Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks

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*

Keywords: Cryptocurrencies, Bitcoin, Payment Channel Networks, Routing, Privacy

Abstract: Cryptocurrency off-chain networks such as Lightning (e.g., Bitcoin) or Raiden (e.g., Ethereum) aim to increase the scalability of traditional on-chain transactions. To support nodes to learn about possible paths to route their transactions, these networks need to provide gossip and probing mechanisms. This paper explores whether these mechanisms may be exploited to infer sensitive information about the flow of transactions, and eventually harm privacy. In particular, we identify two threats, related to an active and a passive adversary. The first is a *probing attack:* here the adversary aims the maximum amount which is transferable in a given direction of a target channel, by active probing. The second is a *timing attack:* the adversary discovers how close the destination of a routed payment actually is, by acting as a passive man-in-the middle. We then analyze the limitations of these attacks and propose remediations for scenarios in which they are able to produce accurate results.

1 INTRODUCTION

Blockchains, the technology underlying cryptocurrencies such as Bitcoin or Ethereum, herald an era in which mistrusting entities can cooperate in the absence of a trusted third party. However, current blockchain technology faces a scalability challenge, supporting merely tens of transactions per second, compared to custodian payment systems which easily support thousands of transactions per second.

Off-chain networks (Gudgeon et al., 2019), a.k.a. payment channel networks or second-layer blockchain networks, have emerged as a promising solution to mitigate the blockchain scalability problem: by allowing participants to make payments directly through a network of payment channels, the overhead of global consensus protocols and committing transactions on-chain can be avoided. Off-chain networks such as Bitcoin Lightning (Lightning Network, 2019f), Ethereum Raiden (Raiden Network, 2020), and XRP Ripple (Fugger, 2004), to just name a few, promise to primarily reduce load on the underlying blockchain, as well as completing transactions in subseconds and substantially reducing transaction fees.

In all these networks, each node typically 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 use source routing, in which 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 paths which support their transaction, i.e., have sufficient channel capacity.

However, the fact that nodes need to be able to find routes 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 (Lightning Network, 2019c). *Probing* is the mechanism which is used to construct a payment route based on a local network view delivered by gossip, and ultimately perform the payment. In the context of §3, 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 (Russell, 2019a). Because the gossip store contains global channel information, it is possible to query payment routes originating from any node on the network.

This paper explores the question whether the inherent need for nodes to discover routes in general,

Nisslmueller, U., Foerster, K., Schmid, S. and Decker, C.
Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks.
DOI: 10.5220/0009429200070014
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 7-14
ISBN: 978-989-758-399-5; ISSN: 2184-4356

and the gossip and probing mechanisms in particular, can be exploited to infer sensitive information about the off-chain network and its transactions.

1.1 Our Contributions

This paper identifies two novel threats for the confidentiality of off-chain networks. In particular, we consider the Lightning Network as a case study and 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. 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

Our paper is organized as follows. We introduce some preliminaries in §2, and then first describe the probing attack in §3 followed by the timing attack in §4. We review related work in §5 and conclude in §6.

2 PRELIMINARIES

While our contribution is more general, to be concrete, we will consider the Bitcoin Lighning 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 (Lightning Network, 2019e). 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 in particular are 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). Each Lightning node keeps track of the current channel set (the gossip store) and modifies it based on the gossip messages it receives from other nodes on the network. *Probing* is the mechanism which used to actually construct the payment route and then perform the payment. The gossip store is queried for viable routes to the destination, based on the desired route properties (Russell, 2019a). Because the gossip store contains global channel information, it is possible to query payment routes originating from any node on the network.

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 (Lightning Network, 2019b) and (Lightning Network, 2019d) 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 (Antonopoulos, 2014). 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).

The gossip messages mentioned earlier are sent

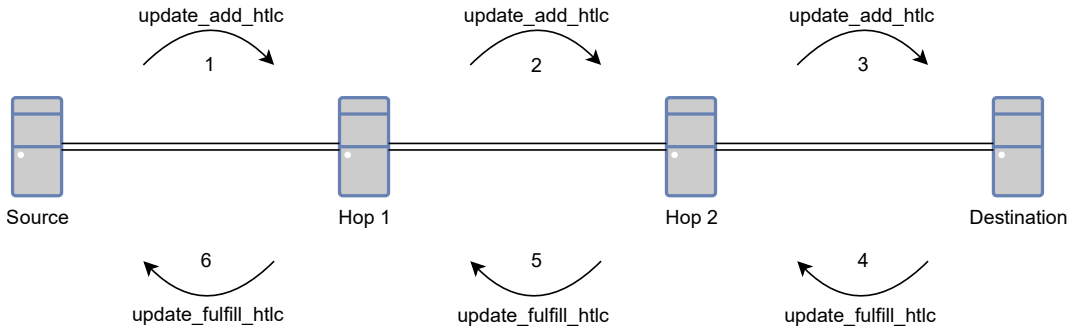

Figure 1: An exemplary transaction from source to destination, involving two intermediate nodes.

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 §3 and §4, we have set up a testing network consisting of four c-lightning nodes, with two local network computers running two local nodes each (Figure 2). Node 1 and 2 are connected via a local network link and can form hops for payment routes between nodes 3 and 4.

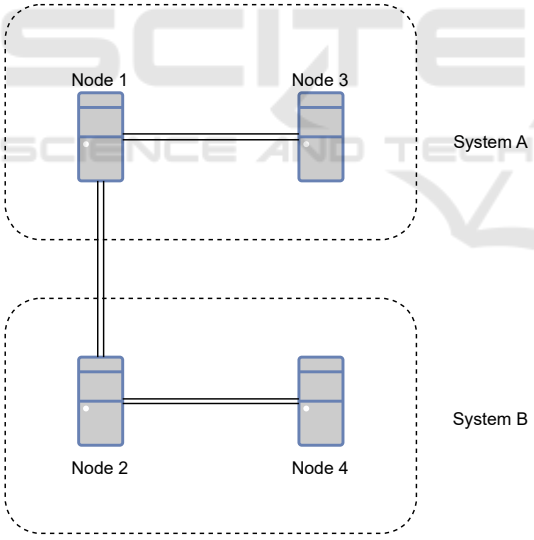

Figure 2: Local Testing Setup.

3 PROBING ATTACK

3.1 Design

The Lightning Network uses an invoice system to handle payments. A 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 probe 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 - the functions involved are `getroute()` (Russell, 2019a) and `sendpay()` (Russell, 2019b), which takes two arguments: the return object from a `getroute()` call for a given route, a given amount and a given riskfactor, and 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 nonexistent 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 destina-

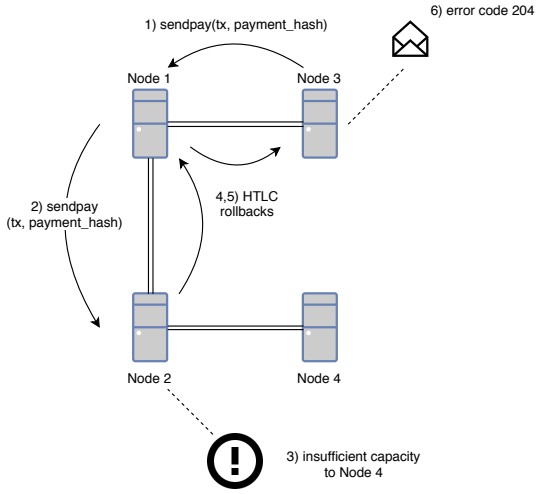

Figure 3: Causing a 204 error by trying to send a payment to Node 4, which Node 3 is unable to perform.

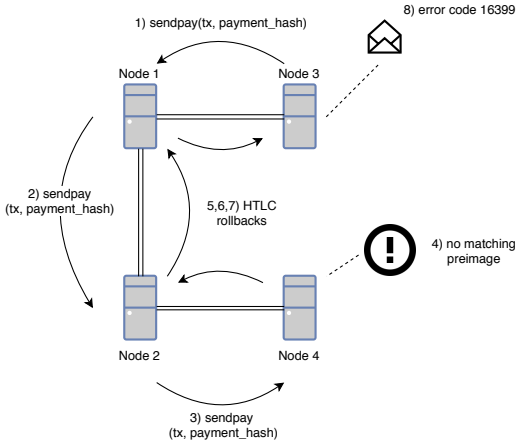

Figure 4: Causing a 16399 error by trying to send a payment to Node 4, who can't produce a matching preimage and thus fails the payment.

tion 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 trace payment flow over a channel, which the attacker node is directly or indirectly connected to. Recalling Figure 2, the goal of Node 3 will therefore be to determine whether a payment has occurred on the channel between Node 2 and Node 4. Each of the channels has a balance of 200,000,000 millisatoshis, with each node holding a stake of 100,000,000 millisatoshis in each of its channels. Node 3 will hold a slightly higher balance in order to accommodate probing fees.

3.2 Implementation

The goal of Node 3 is to determine the maximum payment size between nodes 2 and 4. We can use the total channel balance, as received via gossip, as an upper ceiling for this value (200,000,000 millisatoshis in this case). We can then send payments from Node 3 to Node 4 with random payment hashes - resulting in either error code 16399 or error code 204 (§3.1). 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.

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 millisatoshis 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 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.

Figure 5 shows the trade-off between probing run time and the error in the channel balance estimate. As we want to avoid overly excessive probing, we are satisfied with any answer which is less than 1000 msat lower than the actual channel balance. Another possible approach could be keeping the amount of probes constant, hence providing a more uniform level of balance error and probing duration.

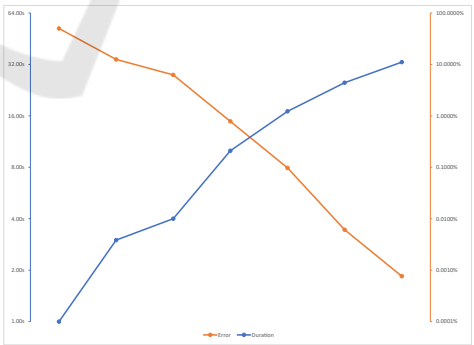

Figure 5: Visualizing the trade-off between probing accuracy and duration.

3.3 Results, Implications and Further Considerations

In §3.2 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. This is particularly a threat to end users, since most of them connect to a single well connected node, in order to interact with the rest of the network (1ML, 2019). Nonetheless, there are several caveats to this method, the most significant of which are:

- **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 nodes on the network invisible to malicious participants.
- **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. This can often happen if an end user node is used as a hop prior to a high-capacity node. The procedure is depicted in Figure 6.

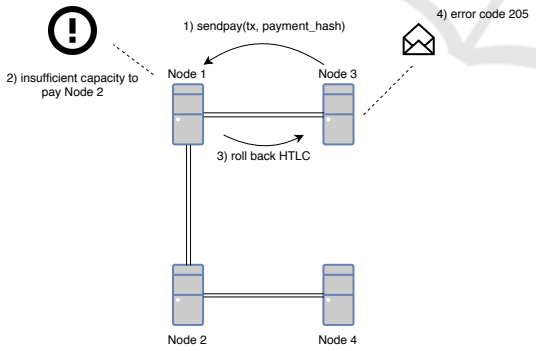

Figure 6: Trying to send a payment to Node 4, with Node 1 having an insufficient balance to Node 2 to cover the whole amount.

4 TIMING ATTACK

4.1 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 (Poon and Dryja, 2016), 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 7. 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.

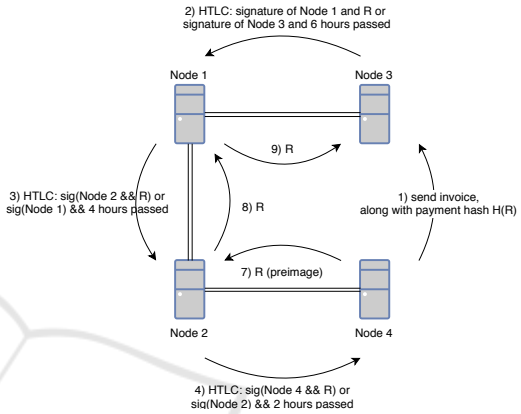

Figure 7: Paying a LN invoice over multiple hops. Messages 2-4 are `update_add_htlc` messages, messages 7-9 are `update_fulfill_htlc` messages.

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 (Lightning Network, 2019b).

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 7, 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 nodes 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 (Lightning Network, 2019c).

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 7, 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, (Lightning Network, 2019a) 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, examined in §4.2.

4.2 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 Python. As in §3.2, the source code can be found at (Nisslmueller, 2020).

Looking at the log file, we are particularly interested in the two messages discussed in §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 2, we arrive at a local (and therefore minimum) latency of 189ms on average. The latency distribution for small (1,000 msat) payments can be seen in Figure 8.

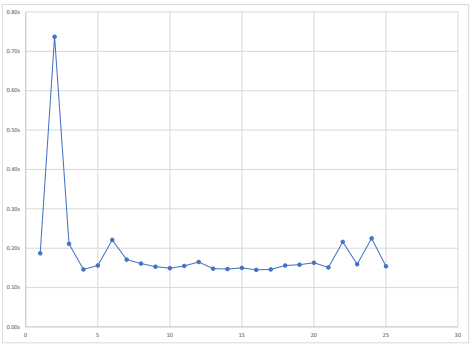

Figure 8: Latency times for local payments containing 1,000 msat ($\mu = 0.1892$, $\sigma = 0.1168$, $n = 25$).

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 9).

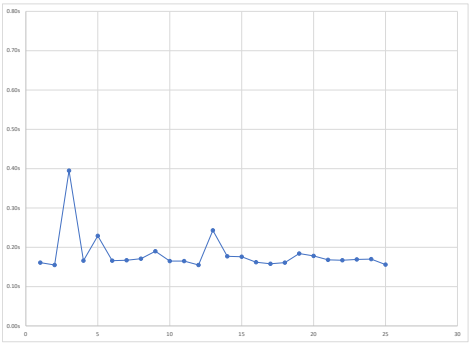

Figure 9: Latency times for local payments containing 100,000,000 msat ($\mu = 0.1822$, $\sigma = 0.049$, $n = 25$).

4.3 Results, Implications and Further Considerations

Considering the findings in §4.2, we can see that timing produces fairly reliable and uniformly distributed results over a local network with little outside interference, however, due to the nature of LN routing, it is not possible to determine the distance or path to the initial payment source. . Further research will have to show how confident the results produced by a timing node are, if the destination is 1 or more hops away. Data acquired during monitoring the local (mostly idle) network suggests that the timing node won't be able to distinguish local network traffic from the traffic in §4.2 due to low latency times ranging from 2ms to 5ms. It would be worth investigating the latency threshold at which timing can no longer produce accurate results.

5 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. (Gudgeon et al., 2019). The Lightning Network as an second-layer network alternative to pure on-chain transactions was first proposed by (Poon and Dryja, 2016), with the technical specifications laid out in (Lightning Network, 2019f). Despite being theoretically currency-agnostic, current implementations such as c-lightning (cli, 2019) support BTC exclusively. A popular alternative for ERC-20 based tokens is the Raiden Network (Raiden Network, 2020).

Several papers have already analyzed security and privacy concerns in off-chain networks. Rohrer et al. (Rohrer et al., 2019) focuses on channel-based attacks and proposes 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. (Tochner et al., 2019) 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. (Herrera-Joancomartí et al., 2019) 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., 2019) examines the LN routing process in more detail and proposes 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. (Béres et al., 2019) makes a 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, (Antonopoulos et al., 2019) is very close to (Antonopoulos, 2014) 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 (Lightning Network, 2019b). The version used in current Lightning versions is based on (Danezis and Goldberg, 2009) and (Kate and Goldberg, 2010), the latter of which also provides performance comparisons between competing protocols.

6 CONCLUSION

This paper has shown that off-chain routing mechanisms may be exploited to infer confidential information about the network state. In particular, considering the LN as a case study, we set up a local infrastructure and proposed two ways in which the current implementation c-lightning 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. Using this technique repeatedly, we were able to determine whether a transaction occurred between this node and another over the monitored channel. By timing the messages related to HTLC construction and termination, we were able to infer the remaining distance of a forwarded packet accurately.

Our work raises several interesting research questions. In particular, it remains to fine-tune our attacks and conduct more systematic experiments including more natural/interconnected network topologies, also 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.

REFERENCES

(2019). 1ML - Bitcoin Lightning Analysis Engine. https://1ml.com/. [Online; accessed 10-November-2019].

(2019). c-lightning GitHub Repository. https://github.com/ElementsProject/lightning. [Online; accessed 26-December-2019].

Antonopoulos, A. M. (2014). *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition.

Antonopoulos, A. M., Osuntokun, O., and Pickhardt, R. (2019). Mastering the Lightning Network. https://github.com/lnbook/lnbook. [Online; accessed 22-November-2019].

Béres, F., Seres, I. A., and Benczúr, A. A. (2019). A cryptoeconomic traffic analysis of bitcoins lightning network. *arXiv*, abs/1911.09432.

Danezis, G. and Goldberg, I. (2009). Sphinx: A compact and provably secure mix format. In *IEEE Symposium on Security and Privacy*, pages 269–282. IEEE Computer Society.

Fugger, R. (2004). 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.

Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., and Gervais, A. (2019). Sok: Off the chain transactions. *IACR Cryptology ePrint Archive*, 2019:360.

Herrera-Joancomartí, J., Navarro-Arribas, G., Pedrosa, A. R., Pérez-Solà, C., and García-Alfaro, J. (2019). On the difficulty of hiding the balance of lightning network channels. In *AsiaCCS*, pages 602–612. ACM.

Kate, A. and Goldberg, I. (2010). Using sphinx to improve onion routing circuit construction. In *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 359–366. Springer.

Lightning Network (2019a). BOLT 2: Peer Protocol for Channel Management. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md. [Online; accessed 6-January-2020].

Lightning Network (2019b). BOLT 4: Onion Routing Protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md. [Online; accessed 3-January-2020].

Lightning Network (2019c). BOLT 7: P2P Node and Channel Discovery. https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md. [Online; accessed 4-December-2019].

Lightning Network (2019d). BOLT 8: Encrypted and authenticated transport. https://github.com/lightningnetwork/lightning-rfc/blob/master/08-transport.md. [Online; accessed 4-January-2020].

Lightning Network (2019e). Lightning Network Specifications. https://github.com/lightningnetwork/lightning-rfc/. [Online; accessed 29-November-2019].

Lightning Network (2019f). Lightning RFC: Lightning Network Specifications. https://github.com/lightningnetwork/lightning-rfc. [Online; accessed 18-November-2019].

Nisslmueller, U. (2020). Python code repository. https://github.com/utzn42/icissp_2020_lightning. [Online; accessed 02-January-2020].

Poon, J. and Dryja, T. (2016). The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf. [Online; accessed 3-January-2020].

Raiden Network (2020). Raiden Network. https://raiden.network/. [Online; accessed 02-January-2020].

Rohrer, E., Malliaris, J., and Tschorsch, F. (2019). Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In *EuroS&P Workshops*, pages 347–356. IEEE.

Russell, R. (2019a). lightning-getroute – Command for routing a payment (low-level). https://lightning.readthedocs.io/lightning-getroute.7.html. [Online; accessed 6-December-2019].

Russell, R. (2019b). lightning-sendpay – Low-level command for sending a payment via a route. https://lightning.readthedocs.io/lightning-sendpay.7.html. [Online; accessed 4-January-2020].

Tochner, S., Schmid, S., and Zohar, A. (2019). Hijacking routes in payment channel networks: A predictability tradeoff. *arXiv*, abs/1909.06890.

Wang, P., Xu, H., Jin, X., and Wang, T. (2019). Flash: efficient dynamic routing for offchain networks. In *CoNEXT*, pages 370–381. ACM.