Finding Possible Receivers of Lightning Transactions by Using Timing Information

Robin Kouwenhoven
Student number: 4695534

to obtain the degree of MSc. Computer Science
at the Delft University of Technology

Software Technology track, MSc Computer Science
Faculty EEMCS
Delft University of Technology
Thesis committee: Stefanie Roos, Zekeriya Erkin
Defence date: 23 February 2024

Finding Possible Receivers of Lightning Transactions by Using Timing Information

Robin Kouwenhoven
4695534
Software Technology track, MSc Computer Science
Faculty EEMCS, Delft University of Technology
Supervised by Stefanie Roos
Thesis committee: Stefanie Roos, Zekeriya Erkin
Defence date: 23 February 2024

***Abstract*—Foremost among the challenges of the Bitcoin blockchain is the scalability bottleneck. To address this issue, the Lightning Network, a payment channel network, was created. Lightning is a payment channel network that is source-routed and uses onion routing, like Tor. However, unlike Tor, the routing path is determined by optimizing a cost function, which uses public information. In this paper, a timing attack is evaluated by simulation of the Lightning Network. The goal of the attack is for an adversary that is part of a payment to determine the destination of the payment. The influence colluding adversaries have on the performance of the attack is evaluated. Three types of colluding adversaries (closest to destination, farthest from destination, average of all adversaries) are compared to a simple adversary that guesses the next hop in the payment is the destination of the payment. It is found that the adversary closest to the destination performs the best.**

Furthermore, shadow routing, a mitigation against these types of attacks, is evaluated against this attack. It is found that shadow routing does not have a significant impact on the performance of this attack.

I. Introduction

Recently, the Bitcoin price has touched new records [1]. As of November 23rd, 2023 according to CoinMarketCap[1] the price of one Bitcoin lies around 37.000 USD. This indicates the success of Bitcoin. However, the remarkable growth of Bitcoin is not without its challenges. Foremost among these challenges is the inherent scalability bottleneck within the Bitcoin blockchain, impeding its ability to process a high volume of transactions quickly [2]. Due to the proof-of-work consensus protocol used by Bitcoin, only one block can be mined approximately every ten minutes. Reducing this block interval would reduce the computational work required to rewrite the blockchain, making it less secure. Thus, the number of transactions Bitcoin can handle is limited by the block size and block interval [3] [4]. The maximum number of transactions for Bitcoin lies at 10 transactions per second [2]. According to Visa, they process 269.8 billion transactions in a year [5]. On average, that is 8555 transactions per second, which does not account for traffic spikes.

To address the scalability issues in Bitcoin, payment channels can be used [6]. A payment channel is opened by creating an opening transaction that sends funds to a 2-of-2 multi-signature address. The participants in the 2-of-2 multi-signature transaction are the owners of the channel. Before the transaction is recorded on the blockchain, another transaction is created that spends this 2-of-2 multi-signature output and sends funds back to the endpoints of the channel. Once both parties have received and signed that transaction, the opening transaction can be recorded on the blockchain. The amounts sent back to the endpoints after recording the second transaction on the blockchain represent their balances. The second transaction is only recorded on the blockchain when the payment channel needs to close. During the lifetime of the payment channel, the parties can create a new transaction that spends the opening transaction to change the balances of the payment channel. They also need to revoke the earlier spending transaction. When the two endpoints have a dispute, one of the endpoints can simply record the spending transaction on the blockchain. Both parties will receive their balance back according to the latest agreed state. This improves scalability, as now only an opening and closing transaction need to be recorded on the blockchain. Furthermore, in theory the channel only needs to be closed when a dispute occurs or when someone wants their money back. All other transactions between the two endpoints will be handled off chain.

Still, users have to create a payment channel with everyone they want to pay, leading to transactions on the blockchain. The Lightning Network addresses this issue by allowing payments to be routed over multiple hops of payment channels [6]. This is achieved by using a cryptographic mechanism called an Hashed Timelock Contract (HTLC) in the output script of a transaction. An HTLC output can be spent in two ways: either the spender has to present the preimage of a hash specified in the HTLC, or the spender has to wait a few blocks until the timelock expires. The waiting time is specified by the first block index the transaction can be spent, called the block height. In both cases, the spender must also present a signature that can be validated by the specified public key. To allow the payment to either completely fail or completely succeed, the receiver of a transaction first shares a hash of a preimage only known by the receiver with the sender of the transaction. The sender creates an HTLC with the first hop, where the funds

[1]https://coinmarketcap.com/currencies/bitcoin/

are paid to the first hop upon showing the preimage of the hash that was shared by the receiver, or the funds can be reclaimed by the sender after a certain block height. The first hop does the same with the second hop and so on until the transaction reaches the destination. Now the destination claims the funds by sharing the preimage that it has kept secret. By sharing the secret, the last hop can also claim its funds. This will continue until all hops have claimed their funds and the payment is finished. In case one of the nodes acts malicious by not sharing the preimage with the previous node, the funds can be reclaimed after the block height specified in the HTLC.

Lightning uses source-routed onion routing [6]. The sender of a payment decides which intermediary nodes are used to forward the payment. Each intermediary charges a fee for forwarding a payment since the transaction value will not be spendable from their balance until either the HTLC timelock has expired or the preimage of the hash is known. The sender of the transaction therefore has to find a path where the fees are low and at the same time the transaction amount is locked for a short period in case a node acts malicious or the balance of the channels is insufficient. The fees nodes charge to use their channel are public knowledge, as well as the number of blocks each node adds to the waiting time for forwarding the payment. Although the protocol uses onion routing and the complete payment path is not known to the intermediate nodes, the routing algorithm tries to optimize for the fee amount and lock time using public information resulting in the privacy guarantees being worse than for example Tor[2]. Kumble *et al.* [7] have already shown that it is effective to exploit the public information of payment channels to find a transaction's sender or destination when that transaction is observed by the adversary. Rohrer *et al.* [8] used a timing attack to find possible senders and receivers.

A possible mitigation against the attack of [7] is shadow routing [7]. With shadow routing, the timelock value in the HTLC is changed by adding a random number to it obtained by extending the payment route[3]. Another mitigation would be to remove the ability to link messages to the same payment. This is done by Malavolta *et al.* [9]. They have introduced anonymous multihop locks (AMHLs). These locks could replace the HTLCs currently in use in the Lightning Network. An AMHL does not use the same hash for the transactions between different intermediate nodes. AMHLs would also work against [8].

The HTLC used in the Lightning Network not only makes it possible to link different messages to a payment but also for multiple adversaries to decide if they took part in the same payment. We built on the timing attack designed by Rohrer *et al.* [8] to decide if shadow routing is an effective mitigation against the attack and to evaluate how to effectively collude when an adversary compromises multiple nodes. A static honest-but-curious local internal adversary is considered that can only execute (probabilistic) algorithms that run in polynomial time. The adversary is an on-path adversary, meaning it will only try to find receivers of payments that it forwards itself. The simplest strategy used to determine the receiver of a payment assumes that the node the adversarial node forwards the payment to is the receiver of the payment. Additionally, the timing attack designed by Rohrer *et al.* [8] is used as an attack strategy. However, in contrast to Rohrer *et al.* [8], we use three different strategies for colluding adversaries to assess how to effectively collude. In all three methods, the nodes first execute the timing attack individually. In the first method, the adversary assumes that the adversarial node closest to the destination found the correct destination of the payment. In the second method, the adversary assumes that the adversarial node farthest from the destination found the correct destination of the payment. In the last method, the probabilities of a node being the destination of a payment assigned to each node by the adversarial nodes are combined by taking the average. The node with the highest average probability is assumed to be the destination of the payment.

We found that the first timing attack method, where the adversary assumes that the adversarial node closest to the destination found the correct destination of the payment always outperforms or equals the other strategies. This is expected, as the probability distribution for nodes being the payment destination has the lowest variance for this method. In scenarios where there are not a lot of nodes compromised by the adversary on the payment path, the strategies involving the timing attack perform equally. When there are three or more nodes compromised on the payment path, the methods start to differ. The method where the payment destination is assumed to be the next hop after the adversarial node performs the worst when there are fewer than 10 nodes compromised. When there are 10 or more nodes compromised, it performs better than the method where the farthest node from the destination is assumed to find the correct payment destination. However, it still performs worse than the other methods. Shadow routing seems to have little impact on the effectiveness of the timing attack. However, in situations where it has an impact, it will have a negative impact.

II. Background

In this section, the workings of the Lightning Network are described. First, payment channels are described. Next, this idea is extended to payment channel networks. Last, we describe how routing is done in the Lightning Network.

A. Payment channels

Poon *et al.* [6] introduced the Lightning Network. As an off-chain scaling solution, payment channels address the inherent limitations of conventional blockchain-based payment systems, including high transaction fees and long transaction confirmation times. These channels enable participants to engage in multiple transactions, without necessarily recording each transaction on the blockchain. Only a funding transaction is recorded on the blockchain and when a dispute occurs or

[2]https://www.torproject.org/

[3]https://github.com/lightning/bolts/blob/master/07-routing-gossip.md

upon closing the channel, a closing transaction is recorded on the blockchain.

We now go over the transactions involved when creating a payment channel as described by Lightning Networks BOLT 3[4]. In the Lightning Network, a payment channel is established by creating a funding transaction and recording it on the blockchain. Initially, an unsigned funding transaction is generated, with as output a 2-of-2 multi-signature script, requiring both parties to sign a spending transaction. Subsequently, signed commitment transactions are created, which spend from the 2-of-2 multi-signature output of the funding transaction. At this stage, both parties can sign the funding transaction and have it recorded on the blockchain without the risk of losing their funds. The commitment transactions, however, are not recorded on the blockchain, with their outputs denoting the current channel balances.

Suppose Alice and Bob want to establish a payment channel and create an unsigned funding transaction. Alice proceeds to create a commitment transaction signed by her and shares the signature with Bob. The commitment transaction spends the 2-of-2 multi-signature output from the funding transaction. Since Alice lacks Bob's signature for the commitment transaction, she is unable to utilize it, whereas Bob can sign and utilize the transaction. One of the commitment transaction's outputs can be spent directly by Alice, representing her channel balance. The other output can be spent either by a signature from a new key held by Bob, known as a revocation key, along with a signature from Alice, or by Bob after the commitment transaction has been recorded on the blockchain for a specified number of blocks, referred to as the timelock period. This output represents Bob's channel balance. Bob creates a similar commitment transaction and shares its signature with Alice. The only difference is that Bob can spend his balance directly and Alice has to wait for the timelock period to spend her balance or have Bob's signature and the signature of the revocation key. In addition, only Alice can publish this transaction, as Bob is lacking Alice's signature.

We now go over the messages involved when changing the balance of a payment channel as described by Lightning Networks BOLT 2[5]. New commitment transactions are generated whenever adjustments to the channel balances are required. Suppose Alice wants to pay Bob. Alice starts by generating a new commitment transaction that reflects the new channel balances and shares her signature with Bob through a `commitment_signed` message. Subsequently, Bob shares his revocation key with Alice in a `revoke_and_ack` message, effectively cancelling the old commitment transaction. Additionally, Bob generates a new commitment transaction reflecting the same channel balances and shares the signature with Alice through a `commitment_signed` message. Last, Alice also shares her revocation key of the previous commitment transaction with Bob through a `revoke_and_ack` message. Consequently, when an old commitment transaction

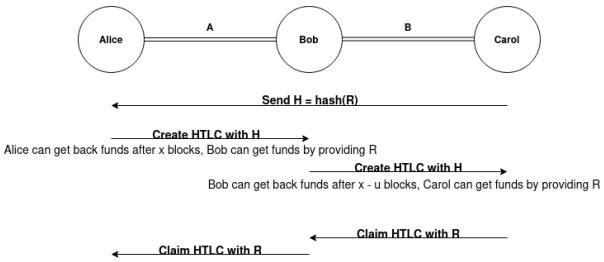


Fig. 1: A schematic overview of a payment from Alice to Carol through Bob over the Lightning Network. Alice and Bob share a payment channel A and Bob and Carol share a payment channel B. R is the preimage of hash H. x is the total timelock delta for Alice when paying Carol through Bob. u is the timelock delta of Bob for channel B.

is recorded on the blockchain, the counterparty is provided with a timelock period, during which they can spend the transaction using a signature from the revocation key and their key. This enables the counterparty to access all the funds in the channel when a party is not acting honestly, while the dishonest party's funds are locked.

To close the channel, either party can record the latest commitment transaction on the blockchain. However, the party initiating the broadcast must wait for the timelock period to elapse before being able to spend their output. Alternatively, both parties can cooperatively close the channel by generating a new transaction without a timelock and revoking the commitment transaction.

B. Payment channel networks

While payment channels offer immediate and efficient off-chain transactions, they are limited to direct channels between two parties, requiring a funding transaction for each recipient. Payment channel networks, on the other hand, enable the creation of a network of interconnected payment channels, allowing for multi-hop transactions across the network. This network-based approach introduces a new level of scalability, enabling participants to transact with any other participant in the network, regardless of direct channel connections, as long as the recipient is somehow connected to the sender.

We now go over the basics of payment channel networks as described by Lightning Networks BOLT 7[6]. To enable a network of payment channels, the Lightning Network employs Hashed Timelock Contracts (HTLCs). To spend an HTLC, the spender must provide a signature and a preimage of a hash (the hash is included in the contract). Alternatively, the spender can spend after a predetermined block height (also included in the contract) when providing the correct signature. The key of the signature for spending the output using the preimage of the hash does not have to be the same as for spending the output with the timelock.

Let's consider a scenario where Alice shares a channel with Bob (channel A), who, in turn, shares a channel with

[4]https://github.com/lightning/bolts/blob/master/03-transactions.md

[5]https://github.com/lightning/bolts/blob/master/02-peer-protocol.md

[6]https://github.com/lightning/bolts/blob/master/07-routing-gossip.md

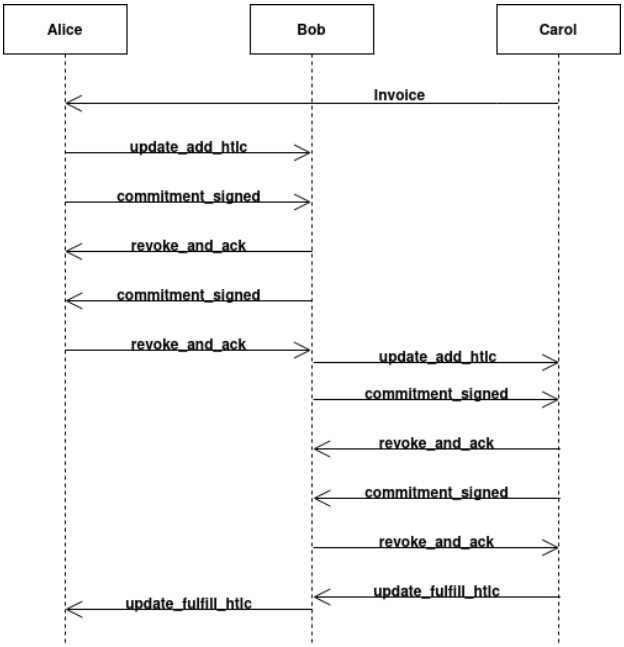


Fig. 2: A sequence diagram of the messages involved when Alice pays Carol through Bob over the Lightning Network.

Carol (channel B), and Alice intends to make a payment to Carol. See Figure 1 for a schematic overview of this situation and Figure 2 for a sequence diagram of the involved messages. The Lightning Network accomplishes this by having Carol generate a hash H of a randomly selected preimage R, which is then shared with Alice. Subsequently, Alice sends a `update_add_htlc` message containing H to Bob and negotiates a new commitment transaction for channel A, featuring an additional output as HTLC. The amount in the HTLC is the amount Alice wants to pay to Carol and the sum of the transaction fees charged by the intermediate nodes. The balance of Alice is lowered by this amount and the balance of Bob is kept the same as in the previous commitment transaction. The block height after which the HTLC can be spent by Alice must be sufficient so that Alice, Bob, and Carol have time to process the transaction. The HTLC can be spent by Bob once he possesses the preimage R. Alternatively, Alice can spend the HTLC after a specified block height.

Once Bob receives the `revoke_and_ack` message from Alice, he sends an `update_add_htlc` message to Carol, containing the same hash H. Bob then negotiates a new commitment transaction with Carol for channel B, incorporating an HTLC output. Carol can spend this output by using the preimage R, while Bob cannot until a certain block height. The amount in the HTLC is the same as the HTLC between Alice and Bob but without the transaction fee Bob charges. The balance of Bob is lowered by the amount of the HTLC and the balance of Carol is kept the same. The block height after which this HTLC (of channel B) can be spent must be lower than the block height of the HTLC on channel A. This way, Alice can not claim the funds of Bob back, before Bob can claim the funds back from Carol. Note that this pattern can continue until the payee receives the `update_add_htlc` message.

As Carol is the payee and therefore possesses the preimage R, she has the option to either close the channel and claim the HTLC funds from Bob or transmit the preimage R to Bob in a `update_fulfill_htlc` message. In the last situation, Bob and Carol can also negotiate a new commitment transaction for channel B where the funds of the HTLC are added to Carol's balance. In either case, Bob will possess the preimage R and can claim the HTLC funds from Alice in the same manner as Carol. In the event of unresponsiveness from one of the parties, channels can be reset or closed without incurring any fund loss. The closing party simply has to wait until a certain block height before they can spend the HTLC output and get their coins back.

C. Routing

We now go over the basics of routing in the Lightning Network as described by BOLT 7[7]. Nodes in the Lightning Network use source routing, meaning that the payer determines the route a payment takes to reach the payee. The payment messages are sent using Onion Routing[8], using the Sphinx format [10].

A payment channel can either be private or public. Private channels cannot be used for others' payments, as the only parties that know about the channel are the endpoints of the channel. In contrast, public channels accept payments for other destinations than the endpoints of the channel.

Once a public channel is fully created, which is after the funding transaction is recorded on the blockchain and the block has propagated through the Bitcoin network, the channel is announced to the network through a `channel_announcement` message. The channel becomes usable once at least one endpoint announces its policy in a `channel_update` message. Nodes can announce themselves using a `node_announcement` message. All three messages are broadcast to the network using gossip. If a node wants to change the policy of one of its channels, it can send a `channel_update` message again. A channel is identified by the short channel id. This id contains the block height of the funding transaction in the three most significant bytes, the index of the transaction on the block in the middle three bytes and the index of the output funding the channel in the two least significant bytes.

The policy of a channel is directional and each endpoint of a channel dictates its policy for forwarding a payment from the node to the other endpoint. As in the case of a non-responding node or a dispute, the funds of a node can be temporarily locked in the HTLC and as a node needs to pay transaction fees for the closing transaction to close a channel, nodes charge a fee for forwarding transactions over one of their channels. The parameters to calculate the fee are included in the policy. The

[7]https://github.com/lightning/bolts/blob/master/07-routing-gossip.md
[8]https://github.com/lightning/bolts/blob/master/04-onion-routing.md

fee consists of a fixed amount and a factor that is multiplied by the amount of the payment.

Additionally, the policy contains the `cltv_expiry_delta`. This is the amount the node subtracts from the block height (`cltv_expiry`) in the received HTLC when forwarding the payment. The higher this value, the longer the forwarding node has to respond in case of a dishonest counterparty that broadcasts a revoked commitment transaction. However, a higher value also means that funds will be locked for longer in case of a dispute.

When a node wants to send a payment, it tries to find a path from itself to the destination. It can model the public channels as a weighted directed multigraph, where the edges correspond to the payment channels and the vertices correspond to the nodes in the network. A cost function is used to determine the weights on the edges. The goal of this function is to make the final transaction fee and the total `cltv_expiry_delta` low. The cost functions differ between Lightning clients, although they all use the public information of the channels, such as capacity, fees, and `cltv_expiry_delta`. The three most prominent Lightning clients are LND[9], Core Lightning[10], and Eclair[11].

A path is only valid if the capacities of the payment channels are sufficient, meaning that of a certain payment channel, the capacity is at least the payment amount the payer wants to pay plus the fees charged by the nodes owning the payment channels after the current channel. Note that the channel balances are not public knowledge. Therefore, payments can fail if a channel along the path does not have a sufficient balance in the correct direction. Nodes use an adapted version of Dijkstra to construct the cheapest path according to the cost function of the used client.

To increase privacy, nodes can use shadow routing. With shadow routing, an additional value is added to the `cltv_expiry` calculated for the payment path. This value could be random, or it could be the sum of the `cltv_expiry_delta` values along an extension of the payment path. By making the `cltv_expiry` bigger, the set of paths a payment can take to complete as seen by an adversary observing the payment will be bigger. This will increase the number of receivers of the payment, increasing privacy.

III. Related work

In this section, an overview of the related work on attacks targeting the privacy of transactions within the Lightning Network is presented. Specifically, two prominent types of attacks are discussed: sender/receiver identification and channel balance probing attacks. Sender/receiver identification focuses on uncovering the identities of the endpoints involved in a transaction. Channel balance probing attacks aim to find the balance of a channel in one or both directions.

[9]https://github.com/lightningnetwork/lnd

[10]https://github.com/ElementsProject/lightning

[11]https://github.com/ACINQ/eclair

Rohrer *et al.* [8] have introduced a timing attack to find the sender and receiver of a transaction that an adversary takes part in. An adversary first builds a probabilistic model for the time it takes to send a message between two nodes from the Lightning Network. To find the receiver, the time difference between forwarding an `update_add_htlc` message and receiving an `update_fulfill_htlc` message from the same payment is measured. The adversary can know which messages are from the same payment, since those messages will contain the same hash in the HTLC. Next, the collection of paths from the adversarial node to all other nodes is filtered such that the remaining paths have sufficient channel capacities for the transaction amount and the sum of the timelock delta's does not exceed the timelock value of the transaction. Last, the remaining paths are ranked based on how likely it is for a path to take the measured time using the probabilistic model. From this ranked collection of paths, the most likely receiver can be found. Finding the sender is done by failing the payment the first time the adversary receives an `update_add_htlc` message. The time it takes until an `update_add_htlc` message with the same hash in the HTLC is received will be measured. This time difference is used with the timing model to rank paths from all nodes to the adversarial node on how likely it is for that path to take that amount of time. Again, from this ranked collection of paths, the most likely sender can be found. Nodes should handle messages as soon as they receive them, causing a possible correlation between payment path length and payment completion time. A proof of concept of the attack is evaluated on a separate partition on the testnet of the Lightning Network, by creating Lightning nodes that are connected to each other, but not to the rest of the network. Furthermore, the attack is evaluated using a simulator. This paper is an extension of the work done by Rohrer *et al.* [8]. We extend the work by evaluating how effective shadow routing is against the attack as a mitigation. Furthermore, we extend the attack by evaluating how effective it is for an adversary to compromise multiple nodes and collude in finding the receiver of a transaction. We use different strategies to combine the ranking of paths obtained by the different nodes.

In [11], [12], the time difference between forwarding an `update_add_htlc` message and receiving an `update_fulfill_htlc` message is measured for different payment path lengths to assess if it would be possible to do a timing attack which finds the receiver of an observed transaction. The measurements were done on both a local network and on the testnet of the Lightning Network. Nisslmueller *et al.* [12] found that in the testnet of the Lightning Network, the time difference between the two messages was long enough for the volatility in the measurement to be negligible, making it a possible attack vector. In contrast to this paper, they were unable to do an actual timing attack.

Kumble *et al.* [7] have introduced an attack to find the senders and receivers of a transaction that is routed through an adversarial node. The adversary first finds all possible paths an observed transaction could have taken based on the routing algorithms of three popular Lightning clients. The paths are

found by using the cost functions and path-finding algorithms from the three Lightning clients and reconstructing which paths could have been found by each node such that the adversarial node was part of that path. Additionally, multiple adversarial nodes can collude by conducting the attack individually and if the transaction has the same hash in the HTLC combining the found paths Similarly to this paper, they try to find the receiver of an observed transaction. However, this paper uses a timing attack to achieve that.

Sharma *et al.* [13] have introduced an attack to find only the receivers and not the senders of a transaction. Similar to [7], they aim to find the receiver of a transaction that is routed through an adversarial node. They evaluate the effectiveness of the attack using a simulator of the Lightning Network for different numbers of adversarial nodes. They also evaluate the effectiveness of the attack when the Lightning Network grows, and conclude that the anonymity when using the Lightning Network suffers when the network grows.

Malavolta *et al.* [9] have introduced anonymous multihop locks (AMHLs). These locks could replace the HTLCs currently in use in the Lightning Network. An AMHL does not use the same hash for the transactions between different intermediate nodes. Therefore, the timing attack from [8] will not be possible. In addition, in the attack from [7], multiple adversaries colluding will no longer be straightforward.

Kappos *et al.* [14] also introduced an adversary that tries to identify the sender and the receiver of an observed transaction. Their attack works by assuming the sender is always the node that forwards the transaction to the adversarial node and the receiver is always the node where the adversarial node forwards the transaction to. This strategy is also used in this paper to compare the performance of more advanced strategies. In addition, they introduce an off-path adversary who learns about transactions that it did not participate in. The adversary uses a channel balance probing attack to find the balances of all channels and creates a snapshot of the network at regular time intervals. The difference in balances between snapshots indicates that at least one transaction happened between those nodes. This paper only focuses on an on-path adversary.

Tikhomirov *et al.* [15] have quantified how likely it is for an adversary to be able to identify the sender and receiver of a transaction. Assumed is that a sender selects a path with at most three intermediary nodes and the adversary controls the first and last intermediary node. The node that sends the transaction to the first adversarial node is assumed to be the sender and the node that receives the transaction from the last adversarial node is assumed to be the receiver of the transaction. In this paper, no such assumptions are made. Instead, a timing attack is used to estimate how far a receiver is from the adversarial node.

In [16], [17], [14], channel balance probing attacks are introduced. They work by first opening a channel with a node A. Then, the adversary sends transactions to one of node A's neighbours B through node A. A binary search strategy is used to find the minimum payment amount where the transaction fails due to insufficient balance on the channel between the other node A and its neighbour B. As the transactions use a random hash in the HTLC, all transactions will fail, only locking funds temporarily. The adversary uses the error code to determine if the payment failed due to the incorrect hash that is not known by the receiver, or due to insufficient balance on the probed channel. Biryukov *et al.* [18] have extended the probing attack to account for multiple channels between the same nodes. In [11], [12], the authors introduce a probing attack where the adversary does not need to have a direct channel with one of the endpoints of a channel. Although probing attacks are also attacks on a Lightning Network user's privacy, they are out of scope for this paper. We focus on identifying the receiver of an observed payment only.

IV. Adversarial model

It is essential to define and understand the adversarial model to evaluate the privacy delivered by the Lightning Network effectively. The adversary considered in this attack has the following properties:

1) **Honest-but-Curious:** The adversary follows the protocol as described in the BOLT documents[12]. It does not send malicious messages to other nodes. It can, however, see all the messages sent and received by the compromised node.
2) **Local internal:** The adversary only corrupts some nodes and not all of them. The adversary does not corrupt communication links. It knows all the information a Lightning node would normally know. However, it does know the private information of payment channels not belonging to one of the compromised nodes. Therefore, the adversary will not know the balances of the channels not belonging to one of the compromised nodes.
3) **Static:** The adversary can only corrupt fixed nodes. It cannot adapt by corrupting other nodes after learning some information.
4) **Probabilistic polynomial time:** The adversary can only finish algorithms with a polynomial runtime. The algorithms can be probabilistic. Therefore, the adversary can solve all problems in the PP complexity class. The adversary cannot break encryption and thus cannot read information hidden in the onion messages.

It is assumed that the nodes corrupted by the adversary have been online long enough to gather all available public information about the Lightning Network. The nodes also collect this information during normal operation. For each node which has at least one open public channel, the network addresses are known together with its public key. These addresses could be anonymous onion addresses through Tor or could be IP addresses. For each public channel, the nodes that form the endpoints of the channel and the capacity of the channel are known. In addition, the timelock delta and fee parameters for both directions are known.

[12]https://github.com/lightning/bolts/blob/master/00-introduction.md

With this information, the adversary can form a graph $G = (V, E)$, with V the set of nodes and E the set of channels. Each node has associated with it a public key and a set of addresses. Each channel has associated with it the capacity and for each endpoint the `cltv_expiry_delta`, the `fee_base_msat`, and the `fee_proportional_millionths`.

Once a transaction arrives at an adversarial node, the adversary also learns some things about the transaction. In addition to the node it receives the transaction from and the node it has to forward the transaction to, it learns the payment amount it will receive and needs to forward. It also learns the timelock value for the incoming and outgoing transactions. All this information is exploited to do the attack.

V. Method

In this section, the attack and the experiments to evaluate the effectiveness of the attack will be explained. First, we explain how to find the receiver of a payment using the adversary described in the previous section. Next, we explain the different variants of the attack. A simulator of the Lightning Network is used to simulate the attack, which is explained next. After describing the simulator, the way the data for the simulator is collected is explained. Then, the metrics used to evaluate simulations with and which simulations are run are given. Last, the ethical considerations for the evaluation of the attack are described.

A. Finding payment receiver

Finding the receiver of a payment is done using a timing attack from [8]. The first step in finding the receiver involves finding the latency distribution of the network connections between all nodes. This can be done by sending payments that will fail at a specific node. Since the payment messages are in the Sphinx format, and the route for the payment is decided by the sender, an adversary can craft payments such that it is only discovered at the target that the payment is invalid. The target will then send a message back to the adversary that the payment has failed. The time between the first `update_add_htlc` message sent by the adversary and the last `update_fail_htlc` received by the adversary is the latency of sending a payment to the target. Since nodes are required to respond to messages immediately, there will be a correlation between this latency and the path length of the transaction.

The adversary will do this probing for nodes with an increasing path length from the compromised node, meaning the number of payment channels to travel until the payment reaches the target is increased. By sending multiple probes to the same node, the adversary can estimate a mean and variance for a certain path length. The mean of the latency for the next channel can then be estimated by estimating the latency for the full path and subtracting the mean of the latency of the previous path length. The variance can be estimated by adding the variance of the previous path length. The distribution of the latency is assumed to be a normal distribution, which makes it possible to simply add the variances and means to get the aggregated distribution.

Once the adversary has probed all the channels, it can start de-anonymizing payments. Since each payment is uniquely identified by a hash in the HTLC, the adversary knows which `update_add_htlc` message corresponds to which `update_fulfill_htlc` message. The adversary has to record the time difference between forwarding the `update_add_htlc` message and receiving the `update_fulfill_htlc` message. This time difference is used to estimate the receiver of the payment.

Before estimating the receiver of the payment, the possible paths the payment can take are filtered. The adversary knows to which node it sends the `update_add_htlc`. Therefore, the rest of the payment path starts with that node. Furthermore, it knows the value of the payment. Thus it can remove any path that does not have enough capacity to forward the payment over. Last, the adversary knows the remaining timelock value. It can therefore filter the paths that require a higher timelock value. Note that when shadow routing is enabled, the timelock value of the payment will be higher than when it is not enabled, and therefore there will be fewer paths excluded.

Once the adversary has found the time difference between the `update_add_htlc` message and `update_fulfill_htlc` message and has a set of possible remaining payment paths, it can estimate the receiver of the payment. For each possible payment path, an aggregate distribution for the latency is created. Using this aggregate distribution, the probability that the observed time difference was measured is determined. Since it is possible for multiple paths to end at the same receiver, the probability that a node is the receiver has to be determined next. This can be done by summing the probabilities that a path was taken when the paths end at the same destination. Now the nodes can be ranked by this probability and the node with the highest probability is most likely to be the receiver of the payment.

B. Type of attacks

As is done in [8], the simplest attack to determine the receiver of a payment is to assume the receiver of the payment is the node where the adversary sends the `update_add_htlc` to. This method is called the **First spy** method in the rest of this paper. The timing attack described before as done in [8] is a more complicated attack. As an extension of the attack, three different methods are assessed in case there is more than one adversarial node on the payment path. The adversary can know the nodes are part of the same payment by comparing the hash in the HTLC.

In all three methods, the adversarial nodes first execute the timing attack individually. In the first method, the adversary assumes that the adversarial node closest to the destination found the correct destination of the payment. The adversary can know which node is the closest by comparing the timelock values or the amounts. The node observing the lowest timelock value is the closest to the payment destination. This method is called **Timing closest** in the rest of this paper. Since the

adversarial node is closest to the destination, the probability distribution of how likely a node is the destination has the smallest variance. Therefore, it will be interesting to see if this method indeed performs best. Alternatively, the adversary can assume that the adversarial node farthest from the destination found the correct destination of the payment. The node observing the highest amount or timelock value is the farthest away. This method is called **Timing farthest** in the rest of this paper. It will be interesting to see if this method performs worse because it is farther from the payment destination.

In the last method, called **Timing collusion** in the rest of this paper, the adversary collects the probabilities the adversarial nodes assign to all the nodes for them being the destination of the payment. Then, for each node in the network, the adversary takes the average of the probabilities found by the different adversarial nodes. The node with the highest average probability will be the estimated destination. This method makes use of all the available data the adversary has. Therefore, it will be interesting to see how it performs in comparison with the **Timing closest** method.

C. Simulator

All experiments are run using a simulator. The simulator is originally created by Rohrer *et al.* [8]. It is written in the Rust programming language.

The simulator is run as a command line application. It simulates a certain number of transactions which is specified when starting the application using a command line argument. All transactions will have the same amount, which is also specified as a command line argument. The number of compromised nodes by the adversary can also be specified as a command line argument. The simulator implements different strategies for adversaries. However, only the most central adversary strategy is used. This strategy picks the nodes to compromise from a list ordered by the betweenness centrality of the node.

A graph of the Lightning Network is used such that the simulator knows which nodes and channels are present in the network. This is supplied as a JSON file, which contains a list of nodes and a list of channels. The node information includes a timestamp of when the last update was received from a `node_announcement` message, the public key of the node, an alias, and a set of addresses the node can be reached on. These addresses can be IPv4 or IPv6 addresses, or a tor .onion address. The channel information includes the channel ID, the short channel ID, a timestamp of when the last update was received from a `channel_announcement` message, and the capacity. The channel information also contains the policies for both endpoints of the channel. A policy includes the `time_lock_delta`, `min_htlc`, `fee_base_msat`, `fee_rate_milli_msat`, `disabled`, `max_htlc_msat`, and a timestamp of when the last update was received from a `channel_update` message. The `min_htlc` and `max_htlc_msat` specify the minimum and maximum amount that can be forwarded over the channel. `fee_base_msat` is the base fee and to get to the total fee this value is added to the product of the transaction amount and `fee_rate_milli_msat`. A channel can be disabled, which is indicated by `disabled`.

The simulator will decide which nodes are compromised using a list of node IDs ordered by descending betweenness centrality. The IDs should be the indexes of nodes in the list of nodes from the Lightning Network graph JSON file.

Last, a CSV file with measurements of latencies is used to sample the latencies of sending a message. A measurement should include from which continent to which continent a message was sent, the IP address of the destination, number of packets that were sent and received, the number of lost packets, the minimum, maximum and average round trip time of the batch of packets, and the standard deviation of the round trip time. In [8], this information was obtained by sending ping messages in batches. Each payment channel is assigned one random measurement with the same regions as the endpoints of the channel. This measurement is used to sample the message latency when a message is sent between these nodes. The distribution is assumed to be a normal distribution with as mean the average round trip time and as standard deviation the standard deviation of the round trip time. When a sample from this distribution is higher or lower than the maximum or minimum round trip time from the measurement, the sample is capped.

The simulator is a discrete event simulator with a logical timer that increments in steps of nanoseconds. It keeps track of the current time and saves events to be executed in an event queue which is a B-tree map with the time of the event as the key and the event type as the value. There are two event types: `ScheduledPayment` and `MessageReceived`. A simulation starts by adding `ScheduledPayment` events to the event queue spaced 12000 seconds apart. Then it loops over the event queue, picking the nearest event. If it is a `ScheduledPayment` event, it will queue the messages needed to start a transaction using a `MessageReceived` event. If it is a `MessageReceived` event, it will check what type of message is received, change channel states, and queue new messages if needed. The simulator simulates `UpdateAddHtlc`, `UpdateFailHtlc`, `UpdateFulfillHtlc`, `CommitmentSigned`, and `RevokeAndAck` messages. The channel states include the balance of the nodes of the channel and which nodes have acknowledged and committed.

For this paper, a few additions are made to the simulator. First, shadow routing is implemented, according to BOLT 7[13]. This is implemented by taking a random number between 1 and 8. Then a random walk is done with a length of that random number from the destination of the payment. The timelock values of the channels encountered are summed up. This final value is added to the timelock values seen by the nodes on the real path. The longer the random walk, the more the original timelock value gets obfuscated. The maximum length of the random walk is chosen to be 8, which is twice the average payment path length (4) from the experiments. In

[13]https://github.com/lightning/bolts/blob/master/07-routing-gossip.md

addition, collusion between adversarial nodes using different strategies is implemented as described in the previous section.

Some issues have been found in the simulator, which were only encountered when it was extended. One shortcoming of the simulator is that the channel state does not allow commitments and acknowledgements per payment. A real Lightning client will keep track of all the not-committed payments separately. This is not a problem in a situation where different payments are spaced in time. However, to estimate the source of a payment, an incoming payment is failed by the adversary and the sender retries the payment one more time. Once this is extended to multiple adversarial nodes that fail payments and combined with the fact that messages can happen out of order due to concurrency, a `CommitmentSigned` or `RevokeAndAck` message can commit or acknowledge the channel, while it is actually waiting for two commits or acknowledgements. Now one of the payments is ignored and will never be received by the receiver. This can be solved by extending the channel state to commit and acknowledge per payment. However, the channel state is one of the lowest-level concepts in the simulator and is connected to everything. When changing the channel state representation, the simulation code of the Lightning Network also needs to change. When the simulation code changes, the attack code also has to change. Therefore, it takes a lot of time to implement this. Furthermore, debugging issues in this code takes a lot of time, since most of the issues are concurrency issues which do not always happen and are hard to reproduce consistently.

D. Data collection

To start a simulation, the simulator has to know the layout of the network. In addition, it has to know how long it takes to send a message. The snapshot used is taken from a GitHub repository that ran a Lightning node and recorded all the messages needed for creating a graph of the Lightning Network [19]. The messages are parsed by the code provided by the repository and converted to a human-readable JSON format. The `channel_announcement` messages are used to link the `short_channel_id` to the public keys of the endpoints. The `channel_update` messages are used to obtain the policies of each endpoint of each channel. This includes the `cltv_expiry_delta`, `fee_base_msat`, and `fee_proportional_millionths`. The `node_announcement` messages are used to obtain the network addresses of each node.

There are 370552 `channel_announcement` messages, 29498954 `channel_update` messages, and 3377090 `node_announcement` messages in the snapshot. Not all channel's have both a `channel_announcement` and `channel_update`. Similarly, not all nodes that appear in a channel have a `node_announcement` and vice versa. This results in 36798 nodes and 370553 (directional) channels.

The capacity is missing in the messages available in the snapshot. This is, however, included in the opening transaction of the channel on the Bitcoin blockchain. Therefore, to obtain the capacities of the channels, the Bitcoin blockchain is scanned. The `short_channel_id` property of a channel contains in the three most significant bytes the block height, in the next three bytes the transaction index and in the two least significant bytes the output index of the spending transaction funding the channel[14]. The value of this output is the capacity of the channel. To obtain the capacities, a Bitcoin node is created and queried for these transactions. The balances of the nodes are not known to the public. Therefore, the simulator assumes that at the beginning of a simulation, the capacity is divided equally over both nodes.

The latency information is obtained by pinging nodes in the snapshot and measuring the round-trip time. Only the nodes with an IPv4 address and at least one active payment channel were pinged. The nodes are divided into regions by their IP address using the GeoIP2 Country[15] database. The continents where the IP addresses are located are mapped directly to a region with the same name. One exception to this is the region China, when an IP address is located in China it is not mapped to the Asia region, but to the China region. This results in the following seven regions: North America, South America, Europe, Afrika, Asia, China, Oceania.

Each node is pinged from seven different locations. This is achieved by deploying a T2 (Or T3 if T2 is not available in that region) micro AWS node in each region. In Table I, the deployed AWS nodes are summarized. The nodes are pinged in batches of ten pings (one ping command with the number of pings set to ten) in random order. This is repeated ten times resulting in each AWS node pinging all Lightning nodes a hundred times. For each batch of ten pings, the minimum, average, and maximum round-trip times are measured together with the number of lost packets and the standard deviation.

TABLE I: Deployed AWS instances for pinging LN nodes

Region	City	AWS region	Instance type
NA	North California	us-west-1	t2.micro
SA	Sao Paulo	sa-east-1	t2.micro
EU	Stockholm	eu-north-1	t3.micro
AF	Bahrain	me-south-1	t3.micro
AS	Mumbai	ap-south-1	t2.micro
CN	Hong Kong	ap-east-1	t3.micro
OC	Sydney	ap-southeast-2	t2.micro

E. Metrics

To evaluate the effectiveness of the attacks, three metrics will be measured using the simulator: precision, recall and F_1-score [8]. The precision is defined as the share of correctly classified payments from the successful payments that are observed by an adversary. The recall is defined as the share of correctly classified payments from all successful payments. The F_1-score is the harmonic mean of the precision and recall and is defined by the following formula:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

[14]https://github.com/lightning/bolts/blob/master/07-routing-gossip.md#definition-of-short_channel_id

[15]https://dev.maxmind.com/geoip/docs/databases/city-and-country

These metrics are chosen as they are easy to interpret. They indicate how successful the adversary is in determining the destination of a transaction in a value between 0 and 1. Here, 1 means that the adversary is always successful and 0 means that it is never successful.

F. Simulations setup

All simulations are executed five times using different seeds (1, 2, 3, 4, 5). Furthermore, the simulations are doing 200 payments each with a fixed value. This value is set to 1, 10, 1000, 10000, 100000 and, 1000000 satoshis. The simulations are also run with and without shadow routing enabled. Last, the simulations are done for different numbers of adversaries. The nodes that are compromised by the adversaries are chosen from a list ordered by betweenness centrality, picking the first n nodes. The simulations are done for $n \in \{1, 2, 3, 4, 5, 10, 15, 20\}$. The adversarial nodes are chosen to have high centrality to make sure a significant number of transactions pass through at least one adversary.

G. Ethical consideration

It is not ethical to deanonymize real transactions. Therefore, all attacks are done using a simulator which simulates fake transactions. The amounts of the transactions are chosen from a fixed set of values that do not correspond to any real transactions on the Lightning Network. Furthermore, senders and receivers of transactions are chosen at random. All information in the Lightning Network graph used by the simulator is obtained from information that is public knowledge. The balances of the channels are assumed to be split equally from the total capacity of the channel at the start of a simulation. Therefore, the balances also do not correspond to real balances in the Lightning Network. The latencies used by the simulator are obtained by sending ping messages to the nodes. Sending ping messages does not produce enough load to accidentally DoS a server or degrade its performance. The network addresses from the nodes are public knowledge. No messages were ever sent in the real Lightning Network.

VI. Results

In this section, the results of the experiments will be presented. First, the effectiveness of different methods of colluding will be compared. Next, the different methods of colluding will be compared for different payment amounts. Last, the effectiveness of shadow routing as a mitigation against the attack will be evaluated.

A. Effectiveness of collusion

In Figure 3a, the average number of compromised nodes on a payment path is plotted for different numbers of total adversaries in the network. In Figure 3b, the share of payment paths that are compromised and the share of payment paths where collusion is possible is plotted. In these plots, and all subsequent plots, the error bars represent the standard deviation. Collusion is possible when more than one node on the payment path is compromised. With only two adversaries in the network, it is almost impossible for adversaries to collude. With five adversaries, the probability of more than one compromised node on a payment path is already 15% and for twenty adversaries this probability goes to 37%.

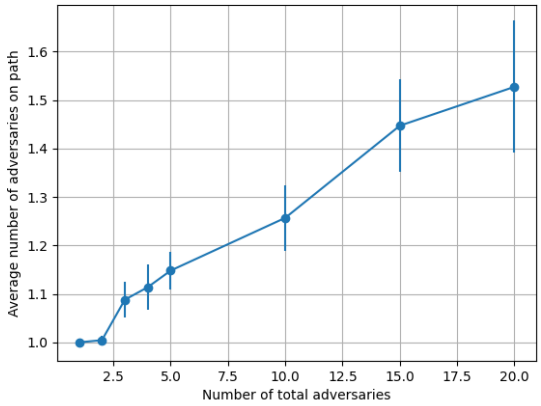
In Figure 4a, the precision for the different attack methods is plotted for different numbers of adversaries in the network when shadow routing is disabled. In Figure 4b this is done for the recall and in Figure 4c this is done for the F_1-score. In the plots, the average for the different transaction amounts is shown. For one or two adversaries in the network, the timing methods all perform equally for all three measures, while the **First spy** method performs a little bit worse. This can be explained by the fact that when there are one or two adversaries in the network, there is on average only one compromised node on the payment path. From three adversaries in the network onward, the performance of the different methods starts to divert. The **Timing closest** method has the highest precision, recall and F_1-score for all simulated numbers of adversaries. Second is the **Timing collusion** method. The **Timing farthest** method has the lowest precision, recall and F_1-score for ten adversaries in the network and higher. Even the simple **First spy** method performs better in this case. For lower numbers of adversaries, the **Timing farthest** method performs better. This could be explained by that for lower numbers of adversaries, there are not many compromised nodes on the payment path and since the compromised nodes are the most central nodes in the network, they will be close to the destination of the payment. When there are more compromised nodes, the farthest compromised node will be farther away from the destination, making the attack perform worse.

In Figure 5a, the precision for the different attack methods is plotted for different numbers of adversaries in the network with shadow routing enabled. In Figure 5b this is done for the recall and in Figure 5c this is done for the F_1-score. In the plots, the average for the different transaction amounts is shown. These figures show the same correlations as when shadow routing is disabled.

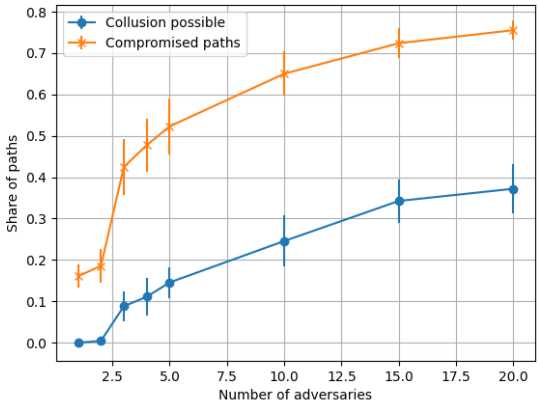
The **Timing closest** method achieves the highest precision, recall and F_1-score in all situations. The **Timing farthest** method achieves the lowest precision, recall and F_1-score of all the timing methods in all situations. Therefore, it seems that the closer a compromised node is to the destination, the higher the probability that it will de-anonymize the destination of a payment. The **Timing collusion** method, which takes the average over the probabilities an adversary assigns to a node being the destination, is not needed. It performs worse than the **Timing closest** method, as it will also include probabilities assigned to nodes estimated by adversaries far away from the destination. In addition, the **Timing closest** method is easier for adversaries to execute, as the adversaries just have to determine which is closer to the destination by looking for the lowest transaction value.

B. Comparison for different payment amounts

In Figure 6 the precision is plotted for the **Timing closest**, **Timing collusion**, and **Timing farthest** methods with respect

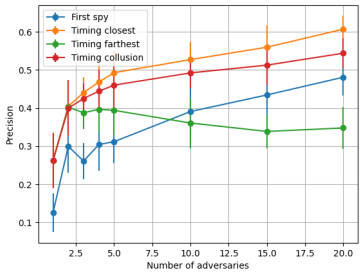


(a) Compromised nodes

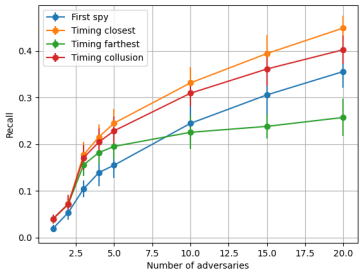


(b) Collusion possible

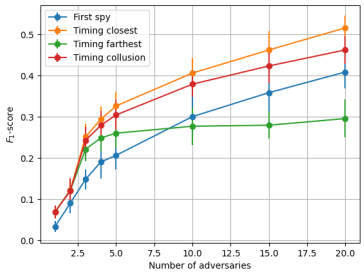
Fig. 3: Average number of compromised nodes on payment paths and share of paths where collusion is possible for different numbers of adversaries in the network



(a) Precision

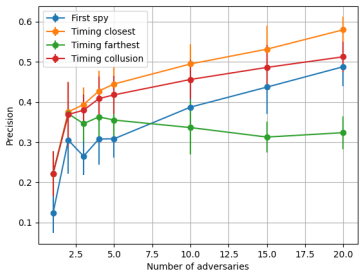


(b) Recall

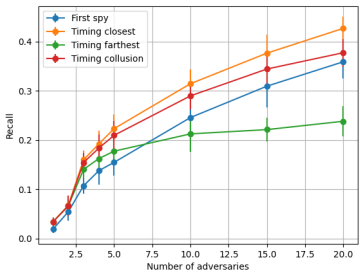


(c) F_1-score

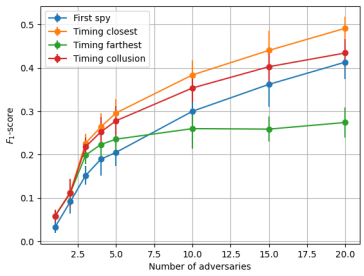
Fig. 4: Precision, recall and F_1-score for different methods of attack without shadow routing



(a) Precision



(b) Recall



(c) F_1-score

Fig. 5: Precision, recall and F_1-score for different methods of attack with shadow routing

to the precision of the **First spy** method when shadow routing is disabled. The precision is plotted separately for the payment amounts of 1, 10, 100, 1000, 10000, and 100000 satoshis.

The **Timing closest** method has a higher precision than the **First spy** method in all situations. Furthermore, the precision of the **Timing closest** method does not differ much for different payment amounts. For the **First spy** method, however, the payment amounts do have an impact on the precision. Lower payment amounts lead to higher precision. A possible explanation could be that when the payment amount is lower, more channels have a sufficient capacity to route the payment over. Therefore, the route from the transaction source to the destination could be shorter than when the payment amount is higher. A shorter route will make it more likely that the hop after the adversary is the payment's destination.

The **Timing farthest** method performs worse than the **First spy** method when there are ten adversaries or more in the network. Interestingly, the **Timing farthest** method shows the same behaviour as the **First spy** method in that it has a higher precision when the payment amount is low. This could also be explained by that the transaction route may be longer for higher payment amounts. On a longer route, the adversary farthest from the destination will be farther away from the destination than on a shorter path. This could result in a lower precision.

The **Timing collusion** method has a higher precision than the **First spy** method, but a slightly lower precision than the **Timing closest** method. The difference in precision for different payment amounts for the **Timing collusion** method is smaller than for the **Timing farthest** and **First spy** methods. However, there is a trend visible showing that higher payment amounts result in lower precision. This could be explained by the fact that the **Timing collusion** method averages over the results of all the adversaries on the path, and that adversaries farther away from the destination show such a trend.

In Figure 7 the precision is plotted for the **Timing closest**, **Timing collusion**, and **Timing farthest** methods with respect to the precision of the **First spy** method when shadow routing is enabled. These plots show the same results as when shadow routing is disabled. However, the precisions are consistently lower.

C. Effectiveness of shadow routing

To assess the effectiveness of shadow routing on the attack, the simulator is run with shadow routing enabled and disabled using the **Timing closest** method. The **Timing closest** method was the most effective method found in the previous section. In Figure 8a, the precision of the attack is plotted with and without shadow routing enabled. In Figure 8b this is done for the recall and in Figure 8c this is done for the F_1-score. For all three measures, the attack works best with shadow routing disabled for all simulated numbers of adversaries. However, for recall, the biggest difference between the two is for ten adversaries, where with shadow routing the recall is 0.22 and without shadow routing, the recall is 0.25. These values differ less than their standard deviations from each other. For the precision, the difference is bigger, but only for three, four, and five adversaries in the network. Therefore, it is not likely that shadow routing has a significant impact on the performance of the attack. However, it has an impact on the performance of other attacks, such as done by [7].

VII. Future work

A simple extension to this paper, is assessing the effectiveness of the attack when determining the source of a transaction when using the **Timing closest**, **Timing collusion**, and **Timing farthest** methods for colluding adversaries. Only the simulators' channel state needs to be change to keep track of the commit and acknowledge status per payment. It would be interesting to see if the same conclusions hold for determining the source of a transaction.

To find possible mitigations against the attack presented in this paper, it would be interesting to see how effective the attack is against other payment channel networks, such as Raiden[16]. Some payment channel networks might deliver more privacy against timing attacks, indicating that their technology is better suited for higher levels of privacy.

Furthermore, it would be interesting to see if this attack can be extended to work when AMHLs by Malavolta *et al.* [9] are used. When AMHLs are used, messages that are sent at different points in time for the same transaction cannot be linked by comparing a hash in the messages. To have a working attack, a statistical model that indicates the probability of two messages belonging to the same transaction based on the time between those messages needs to be constructed. It would be interesting to see if the attack can still be effective.

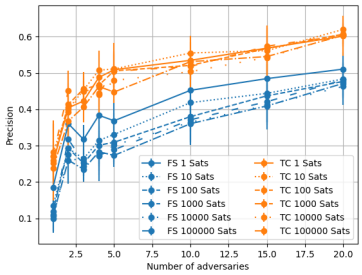
Last, a possible mitigation will be to delay all messages for a random amount of time. It would be interesting to see how effective the attack will be against such a network. Furthermore, it would be interesting to see what strategy of mixing messages will be the best with regard to privacy of the sender and receiver of a transaction.

VIII. Discussion

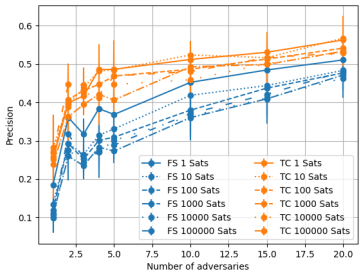
In this paper, a few assumptions are made that make the simulation simpler, but less like the real world. One of these assumptions is the processing time of messages. The processing time of a message is set to 10 ms per message. In the real world, this value might differ. In addition, the processing delay will not be a fixed value in the real world. Therefore, to make the simulator more accurate, the processing time should be a random variable of which the properties need to be determined through experimentation.

The latency model for sending a message over a channel is assumed to be a normal distribution. In the real world, this might not hold. For this paper, ping measurements were taken in batches of ten pings at once, meaning one ping command did send ten ping messages. The ping command provides statistics on the maximum, minimum and average round trip time as well as the standard deviation. From this, a normal

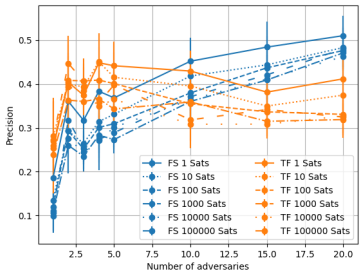
[16]https://raiden.network/



(a) **First spy** (FS) and **Timing closest** (TC) attack

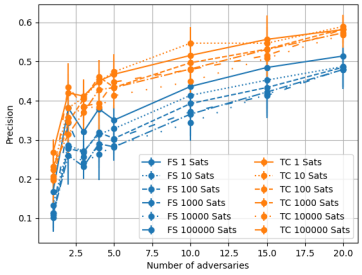


(b) **First spy** (FS) and **Timing collusion** (TC) attack

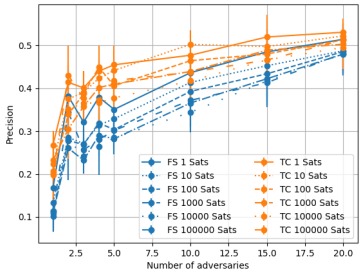


(c) **First spy** (FS) and **Timing farthest** (TF) attack

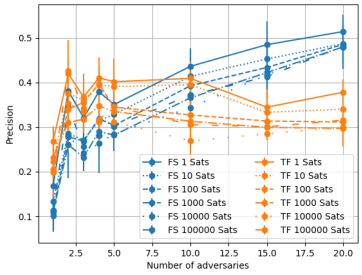
Fig. 6: Precision for attacks without shadow routing for different payment amounts



(a) **First spy** (FS) and **Timing closest** (TC) attack

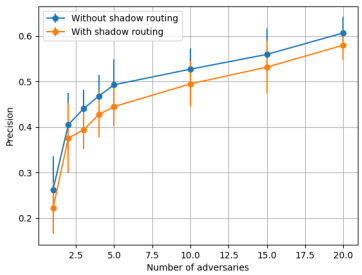


(b) **First spy** (FS) and **Timing collusion** (TC) attack

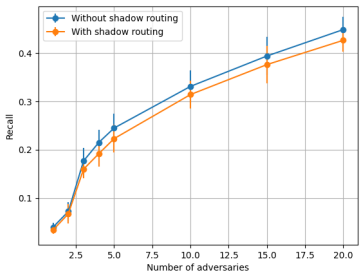


(c) **First spy** (FS) and **Timing farthest** (TF) attack

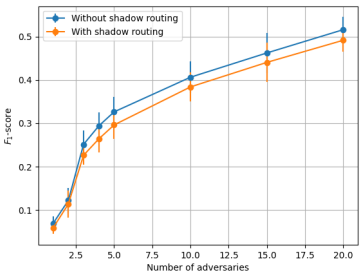
Fig. 7: Precision for attacks with shadow routing for different payment amounts



(a) Precision



(b) Recall



(c) F_1-score

Fig. 8: Precision, recall and F_1-score of timing closest method with and without shadow routing enabled

distribution can easily be modelled. However, it is not possible to see if a normal distribution fits the measurements. This can be solved by doing individual measurements instead of batches and checking what distribution the round trip time follows.

Furthermore, the round trip time of sending a ping message will be different from sending a message of the Lightning protocol. To be more accurate in the simulation, the round trip time of real Lightning protocol messages needs to be measured. Also, this round trip time may differ depending on the size of the message. The correlation between round trip time and message size needs to be investigated to have a more realistic simulation. This only needs to be done for the sizes of the messages used in the Lightning protocol.

The round trip time measurements were done only using IPv4. However, some transactions might be sent over Tor or over IPv6. To have more realistic simulations, a share of the messages also needs to be sent over Tor and IPv6. The latencies when using these protocols also need to be estimated. While it may be hard to determine which share of transactions is done over which protocol, some nodes will only have a Tor or IPv6 address. In this case, we can be sure that the transaction will be sent using that protocol.

Last, it is assumed that all nodes always respond. However, as was observed when obtaining the latency measurements,

some nodes do not respond at all. The simulation will be more realistic when some nodes do not respond. This will cause some payments to fail and the sender to try again. Currently, almost all payments succeed. The payments that do fail, fail because of an insufficient balance, and these payments are ignored in the evaluation.

IX. Conclusion

In this paper, we try to find the effect of colluding adversaries that try to de-anonymize the receiver of transactions in the Lightning Network. The adversary is classified as an honest-but-curious static local internal adversary. Only an on-path adversary is considered, meaning the adversary only de-anonymizes receivers of payments it takes part in.

Three methods for colluding are considered: **Timing closest**, **Timing farthest**, and **Timing collusion**. These three methods are compared to the baseline method **First spy**. When there are one or two adversaries in the network, the timing methods perform equally well in precision, recall, and F_1-score. The **First spy** method performs a little bit worse. When there are one or two adversaries in the network, there will on average only be one adversary on the payment path. This results in the **Timing closest**, **Timing farthest**, and **Timing collusion** methods doing the same thing, resulting in the same scores.

The **Timing closest** method has the highest precision, recall, and F_1-score in all situations. The **Timing collusion** method has the second-highest precision, recall, and F_1-score in all situations. The **Timing farthest** method performs better than the **First spy** method when there are less than ten adversaries in the network. When there are ten or more adversaries in the network, the **First spy** method performs better than the **Timing farthest** method. When there are more adversaries in the network, there are also on average more adversaries on the payment path. Since the adversaries are the most central nodes in the network, when there are more adversaries on the payment path, the adversary farthest away from the destination can be farther away than when there are fewer adversaries on the payment path. This indicates that the timing attack performs better when the adversary is closer to the target.

For the **First spy** and **Timing farthest** methods, the precision is higher when the payment amount is lower. When the payment amount is lower, more channels have sufficient capacity to route the payment through. Therefore, the payment routes can be shorter. A shorter payment route results in the adversary being closer to the target. This indicates again that the timing attack performs better when the adversary is closer to the target.

This paper also assesses the effectiveness of shadow routing against the timing attack. Enabling shadow routing does not influence previous conclusions. Furthermore, the difference in precision, recall and F_1-score is not statistically significant between when shadow routing is enabled or disabled, although the scores are consistently lower when shadow routing is enabled. Therefore, it seems that shadow routing is not a sufficient mitigation against the timing attack.

References

[1] S. V. Ward, "Why is bitcoin's price surging?" https://www.forbes.com/sites/digital-assets/2023/10/29/why-is-bitcoins-price-surging/, accessed: 2023-11-10.

[2] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 90–105.

[3] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "{Bitcoin-NG}: A scalable blockchain protocol," in *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, 2016, pp. 45–59.

[4] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 3–16.

[5] Visa, "Visa fact sheet a global payments technology company at a glance," https://usa.visa.com/dam/VCOM/global/about-visa/documents/aboutvisafactsheet.pdf, accessed: 2023-11-10.

[6] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[7] S. P. Kumble, D. Epema, and S. Roos, "How lightning's routing diminishes its anonymity," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ser. ARES 21. New York, NY, USA: Association for Computing Machinery, 2021.

[8] E. Rohrer and F. Tschorsch, "Counting down thunder: Timing attacks on privacy in payment channel networks," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, ser. AFT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 214–227.

[9] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," Cryptology ePrint Archive, Paper 2018/472, 2018.

[10] G. Danezis and I. Goldberg, "Sphinx: A compact and provably secure mix format," in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 269–282.

[11] U. Nisslmueller, K. Foerster, S. Schmid, and C. Decker, "Toward active and passive confidentiality attacks on cryptocurrency off-chain networks," *CoRR*, vol. abs/2003.00003, 2020.

[12] U. Nisslmueller, K.-T. Foerster, S. Schmid, and C. Decker, "Inferring sensitive information in cryptocurrency off-chain networks using probing and timing attacks," in *Information Systems Security and Privacy*, S. Furnell, P. Mori, E. Weippl, and O. Camp, Eds. Cham: Springer International Publishing, 2022, pp. 1–21.

[13] P. Kumar Sharma, D. Gosain, and C. Diaz, "On the anonymity of peer-to-peer network anonymity schemes used by cryptocurrencies," in *The Network and Distributed System Security Symposium*. Internet Society, 2023.

[14] G. Kappos, H. Yousaf, A. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn, "An empirical analysis of privacy in the lightning network," in *Financial Cryptography and Data Security*, N. Borisov and C. Diaz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 167–186.

[15] S. Tikhomirov, P. Moreno-Sanchez, and M. Maffei, "A quantitative analysis of security, anonymity and scalability for the lightning network," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 387–396.

[16] J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal-Pedrosa, C. Pérez-Solà, and J. Garcia-Alfaro, "On the difficulty of hiding the balance of lightning network channels," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 602–612.

[17] S. Tikhomirov, R. Pickhardt, A. Biryukov, and M. Nowostawski, "Probing channel balances in the lightning network," *CoRR*, vol. abs/2004.00333, 2020.

[18] A. Biryukov, G. Naumenko, and S. Tikhomirov, "Analysis and probing of parallel channels in the lightning network," in *Financial Cryptography and Data Security*, I. Eyal and J. Garay, Eds. Cham: Springer International Publishing, 2022, pp. 337–357.

[19] C. Decker, "Lightning network research; topology datasets," https://github.com/lnresearch/topology#dataset-2022-08-23, accessed: 2023-10-25.