needs to be sent to the first hop along the route. Each payment request is accompanied by an onion routing packet containing route information. Upon receiving a payment request each node strips one layer of encryption, extracting its routing information, and ultimately preparing the onion routing packet for the next node in the route. For the sake of simplicity, cryptographic aspects are going to be omitted for the rest of this chapter. We refer to [14] and [16] for specifics.

Two BOLT Layer 2 messages are essential in order to to establish a payment chain:

- `update_add_htlc`: This message signals to the receiver, that the sender would like to establish a new HTLC (Hash Time Locked Contract), containing a certain amount of millisatoshis, over a given channel. The message also contains an `onion_routing_packet` field, which contains information to be forwarded to the next hop along the route. In Figure 1, the sender initially sets up an HTLC with Hop 1. The `onion_routing_field` contains another `update_add_htlc` (set up between Hop 1 and Hop 2), which in turn contains the ultimate `update_add_htlc` (set up between Hop 2 and Destination) in the `onion_routing_field`.
- `update_fulfill_htlc`: Once the payment message has reached the destination node, it needs to release the payment hash preimage in order to claim the funds which have been locked in the HTLCs along the route by the forwarded `update_add_htlc` messages. For further information on why this is necessary and how HTLCs ensure trustless payment chains, see [4]. To achieve this, the preimage is passed along the route backwards, thereby resolving the HTLCs and committing the transfer of funds (see Steps 4, 5, 6 in Figure 1).

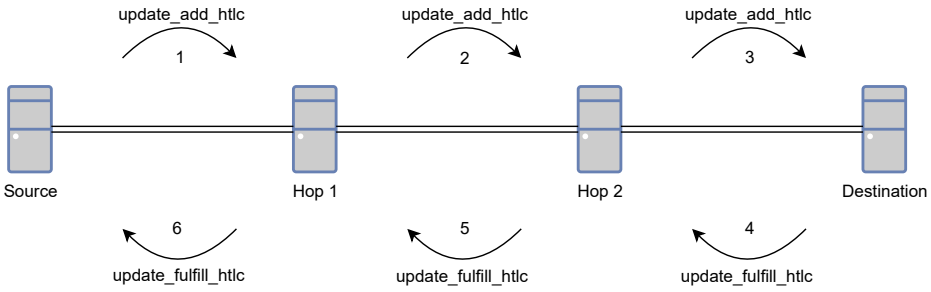

Fig. 1. An exemplary transaction from source to destination, involving two intermediate nodes. [20]

The gossip messages mentioned earlier are sent to every adjacent node and eventually propagate through the entire network.

`update_add_htlc` and `update_fulfill_htlc` however, are only sent/forwarded to the node on the other end of the HTLC.

In order to test the attacks proposed in Section 4 and Section 5, we have set up a testing network consisting of four c-lightning [2] nodes, with two local network computers running two local nodes each (Figure 2). Nodes 1 and 2 are connected via a local network link and can form hops for payment routes between Nodes 3 and 4. In order to interact with the nodes, we have made use of c-lightning's RPC interface and built our software tool set in Python [19]. The tests and their corresponding results in §5 have also been verified with LND [3], another BOLT-conform Lightning Network implementation, written in Go.

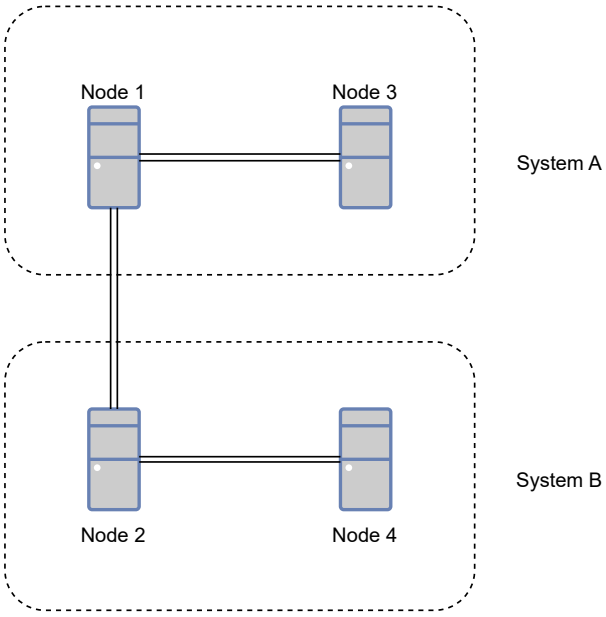

Fig. 2. Local Testing Setup [20]

3 Related Work

Off-chain networks in general and the Lightning network in particular have recently received much attention, and we refer the reader to the excellent survey by Gudgeon et al. [9]. The Lightning Network as a second-layer network alternative to pure on-chain transactions was first proposed by [21], with the technical specifications laid out in [18]. Despite being theoretically currency-agnostic, current implementations such as c-lightning [2] and LND [3] support BTC exclusively.

A popular alternative for ERC-20 based tokens is the Raiden Network [22].

Several papers have already analyzed security and privacy concerns in off-chain networks. Rohrer et al. [23] focus on channel-based attacks and propose methods to exhaust a victim's channels via malicious routing (up to potentially total isolation from the victim's neighbors) and to deny service to a victim via malicious HTLC construction. Tochner et al. [26] propose a denial of service attack by creating low-fee channels to other nodes, which are then naturally used to route payments for fee-minimizing network participants and then dropping the payment packets, therefore forcing the sender to await the expiration of the already set-up HTLCs.

[10] provides a closer look into the privacy-performance trade-off inherent in LN routing. The authors also propose an attack to discover channel balances within the network. Wang et al. [27] examine the LN routing process in more detail and propose a split routing approach, dividing payments into large size and small size transactions. The authors show that by routing large payments dynamically to avoid superfluous fees and by routing small payments via a lookup mechanism to reduce excessive probing, the overall success rate can be maintained while significantly reducing performance overhead. Beres et al. [6] make a case for most LN transactions not being truly private, since their analysis has found that most payments occur via single-hop paths. As a remediation, the authors propose partial route obfuscation/extension by adding multiple low-fee hops. Currently still work in progress, [5] is very close to [4] in its approach and already provides some insights into second-layer payments, invoices and payment channels in general. The Lightning Network uses the Sphinx protocol to implement onion routing, as specified in [14]. The version used in current Lightning versions is based on [7] and [11], the latter of which also provides performance comparisons between competing protocols.

4 Probing Attack

4.1 Design

The Lightning Network uses an invoice system to handle payments. An LN invoice consists of a destination node ID, a label, a creation timestamp, an expiry timestamp, a CLTV (Check Lock Time Verify) expiry timestamp and a payment hash. Paying an invoice with a randomized payment hash is possible (since the routing nodes are yet oblivious to the actual hash) and will route the payment successfully to its' destination, which forms the basis of this attack. Optionally it can contain an amount (leaving this field empty would be equal in principle to a blank cheque), a verbal description, a BTC fallback address in case the payment is unsuccessful, and a payment route suggestion. This invoice is then encoded, signed by the payee, and finally sent to the payer.

Having received a valid invoice (e.g. through their browser or directly via e-mail), the payer can now either use the route suggestion within the invoice or query the network themselves, and then send the payment to the payee along the route which has been determined. In this section, we will use the c-lightning RPC interface via Python exclusively - the functions involved are `getroute()` [24] and `sendpay()` [25], which takes two arguments: the return object from a `getroute()` call for a given route, a given amount and a given riskfactor, as well as the payment hash. Using `sendpay()` on its own (meaning, with a random payment hash instead of data from a corresponding invoice) will naturally result in one of two following error codes:

- **204 (failure along route):** This error indicates that one of the hops was unable to forward the payment to the next hop. This can be either due to insufficient funds or a non-existent connection between two adjacent hops along the specified route. If we have ensured that all nodes are connected as depicted in Figure 2, we can safely assume the former. One sequence of events leading up to this error can be seen in Figure 3.
- **16399 (permanent failure at destination):** Given the absence of a 204 error, the attempted payment has reached the last hop. As we are using a random payment hash, realistically the destination node will throw an error, signalling that no matching preimage has been found to produce the payment hash. The procedure to provoke a 16399 error code can be seen in Figure 4.

The goal of this attack is to trace payment flow over a channel, which the attacker node is directly or indirectly connected to. The attacker node will therefore initially attempt to determine whether a payment has occurred over the observed channel between the penultimate and final node along the route. To this end, the attacker will send out periodic probes to the final node (the "victim"), containing the amount which has been determined by the initial probe. If channel weights remain unaltered, each of these probes should return a 16399 error code. If a payment does occur however, the penultimate node will find itself unable to forward the payment on the outgoing channel to our target, yielding a 204 error response. Upon receiving this message, we can then restart the process of our initial probe and ultimately arrive at the exact amount of millisatoshis (msat), which have been transferred.

4.2 Lab Implementation

Recalling Figure 2, we have chosen Node 3 as our attacker node and Node 4 as our target node - hence, the initial goal of Node 3 is to determine the maximum payment flow between Nodes 2 and 4. To conduct our tests, each of the channels has been set up with a balance of 200,000,000 msat, with each node holding a stake of 100,000,000 msat in each of its channels. Node 3 will hold a slightly higher balance in order to accommodate probing fees. We can use the total channel balance, as received via gossip, as an upper ceiling for this value (200,000,000 msat in this case). We can then send payments from Node 3 to

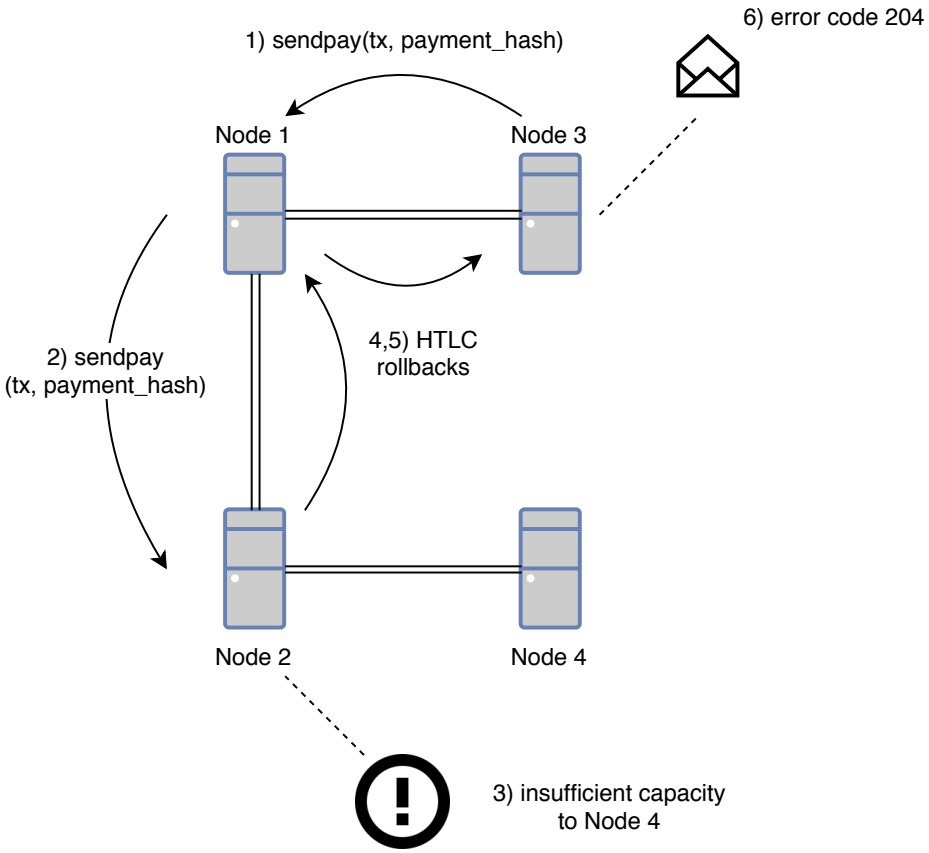

Fig. 3. Causing a 204 error by trying to send a payment to Node 4, which Node 3 is unable to perform. [20]

Node 4 with random payment hashes - resulting in either error code 16399 or error code 204 (Section 4.1). To this end, we perform a binary search on the available funds which we can transfer, searching for the highest value yielding a 16399 error instead of a 204 error. The algorithms used for both initial probing and deriving the actual channel balance from Node 2 to Node 4 are depicted in Algorithms 1 and 2.

We thus arrive at the approximate maximum amount, which Node 2 can transfer to Node 4. The next step is to continuously probe for this amount of msat in regular intervals. The expected response is a 16399 error code, with a 204 error code implying that the amount we are trying to send is higher than the available amount which Node 2 can transfer to Node 4 (or that it has disconnected from Node 4). Upon receiving a 204 response, we start looking for the

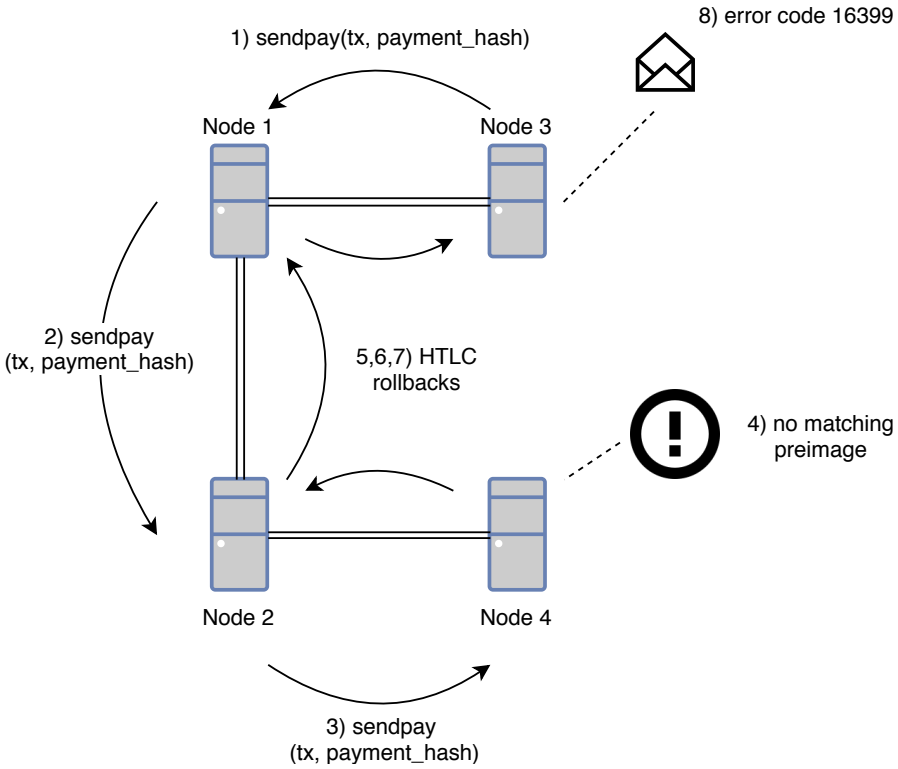

Fig. 4. Causing a 16399 error by trying to send a payment to Node 4, who can't produce a matching preimage and thus fails the payment. [20]

maximum payable amount to Node 4 once more. Subtracting the new amount from the old amount, we arrive at the size of the transaction which has occurred between Nodes 3 and 4. After 17 probes by Node 3, Algorithm 2 has yielded an initial balance of 99,999,237 msat, which is in line with the channel balance we have allocated between Nodes 2 and 4. The next step is to monitor the channel for potential weight changes (Algorithm 3). To verify this, we have transferred 50,000,000 msat from Node 2 to Node 4, with our program detecting this soon after (we have set t to 5 seconds in order to avoid excessive probing) and returning an updated balance of 49,998,237 msat. We then transferred another 30,000,000 msat from Node 1 to Node 4, with our program again picking up the change and reporting the new channel balance at 19,997,389 msat.

Figure 5 shows the trade-off between probing run time and the error in the channel balance estimate we observed for test runs on our lab setup. As we wanted to avoid overly excessive probing while conducting our tests, we were generally satisfied with any answer which is less than 1000 msat (the actual minimum BTC denomination) lower than the actual channel balance. Another possible

Algorithm 1: Probing a channel for a given amount of msat

```
Result: Either error code 204 or 16399
payment_hash = random.hex();
node_id = node ID of final node on victim channel;
msat = value to probe for;
route = getroute(node_id, msat);
sendpay(route, payment_hash);
```

Algorithm 2: Finding the initial maximum channel balance

```
Result: amount_msat - initial channel balance
min_msat = 0;
max_msat = channel.balance;
amount_msat = channel.balance / 2;
while True do
    if probe(amount_msat) == 16399 then
        min_msat = amount_msat;
    else
        if amount_msat) == 204 then
            max_msat = amount_msat;
        else
            return "No suitable route found.";
        end
    end
    if max_msat - min_msat ¡ 1000 then
        return amount_msat;
    else
        // continue to minimise maximum error
    end
    amount_msat = (min_msat + max_msat) / 2
end
```

approach could be keeping the number of probes sent out to the target constant, hence providing a more uniform level of balance error and probing duration.

4.3 BTC Testnet Evaluation

For analysis on the feasibility of our attack over the BTC Testnet, we connected Nodes 1, 2 and 3 from Figure 2 to the "ion.radar.tech" Testnet Lightning node. We chose this host in particular, since their website allowed us to alter the channel weights by generating payable invoices with parameters of our choosing. The exact connections along with the corresponding channel weights can be seen in Figure 6. Our goal was to verify the results we obtained in §4.2 and see whether probing duration (see Figure 5) was affected by the public Testnet hop in place of the local hop(s) used in §4.2. Running an initial series of probes from Node

Algorithm 3: Finding the initial maximum channel balance

```
Result: New maximum flow from penultimate to final node
init_max = initial channel balance;
new_max = init_max;
t = time to wait between checks;
while True do
    sleep(t);
    if probe(init_max) == 204 then
        // channel balance has decreased
        return find_init_max();
        // potentially calculate delta
    else
        if (init_max + 1000) == 16399 then
            // channel balance has increased
            return find_init_max();
            // potentially calculate delta
        else
            return error;
        end
    end
end
```

3 to Node 1, we arrived at a channel balance of 149,926,757 msat between the radar.ion.tech node and Node 1 (99.95% accuracy). We attribute this comparatively high error in regard to our tests in §4.2 due to the Testnet nodes' differing fee structure, which is necessarily taken into account when constructing the payment route. Then, we sent a payment containing 50,000,000 msat from Node 2 to Node 1 - predictably, Node 3 returned the updated maximum payment flow on the observed channel correctly with 99,902,343 msat (99.9 % accuracy).

After verifying the correct operation of our program for 16399 error codes, we were keen on discovering whether 204 error code scenarios would be dealt with correctly as well. In order to test this, we transferred back any amounts which have been redistributed as part of our initial test, increased the channel balance between Node 1 and radar.ion.tech by a factor of 10 and modified the setup from Figure 6 slightly by placing an intermediary hop between radar.ion.tech and Nodes 2 and 3. The updated infrastructure can be seen in Figure 7.

It became apparent however, that we would need to rethink the weights we allotted to the respective nodes, as we were initially unaware of the true channel weights between the radar.ion.tech and "lnd.vanilla.co.za" nodes. Naturally, we were inclined to simply run the `find_init_max()` function (Algorithm 2) from Node 3 on the ion.radar.tech node. However, we found that the two nodes were connected by 6 channels rather than one. To circumvent this route ambiguity, we queried a route for 1,000, 1,000,000 an 1,000,000,000 msat using default param-

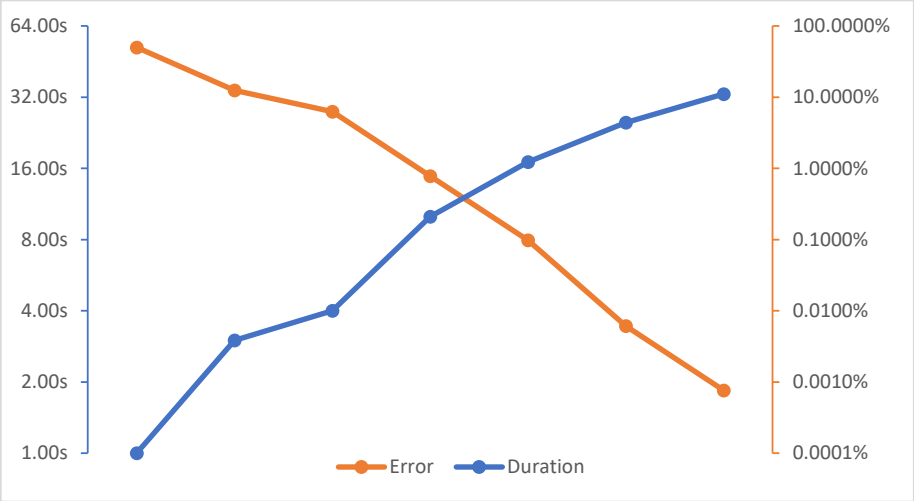

Fig. 5. Visualizing the trade-off between probing accuracy and duration. [20]

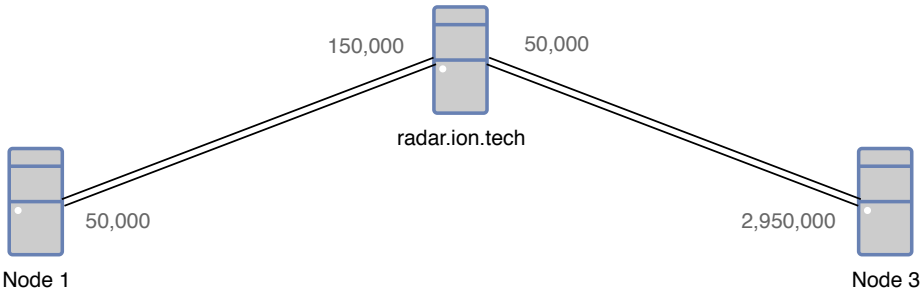

Fig. 6. Setup and balance allocations of our first testnet evaluation (balances given in satoshis).

eters, hoping all of them would return the same route, thus allowing us to treat the resulting channel as the only one connecting these two nodes. Unfortunately though, we received varying responses for all of these amounts, introducing a large uncertainty in any subsequent measurements. We then tried to run our tests on these channels, with all of them reporting failure in establishing a route to the target. We are not sure why even the initial probes failed and only further analysis and testing of our program will unveil the error in our approach. We decided to conclude our Testnet evaluation at this point, since despite extensive refactoring, we were not able to produce meaningful results for this constellation of nodes and channels, leaving route ambiguity and handling of multiple channels to be explored by further research in this area.

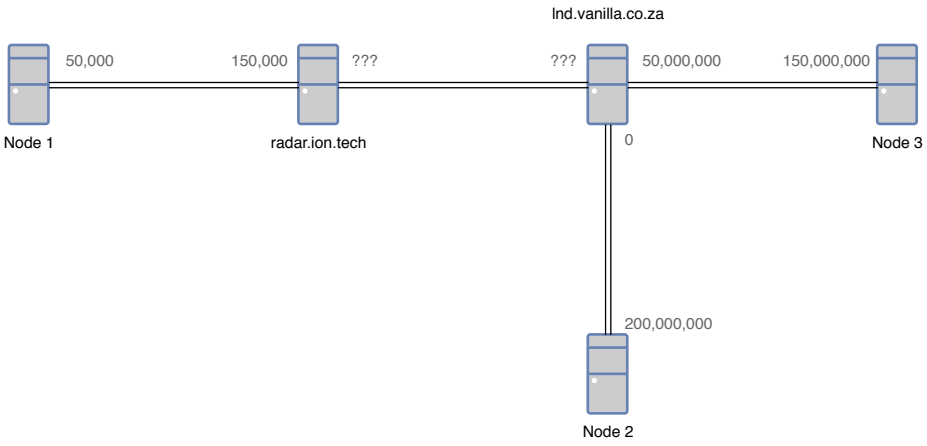

Fig. 7. Initial setup and balance allocations of our second Testnet evaluation (balances given in satoshis).

4.4 Results, Implications, and further Considerations

In Section 4.2 we have demonstrated that it is in fact possible to trace channel payments if the network is structured in a certain way. In theory, this method should hold true for any node which is reachable from the attacking node and has only one channel whose balance is lower or or equal to the second lowest balance on the route from the attacking node. We have partially verified this supposition in §4.3 while maintaining a high accuracy in our successful measurements. This is particularly a threat to end users, since most of them connect to a single well-connected node over a single channel, in order to interact with the rest of the network [1]. Nonetheless, there are several caveats to this method, the most significant of which are:

- **Excluding the possibility of payment forwarding:** The attack laid out in this chapter does not take into account the fact that nodes can be used to forward payments. Hearkening back to Figure 2, if we were to select the channel between Nodes 1 and 2 as our target, transactions between Nodes 1 and 4 would appear as if they were transactions to Node 3. One opportunity of accounting for this would be to monitor *every* channel to and from Node 3 for changes in directed channel balance, which would create problems on its own (see below).
- **Surge of unresolved HTLCs while probing:** Recalling steps 5-7 in Figure 4, each probe sets up a chain of irredeemable HTLCs (since a matching preimage would have to be brute-forced). Eventually, running multiple probes over the same channels will escrow its funds in these HTLCs, effectively DOSing the probe route and forcing the nodes to wait until the HTLCs time out before being able to forward other payments. This is an issue we encountered over and over during §4.2 and §4.3, often giving us one shot at

probing before having to wait multiple hours for the HTLCs to expire. This is also why we chose the channels leading up to our final target to have a much higher balance, so that we would have enough balance left after initial probing to monitor the channel for a reasonable period of time.

- **Insufficient sensitivity for high-frequency transactions:** Looking back at Algorithm 3, we have defined the parameter t as the time, for which to wait during probes for monitoring the channel balance, one the initial maximum value has been discovered. If more than one transaction would occur during this timeframe, it would still only show up as a singular payment with our tool. In the worst-case scenario, two transactions covering the same amount could take place in opposite directions, not changing the weighted balance at all and thus eluding our detection mechanisms.
- **Omission of private channels:** Upon creating a channel, the node can declare the channel as private, and thus prevent it from being broadcast via gossip. The channel is fully functional for both nodes which are connected by it, but no foreign payments can be routed through it. Looking ahead to increasing adoption of the Lightning Network, this provides an intriguing opportunity for nodes, which do not wish to participate in routing (e.g. mobile wallets) or nodes with limited uptime (personal computers). Routing would only occur between aggregating nodes (such as payment providers), with most of the channels (and therefore nodes) on the network remaining invisible to malicious participants as the gossip protocol would only propagate public channels. This further exacerbates our ability to detect forwarded payments (see above) as opposed to actual payments, since private channels can't be monitored by design.
- **Disregard of potential bottlenecks:** The proposed method of monitoring channel transactions does not hold, if a single channel along the route has a lower balance than the target channel in the desired direction. The node which has an insufficient amount of msat on its' outgoing channel would return a 204 error (Figure 3) This can often happen if an end user node is used as a hop prior to a high-capacity node. It is easy to detect which channel acts as a bottleneck, however a bit trickier to circumvent this obstacle - we would like to point the interested reader to [26] for suggestions on route hijacking and thus effectively bypassing the bottleneck along the route.

During the tests we conducted in §4.3, we also encountered the hops between Node 1 and Node 3 being connected via multiple channels. As confirmed by our observations, it is entirely possible to receive varying routes for differing amount_msat, riskfactor, cltv and fuzzpercent [24] combinations. Our tool failed to produce accurate results in this scenario, as it was designed assuming singular channels between pairs of nodes. It is however perfectly reasonable to have multiple channels between two nodes, as channel balances are final and can't be increased after creation. 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.

All in all, the probing attack we laid out in this chapter can be seen more as a proof of concept rather than a realistic attack vector, due to the limitations discussed in §4.4. We are confident that certain aspects such as the exact algorithm and route construction could be refined to provide more reliable results. However other aspects such as the binding of channel funds in irredeemable HTLCs and the incomplete network view due to private channels provide a much more consistent barrier to uncovering payment flows in real-world scenarios.

5 Timing Attack

5.1 Design

The Lightning Network is often referred to as a payment channel network (PCN). Performing payments over multiple hops is possible due to the use of HTLC's [21], a special bitcoin transaction whose unlocking conditions effectively rid the Lightning Network and its users of all trust requirements. An exemplary chain of HTLCs along with their shortened unlocking conditions is shown in Figure 8. Note that any node can only retrieve the funds locked in the HTLCs if they share R, and that each HTLC starting from Node 4 is valid for 2 hours longer than the previous HTLC to provide some room for error/downtime.

Due to the Onion Routing properties of the Lightning Network, it is cryptographically infeasible to try and determine where along the route a forwarding node is located, since each node can only decrypt the layer which was intended for it to decrypt. Attempts to analyze the remaining length of the routing packet have been thwarted at the protocol level by implementing a fixed packet size with zero padding at the final layer [14].

The only opportunity left to analyze the encrypted traffic between the nodes is to extract time-related information from the messages. One possibility would be to analyze the `cltv_expiry_delta` field (analogous to "hours passed" in Figure 8, measured in mined blocks since the establishment of the HTLC): By looking at the delay of both the incoming and the outgoing HTLC, a node could infer how many hops are left until the payment destination. However, this possibility has been accounted for by the adding "shadow routes" to the actual payment path, with each node fuzzing path information by adding a random offset to the `cltv_expiry_delta` value, hence effectively preventing nodes from guessing their position along the payment route [15].

The method we propose, is to time messages at the network level, rather than at the protocol level (e.g. through `cltv_expiry_delta`). Recalling Figure 8, Node 2 can listen for response messages from Node 4, since there is currently no mechanism in place to add delay to `update_fulfill_htlc` responses (in fact, [13] states that "*a node SHOULD remove an HTLC as soon as it can*"). Based on response latency, Node 2 could infer its position along the payment route to a certain extent, as examined in Section 5.2.

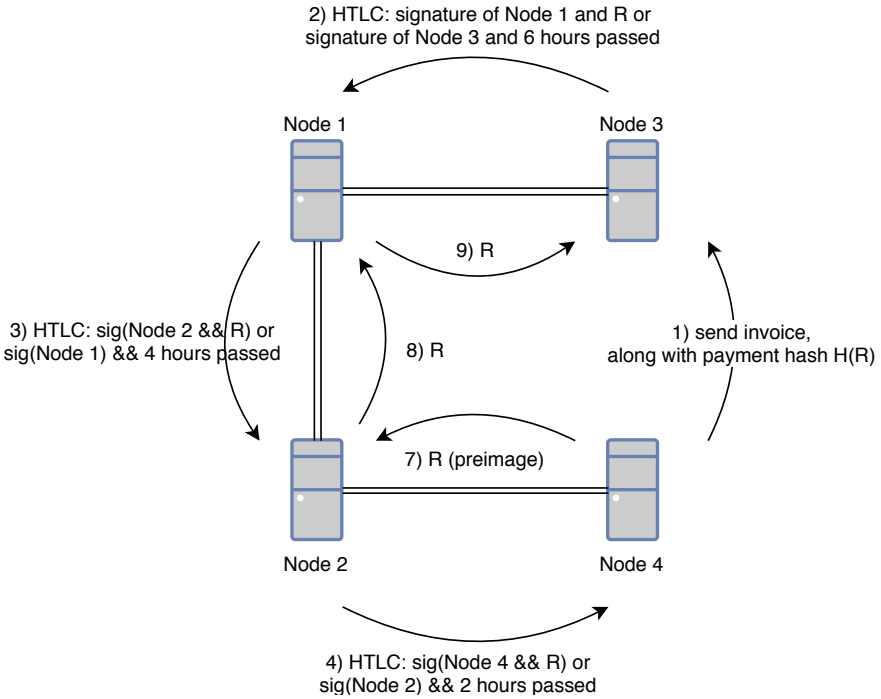

Fig. 8. Paying a LN invoice over multiple hops. Messages 2-4 are `update_add_htlc` messages, messages 7-9 are `update_fulfill_htlc` messages. [20]

5.2 Lab Implementation

Initial analysis has shown that analyzing packets directly (e.g. via Wireshark) is of little avail, since LN messages are end-to-end encrypted - meaning that even if we know the target nodes' IP address and port number, we can not detect the exact nature of the messages exchanged. We hence chose to redirect the output of the listening c-lightning node to a log file, which we then analyze with a Python script. As in Section 4, the source code can be found at [19].

Looking at the log file, we are particularly interested in the two messages discussed in Section 2: `update_add_htlc` and `update_fulfill_htlc`. The node output includes these events, complete with timestamps and the corresponding node ID with which the HTLC is negotiated. By repeatedly sending money back and forth between Nodes 1 and 3 in our test setup (Figure 2), we arrive at a local (and therefore minimum) latency of 182ms on average. The latency distribution for small (1,000 msat) payments can be seen in Figure 9. We have found that latencies remain largely unaffected by transaction size - increasing payment size by a factor of 100,000 actually slightly reduced average settlement time and standard deviation (Figure 10).

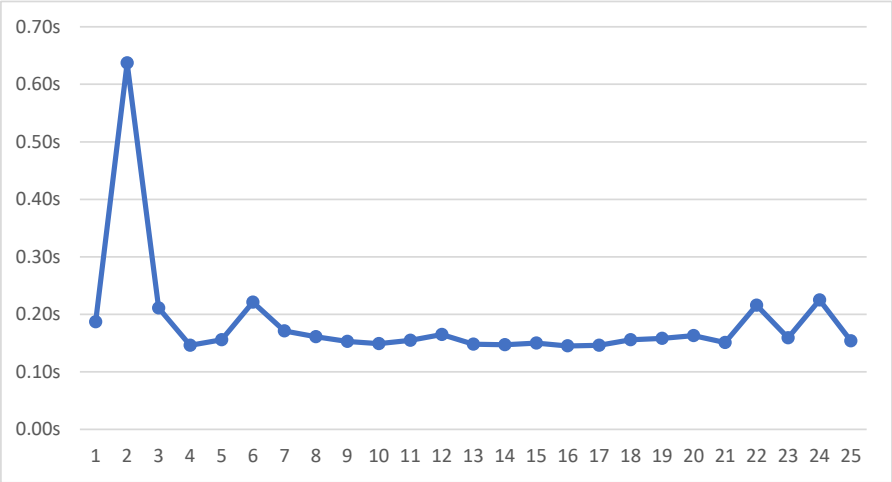

Fig. 9. Latency times for local payments containing 1,000 msat ($\mu = 0.1892$, $\sigma = 0.1168$, $n = 25$) [20]

Next, we examined whether an increase in hop distance would yield predictable results. To this end, we first timed payments over 1 network hop from Node 2 to Node 1 (Figure 11). Then, we timed payments over the same amount over 1 network and 1 local hop from Node 2 to Node 3 (Figure 12). Based on these results, we derive that timing messages on a local network with little to no interfering traffic scales predictably over several hops, with 1 network hop roughly corresponding to 1.284 local hops in terms of latency.

5.3 BTC Testnet Evaluation

Building on the results obtained in §5.2, we were keen to discover whether the they would carry over into real-world evaluations. To this end, we connected Node 1 and Node 3 from Figure 2 to the "endurance" Lightning Testnet node. Located in Dublin, Ireland and being connected to over 500 other Lightning Testnet nodes [1], we concluded that this node would provide a good entry point to test network latency from our location in Vienna, Austria, with the possibility to construct longer and more complicated routes over it as we saw fit. In order to constitute an initial RTT value, we established an HTTP connection to Lightning's default port 9735 [12], since the target host appeared to drop our ICMP ping requests. Alternating our requests between Systems A and B (Figure 2) in an attempt to prevent cached responses, we have found that HTTP response times were fairly constant from this node, with an average response time of 0.067s ($\sigma = 0.0206$).

Next, we were interested whether payments over the public hop were subject to an equally uniform latency as in §5.2. Thus, we created 25 invoices over 1,000,000

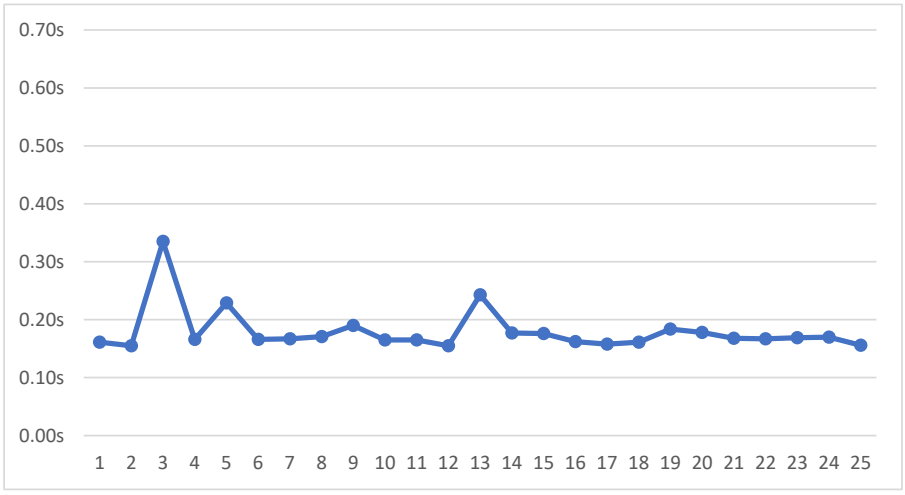

Fig. 10. Latency times for local payments containing 100,000,000 msat ($\mu = 0.1798$, $\sigma = 0.0385$, $n = 25$)

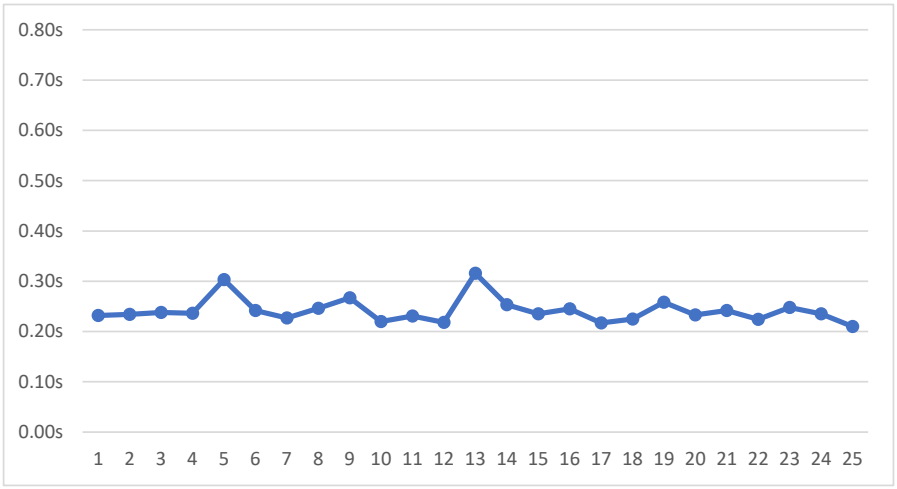

Fig. 11. Latency times for payments containing 100,000,000 msat over 1 network hop ($\mu = 0.234$, $\sigma = 0.025$, $n = 25$)

msat each (having found in §5.2 that response latency is independent of payment size) at Node 3 and sent the payment from Node 1. As seen in Figure 13, the fulfill message response times were remarkably consistent, however latency did not scale to our expectations. Based on Figure 12, we expected to be overall latency to be in the ballpark of 0.5-0.7 seconds (2x local network RTT + HTTP request RTT), however actual latency was twice that value. Results from the aranguren.org Testnet node, located in Melbourne, Australia, proved equally consistent with an average ping time of 0.314s ($\sigma = 0.035$) and an average HTLC

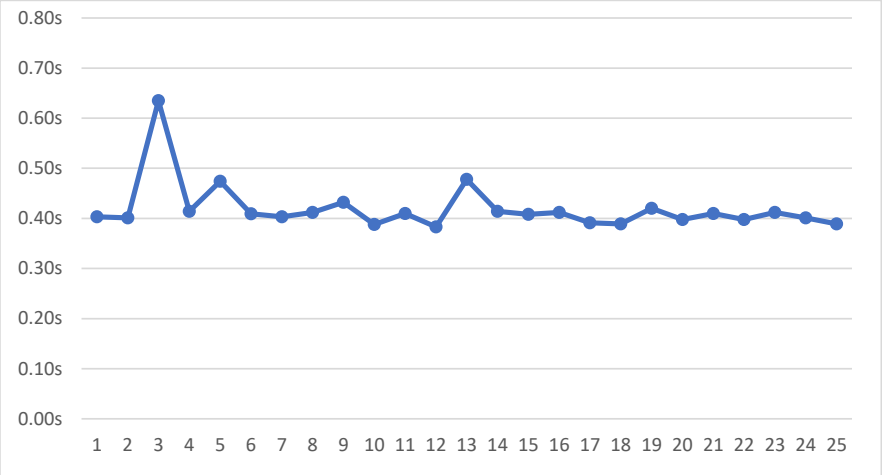

Fig. 12. Latency times for payments containing 100,000,000 msat over 1 network hop and 1 local hop ($\mu = 0.414$, $\sigma = 0.05$, $n = 25$)

fulfillment latency of 1.68s ($\sigma = 0.0972$)

Finally, we were curious about HTLC fulfillment delays over 2 public hops. To this end, we closed the channel between Node 3 and endurance and opened a new channel to the "aranguren.org" Testnet node, which in turn has a channel with endurance and thus re-establishes the chain of channels from Node 1 to Node 3. Timing results for this route can be seen in Figure 14. This marked the end of our timing tests, since we were not able to establish an acyclic payment route over 3 or more publicly available LN nodes. This coincides with the observation that neither the attempted nor the actual payments we performed during the course of §4 and §5 were routed over more than two public hops.

5.4 Results, Implications and further Considerations

Considering the findings in Section 5.2, we can see that timing produces fairly reliable and uniformly distributed results over a local network with little outside interference. Yet, due to the nature of LN routing, it is not possible to determine the distance or path to the initial payment source. To our surprise however, RTT remained equally consistent over 1-2 internet hops. Data acquired during monitoring of the local (mostly idle) network suggests that the timing node won't be able to distinguish traffic originating from a local node from the traffic in Section 5.2 without further information due to low latency deltas ranging from 2ms to 5ms.

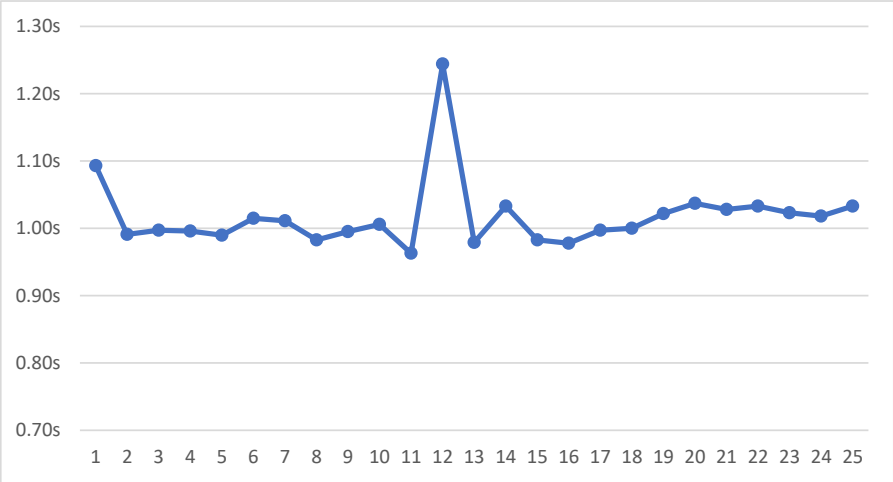

Fig. 13. Latency times for payments between Node 1 and Node 3 over the endurance Lightning node ($\mu = 1.0179$, $\sigma = 0.0542$, $n = 25$)

While performing timing measurements for payments across the BTC Testnet network, we have found that HTLC settlement takes long enough over even 1 hop to make traffic RTT volatility negligible. Over 1 hop, we conclude that HTLC settlement for our Vienna-based node should be in the ballpark of 0.86 - 1.97 seconds with 2-hop latency amounting to roughly 1.99 - 2.68 seconds, depending on the geographical location and assuming a normal distribution for the measured latency deltas. Further research could include a further statistical examination of the ability to differentiate distances for HTLC deltas at the sub-2-second threshold. We suggest that overall network bandwidth does not affect the acquired results significantly, since after performing all payments in §4, Node 1 has sent 64 KB and Node 3 has received 55 KB - only a fraction of which were outgoing/incoming HTLCs (alongside gossip, pings, etc.).

Our results open many new avenues for further timing-based research on the Lightning Network. The next step for us would be to develop a tool to predict the distance to the final destination of an HTLC which is passing through the listening node, based on the measurements laid out in §5.3. It would be interesting to see whether there is a possibility to force payment-unrelated response messages, e.g. by forging ping messages [12] in order to estimate (possibly network-wide) RTTs, correlate HTLC settlement latencies against them and finally arrive at a set of nodes which must have been the ultimate recipient of the forwarded payment. Furthermore, experiments could be conducted on the feasibility of adding a random time offset to HTLC fulfillments, and the trade-offs involved therein.

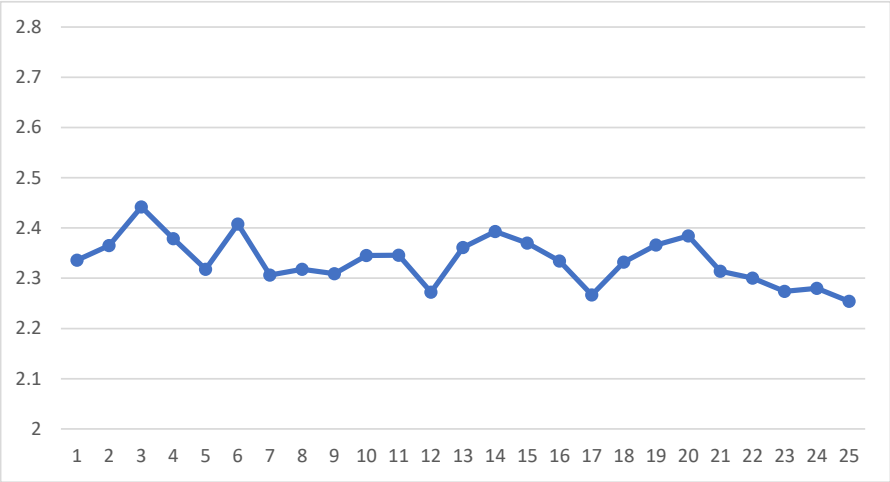

Fig. 14. Latency times for payments between Node 1 and Node 3 over the endurance and aranguren.org Lightning nodes ($\mu = 2.3349$, $\sigma = 0.0475$, $n = 25$)

6 Conclusion

This paper has shown that off-chain routing and payment settlement mechanisms may be exploited to infer confidential information about the network state. In particular, considering the Lightning Network with Bitcoin as the underlying blockchain as a case study, we set up a local infrastructure and proposed two ways in which two current state-of-the-art implementations, c-lightning and LND, can be exploited to gain knowledge about distant channel balances and transactions to unconnected nodes: By deliberately failing payment attempts, we were able to deduce the exact amount of (milli-)satoshis on a channel located two hops away on our local lab infrastructure. Using this technique repeatedly, we were able to determine whether a transaction occurred between one node and another over the monitored channel. To a certain extent, we were able to reproduce these results in the public Bitcoin Testnet chain. We also identified this attacks' limitations and proposed some workarounds to these obstacles.

By timing the messages related to HTLC construction and termination, we were able to infer the remaining distance of a forwarded packet accurately in our test lab. These results transferred well into our Testnet evaluation, while being free of the partially restrictive limitations which we discovered during our examination of the probing attack. We concluded that RTT volatility of the HTLC message cycle was low enough for public Testnet hops which were within geographical vicinity to our node in Vienna, Austria, as well as for hops which were located in East Asia, to establish latency approximate latency boundaries for the number

of remaining hops along the payment route of a forwarded transaction.

Our work raises several interesting research questions. In particular, it remains to fine-tune our attacks, to improve the flexibility of our software tools and to finally conduct more systematic experiments including more natural/interconnected network topologies, particularly on other off-chain networks. More generally, it will be interesting to explore further attacks on the confidentiality of off-chain networks exploiting the routing mechanism and investigate countermeasures. Furthermore, our work raises the question whether such vulnerabilities are an inherent price of efficient off-chain routing or if there exist rigorous solutions.

Bibliographical Note

A preliminary version of this article appeared at ICISSP 2020 [20].

References

1. 1ML - Bitcoin Lightning Analysis Engine. https://1ml.com/ (2019), [Online; accessed 10-November-2019]
2. c-lightning GitHub Repository. https://github.com/ElementsProject/lightning (2019), [Online; accessed 26-December-2019]
3. LND GitHub Repository. https://github.com/lightningnetwork/lnd (2020), [Online; accessed 18-January-2020]
4. Antonopoulos, A.M.: Mastering Bitcoin: Unlocking Digital Crypto-Currencies. O'Reilly Media, Inc., 1st edn. (2014)
5. Antonopoulos, A.M., Osuntokun, O., Pickhardt, R.: Mastering the Lightning Network. https://github.com/lnbook/lnbook (2019), [Online; accessed 22-November-2019]
6. Béres, F., Seres, I.A., Benczúr, A.A.: A cryptoeconomic traffic analysis of bitcoins lightning network. arXiv **abs/1911.09432** (2019)
7. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: IEEE Symposium on Security and Privacy. pp. 269–282. IEEE Computer Society (2009)
8. Fugger, R.: Money as IOUs in social trust networks & a proposal for a decentralized currency network protocol. Hypertext document. Available electronically at http://ripple. sourceforge. net **106** (2004)
9. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Off the chain transactions. IACR Cryptology ePrint Archive **2019**, 360 (2019)
10. Herrera-Joancomartí, J., Navarro-Arribas, G., Pedrosa, A.R., Pérez-Solà, C., García-Alfaro, J.: On the difficulty of hiding the balance of lightning network channels. In: AsiaCCS. pp. 602–612. ACM (2019)
11. Kate, A., Goldberg, I.: Using sphinx to improve onion routing circuit construction. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 6052, pp. 359–366. Springer (2010)
12. Lightning Network: BOLT 1: Base Protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/01-messaging.md (2019), [Online; accessed 23-January-2020]

13. Lightning Network: BOLT 2: Peer Protocol for Channel Management. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md (2019), [Online; accessed 6-January-2020]
14. Lightning Network: BOLT 4: Onion Routing Protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md (2019), [Online; accessed 3-January-2020]
15. Lightning Network: BOLT 7: P2P Node and Channel Discovery. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md (2019), [Online; accessed 4-December-2019]
16. Lightning Network: BOLT 8: Encrypted and authenticated transport. https://github.com/lightningnetwork/lightning-rfc/blob/master/08-transport.md (2019), [Online; accessed 4-January-2020]
17. Lightning Network: Lightning Network Specifications. https://github.com/lightningnetwork/lightning-rfc/ (2019), [Online; accessed 29-November-2019]
18. Lightning Network: Lightning RFC: Lightning Network Specifications. https://github.com/lightningnetwork/lightning-rfc (2019), [Online; accessed 18-November-2019]
19. Nisslmueller, U.: Python code repository. github.com/utzn42/icissp_2020_lightning (2020), [Online; accessed 02-January-2020]
20. Nisslmueller, U., Foerster, K.T., Schmid, S., Decker, C.: Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. In: Proc. 6th International Conference on Information Systems Security and Privacy (ICISSP) (2020)
21. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf (2016), [Online; accessed 3-January-2020]
22. Raiden Network: Raiden Network. https://raiden.network/ (2020), [Online; accessed 02-January-2020]
23. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In: EuroS&P Workshops. pp. 347–356. IEEE (2019)
24. Russell, R.: lightning-getroute – Command for routing a payment (low-level). https://lightning.readthedocs.io/lightning-getroute.7.html (2019), [Online; accessed 6-December-2019]
25. Russell, R.: lightning-sendpay – Low-level command for sending a payment via a route. https://lightning.readthedocs.io/lightning-sendpay.7.html (2019), [Online; accessed 4-January-2020]
26. Tochner, S., Schmid, S., Zohar, A.: Hijacking routes in payment channel networks: A predictability tradeoff. arXiv **abs/1909.06890** (2019)
27. Wang, P., Xu, H., Jin, X., Wang, T.: Flash: efficient dynamic routing for offchain networks. In: CoNEXT. pp. 370–381. ACM (2019)