Route Hijacking and DoS in Off-Chain Networks

Saar Tochner
School of Computer Science and Engineering, The Hebrew University
saart@cs.huji.ac.il

Aviv Zohar
School of Computer Science and Engineering, The Hebrew University
avivz@cs.huji.ac.il

Stefan Schmid
Faculty of Computer Science, University of Vienna
stefan_schmid@univie.ac.at

ABSTRACT

Off-chain transaction networks can mitigate the scalability issues of today's trustless blockchain systems such as Bitcoin. However, these peer-to-peer networks also introduce a new attack surface which is not yet fully understood. This paper identifies and analyzes a novel type of Denial-of-Service attack which is based on attracting routes, i.e., which exploits the way transactions are routed and executed along the channels of the network in order to attract nodes to route through the attacker. This attack is conceptually interesting as it highlights a fundamental design tradeoff for the defender (who determines its own routes): to become less susceptible to hijacking, a rational node has to pay higher fees to nodes that forward its payments.

We focus on the Lightning network, and we investigate both the structure of the network using real data collected over an extended period of time, and the routing algorithms of different implementations. We find that the three most common implementations (*lnd*, *C-lightning*, *Eclair*) approach routing differently. We then explore the routes chosen by these implementations in the current Lightning network. We find that very few nodes route most of the traffic: nearly 60% of all routes pass through only five nodes, while 80% go through only 15 nodes. Thus, a relatively small number of colluding nodes can deny service to a large fraction of the network.

We then turn to study an external attacker who creates links to the network and draws more routes through its nodes by asking for lower fees. While finding the optimal set of links to create is NP-complete, we show that using a greedy attack strategy, an attacker can obtain a $1 - 1/e$ approximation. Given this strategy, we find that just five new links are enough to draw the majority (65% - 75%) of the traffic regardless of the implementation being used. The cost of creating these links is very low. Drawing this traffic, the attacker is then able to deny service to all those who route through it. We further show that newer routing algorithms recently introduced into lnd, to penalize routes that failed in the past and avoid selecting them again, do not effectively prevent such attacks. Finally, we suggest modified routing policies, which may help alleviate the problem.

CCS CONCEPTS

• **Networks** → *Network experimentation*; • **Security and privacy** → **Network security**; **Denial-of-service attacks**; **Distributed systems security**.

KEYWORDS

Cryptocurrencies; Routing Attack; Lightning Network; Payment Channels Networks

1 INTRODUCTION

Emerging decentralized ledger and blockchain technologies bear the promise to streamline business, governance and non-profit activities, by eliminating intermediaries and authorities. A main hurdle toward such more decentralized applications however remains scalability [3, 8, 38]. The typical example is that Bitcoin can only support dozens of transactions per second, compared to several thousands in deployed payment services such as Visa [40].

Off-chain peer-to-peer networks (a.k.a. payment channel networks) [13] are a promising approach to mitigate this 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. This not only improves transaction throughput but also avoids the blockchain's transaction latency; ideally, in a payment channel network, transactions require communications only between a handful of nodes, while the blockchain is used only rarely, to establish or terminate channels. As an incentive to participate in others' transactions, intermediate nodes obtain a small fee from every transaction that was relayed through their channels. Over the last few years, payment channel networks such as Lightning [32], Ripple [2], and Raiden [30] have been implemented, deployed and have started growing.

This paper is concerned with the *routing* mechanisms which lie at the heart of payment channel networks. An important feature of payment channel networks is that they support transactions between participants without direct channels, using *multihop* routing [24, 32]. The route itself is selected by the source node, which differs greatly from conventional communication networks. The design tradeoffs and security implications of such routing for multi-hop payments are not well-understood today. In fact, routing in payment channel networks differs from routing in traditional communication networks in additional ways: in traditional communication networks, routing algorithms typically aim to find short and low-load paths in a network whose links are subject to fixed capacity constraints. In a payment channel network, link capacities represent channel balances, which can be highly dynamic: every transaction changes the balance initially set up for the channel. This in turn implies that transactions may fail due to temporary liquidity shifts within channels that are not known to all nodes routing through this channel. Moreover, both the establishment as well as the use of payment channels is an inherently strategic decision: it is subject to complex incentives and the extent to which a participant thinks she or he can benefit from different behaviors.

ACM Conference on Advances in Financial Technologies, 2020, New York, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In fact, a participant may not only try to strategically maximize her or his profit, but may also be *malicious*. Payment channel networks apply onion routing techniques to help hide the source and destination of payments (to preserve privacy); but these tools also aid attackers, as denial of service or other failures cannot be properly attributed to the responsible nodes.

1.1 Our Contributions

This paper is motivated by the question whether and how malicious players can strategically influence and *exploit* the way transactions are routed in off-chain networks. Our main contribution is the identification, analysis, and evaluation of a novel Denial-of-Service attack which is based on the hijacking of transaction routes. To this end, we examine different existing implementations (which turn out to differ significantly), and provide empirical insights into the structure and properties of payment channel networks.

Information on the structure of the Lightning network is gossiped to all its nodes. Examining this information and applying the different routing algorithms used, we find that there exists a group of 10 nodes that participates in 80% of the routes, and 30 nodes that participate in more than 95% of the routes.

We further consider an attack by an external participant that creates links to the network, hijacks traffic and performs a denial of service attack. We analyze the optimal attack and show that it is NP-complete to locate the best links to form. Still, we show that due to the submodularity of the problem, a greedy algorithm finds a 1-1/e approximation. Using this approach, we find that by creating only 5 *new* channels, with a total one-time cost of less than 16$ (evaluated in May 2020), an attacker can hijack about 65% of the routes, and with 30 channels (which cost less than a 100$ to establish), it can hijack 80% of the routes, regardless of the specific implementation clients use. The costs of channel creation are extremely low – around 3.1$ per channel (correct to May 31, 2020), which includes both the liquidity locked in channels for the attack, and the on-chain fees to create the channel.

Our findings are robust, and do not change significantly over network snapshots that were taken during the last 13 months.

In general, we observe only limited empirical evidence that users configure their nodes to extract high rewards. Nodes typically use default values or set minimal fees and contribute cheap routes to the network. Both aspects can be exploited by selfish and malicious players.

Reasoning about more secure solutions, we find that the underlying routing problem exhibits a fundamental design tradeoff, related to the desire of rational players to save on costs: by randomizing over more routes, even ones that cost more, a rational player can become more secure against hijacking attacks; however, such behaviors cost more, and may in fact allow intermediate nodes to *increase* their fees (knowing that routing algorithms will keep using them even if fees are not competitive) ruining the economic incentives that drive down relay fees in the network.

An additional tradeoff that we discuss regards the times nodes must be online (how often they need to check the network to make sure funds are not stolen from them) and the time it takes incomplete payment attempts to expire. For reasons of security, network defaults currently allow nodes relatively long periods of time for fund recovery, which implies that connection attempts that fail are not retried quickly.

In order to ensure reproducibility and in order to facilitate followup work, we share our code with the research community at https://github.cs.huji.ac.il/saart/saart-lightning.

Our initial findings and suggested mitigation techniques have been widely disseminated amongst Lightning developers and have been discussed at length.

1.2 Organization

The remainder of this paper is organized as follows. In §2, we will present our DoS attack and explore different hijack possibilities. §3 examines the state-of-the-art routing algorithms, and then provides an overview of the experiments we conducted. We then describe in §4 a model resulting from our attack as well as the algorithms that we used in our experiments. In §5 we explore methods to mitigate this vulnerability by using a modified routing algorithm and measure the impact of such a modification. Related work will be presented in §6. Finally, we conclude our contribution and discuss future work in §7.

2 DOS ATTACK VIA ROUTE HIJACKING

This section uncovers a potential vulnerability, based on route hijacking, which may be used for a Denial-of-Service (DoS) attack on off-chain networks. We will first provide an explanation of the key elements of the protocol, then present the basic attack, and finally, describe how it may be amplified. In the next sections, we explore the empirical feasibility of the attack using network data we collected and reason about optimization opportunities for both the attacker and the defender.

2.1 Context of the Attack

To be more concrete, we consider the Lightning network as a case study. However, the concepts are similar in other payment channel networks as well. In the Lightning off-chain network, the ***channels*** are established by the nodes for secure payments. Every two nodes that are willing to create such a channel, place a Bitcoin transaction (on chain) in order to lock money (i.e., liquidity) for this channel[1]. An off-chain transaction (within the channel) is then simply an agreement between the two end-points of the channel, which leads to a different split of the money between them. The intermediate states resulting from this transaction do not have to be committed to the blockchain: Once they will commit the state into the blockchain, the channel will be closed (because it *spends* the channel establishment transaction). Until this occurs, the channel can remain operational and the internal split of funds can be adjusted by the participants. As the intermediate states of channels are built, older states are *revoked*: if one tries to commit an old state, the other participant can claim funds back. This recovery of funds can only be done within a certain pre-set period of time. This setup thus requires each of the participants in a channel to occasionally check the blockchain and make sure that the other party did not close the channel using a revoked state. The timeout is set to allow for this,

[1]Note that this means that every channel is fully backed-up with a real bitcoins, therefore no one can spoof channels

and also to give time for revocation transactions to be accepted in the blockchain (in case of congestion on the blockchain).

Off-chain networks such as Lightning do not only support transactions between nodes that have a direct channel between them, but also allow chaining channels into longer paths in order to connect nodes *indirectly*. In order to transfer over such "multi-hop" paths, a transfer is executed on each channel along the composed path.

The technique used to chain channels together but still guarantee that funds are not stolen by intermediate nodes is based on "Hashed TimeLocked Contracts", or HTLCs, which are essentially contracts awarding nodes a slightly different split of the money in each channel if a secret is revealed. Paths are then created by establishing a chain of channels with HTLCs conditioned on the release of the same secret, and the transfer is finally executed as the recipient node releases the secret (additional details can be found in [32]). HTLCs must additionally posses an expiration time which specifies the timeout of each conditional payment. This timeout is the timeout of the next node in the path plus some small delay specified by the preceding node. These decreasing timeouts ensure that intermediate nodes never reach a situation where they might have an outgoing payment without being compensated by an incoming payment (due to an earlier timeout of the incoming channel). This difference between HTLC timeouts is expressed by a number of blocks in the blockchain (as timestamps on blocks are considered unreliable, and block height is the fundamental way to meassure the progress of time in blockchains). HTLC timeouts are set to allow intermediate nodes suficient time to claim funds if the node in the next hop in the path claimed funds from them. This timeout (that can not be set too low) in our context results in a ***delay*** for the transaction, and aids the attacker.

To motivate intermediate nodes to relay transactions, they are allowed to charge a ***fee*** for forwarding transactions. This fee consists of a *base fee* (a flat payment) and a *proportional fee* that is relative to the transaction size (number of coins processed in the transaction). For example, to use a channel that has a base fee of 100 millisatoshis (msat), and a proportional fee of 1 per million, a 1 million msat transaction will pay a fee of 101 msat, and a 2 million msat transaction will pay a fee of 102 msat.

Information about the structure of the channel graph and about the fees is continuously publicized by Lightning nodes through a gossip algorithm. Given this knowledge of the network graph's structure, nodes are able to utilize *source routing* in order to pick the path their own payments will follow. As we will see, different implementations use different routing algorithms for path selection, optimizing different measures (e.g. fee, timeout delay, security, etc.). We ourselves make use of the fact that the channel graph information is made public to evaluate our attack on the actual graph that exists in today's network.

2.2 Basic Attack: A Rerouting Vulnerability

The fact that nodes can strategically choose transaction paths introduces a potential vulnerability. In the following, we model the off-chain network as a graph, where vertices represent Lightning nodes and edges represent payment channels. In the basic attack, an adversary can aim to establish a set of edges in this graph which put it in a topologically important location, as well as to announce

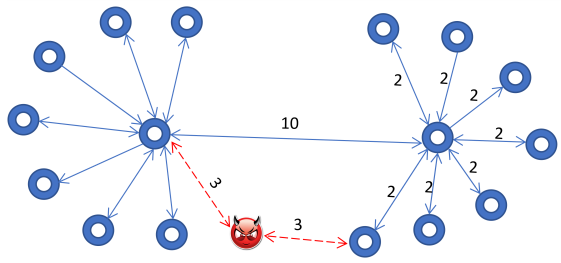

Figure 1: The routing vulnerability. An adversary creates edges that decrease the fees for many nodes.

a low fee. As a consequence, other nodes are likely to route transactions through the adversarial node. As route establishment is done via an onion-routing approach, an intermediary node may drop the payload and fail to follow through on the establishment of the rest of the path.[2] In this case, the payment does not take place, and the sender must wait for the original HTLC to expire before attempting to re-send the payment.

By maximizing its *centrality*[3], the adversary can hijack a large number of transaction paths, which in turn allows it to launch a Denial-of-Service attack. Even if payments are re-sent, randomized route selection may yet again cause the path to go through one of the attacker's nodes.

For an illustration, consider Figure 1. First, ignore the red node and its edges. In the blue network, there are then two groups of nodes that communicate through a single link with a high fee of 10. If an adversary (indicated as red node) now introduces the two red edges of low fee, it essentially creates a shortcut between some of the nodes, hijacking the transactions that aim to minimize the fee.

We note that counteracting this attack is non-trivial. Essentially, there are two options within the current framework:

(1) One may consider introducing mechanisms which quickly alert nodes about interrupted channels. However, this can also be problematic as it may violate payment privacy, and also be exploited by adversaries to make false reports of failed paths.
(2) If the source node does not know which specific channels were stopped, it may only heuristically avoid nodes or channels from the original path (which may disconnect the network or lead to higher fees), and/or hope that a new randomly chosen path may reestablish connectivity.

Note that every node can create channels to most of the nodes that it chooses: the default behavior in all the implementations is to accept every channel suggestion. This willingness to connect can be attributed to the perceived low risk in doing so: the construction guarantees that none of the funds in the channel is at risk of theft. Given the attack that we propose, it may be wise to accept channel connections only from known and trusted entities.

[2]We validated it as a POC in the lab, using several instances of Lightning nodes connected to a local Bitcoin network running in "regnet" mode.
[3]In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph [10].

2.3 Amplified Attack: Delay Vulnerability

The basic attack may be further amplified by exploiting the timeout mechanism to induce delays. If the attacker pretends to participate in path establishment, but does not really relay the path establishment request, other nodes before it on the path have their funds locked and will be able to free them only after a timeout. This means that a higher timeout value will lock the money of the nodes in the path for a longer time, but at the same time, some of the Lightning implementations attempt to avoid these high delays. We explore this aspect as well in our evaluation.

3 FEASIBILITY AND CASE STUDY

We now explore the feasibility of the attack identified above. To this end, we consider different Lightning network implementations and also conduct an empirical study on today's network topology, its fees and other parameters, which may be of independent interest. We also report on our experimental evaluation results.

3.1 Existing Implementation Details

In order to investigate the feasibility of our attack, as a case study, we consider the three main implementations of the Lightning network: *lnd* (implemented in Golang), *C-lightning* (implemented in C) and *Eclair* (implemented in Scala). The implementations differ in the way they operate relative to aspects not covered in the BOLTs [23] which make up the Lightning network's standard. Specifically, the standard does not dictate any routing behavior, leaving each implementation to set its own. In our experiments, we use the default parameters of every implementation.

3.1.1 lnd. *lnd* chooses the path of minimum weight, calculated using the following recursive formula, where p is the list of channels in the path, and *ams* is the list of amounts that go through each channel (which changes due to fees that are removed at each hop):[4]

$$fee = ams[i+1] \cdot p[i].propFee + p[i].baseFee$$

$$weight[i] = ams[i+1] \cdot p[i].delay \cdot riskFactor + fee$$

The default $riskFactor$ is set to 15/1,000,000,000.

Note that *lnd* changed this weight function in March 2019, in commit 6b70791, and added[5] a new summand to the channel's weight: $\frac{100}{edgeProbability}$. This parameter is an aggregated success score over the previous routing through this channel. If the node has no prior knowledge about the channel, then it uses the default value that is relative to the a-priori failure rate in the network. Otherwise, the penalty considers only the time of the last failure. In the first hour, the probability is 0 (dividing by 0 here will yield infinity), and then it increases exponentially with the formula: $0.6 - \frac{0.6}{2^h}$ (by default, h counts in hours). We note that *lnd* looks only at the last transaction failure when it discounts channels, so if the time between failed attempts is long enough, it will effectively choose between at most two channels in the network, both with low weight (quite similarly to *Eclair*'s "top 3" approach outlined below). This is not in itself sufficient to bypass the attacker, as it is easy for it to be on both routes (just like we show for *Eclair*). Additionally, the decay rate of the penalty on failures needs to be slow in order to remain relevant as the previous HTLC contract times out. At last, an attacker that completes the route as requested, but delays the HTLC secret release until the very last moment, will delay the transfer significantly and will not suffer the penalty at all.

Our empirical analysis examines *lnd* without this mechanism in order to examine "the first route". Later, Figure 17 shows the effectiveness of this added penalization method.

3.1.2 C-lightning. *C-lightning* multiplies the fee for each edge by a random fuzz factor, and gives a penalty for high timeouts. Denote by h the hash that was calculated using siphash24 on a random string that the user generated (before every path selection) and the short channel id. Let $fuzz$ denote the range of the random noise factor (plus or minus 0.05 by default).[6] The weight assigned to each channel is then:

$$scale = 1 + fuzz \cdot (2 \cdot \frac{h}{2^{64}-1} - 1)$$

$$fee = scale \cdot (ams[i+1] \cdot p[i].propFee + p[i].baseFee)$$

$$weight[i] = (ams[i+1] + fee) \cdot (p[i].delay \cdot riskFactor) + 1$$

for a configurable $riskFactor$, which is 10 by default.

3.1.3 Eclair. *Eclair* multiplies the fee by a proportional factor depending on the channel properties: delay, capacity, and height (while assuming upper and lower bounds for each of them). In addition to the above, *Eclair* also randomizes the selected paths uniformly, from the 3 best routes.[7]

$$fee = ams[i+1] \cdot p[i].propFee + p[i].baseFee$$

$$\begin{aligned} weight[i] = fee \cdot (&normalizedDelay \cdot delayRatio \\ &+ normalizedCapacity \cdot capacityRatio \\ &+ normalizedHeight \cdot ageRatio) \end{aligned}$$

For upper and lower bounds[8]:

$$9 < delay < 2016 \quad delayRatio = 0.15$$
$$1000 < capacity < 2^{24} \quad capacityRatio = 0.5$$
$$0 < height < 8640 \quad ageRatio = 0.35$$

3.2 First Empirical Insights

We first provide some general analysis of today's Lightning network. While some of the analysis is not directly related to routing, it provides insights into the behavior of users, and what this behavior implies for the vulnerability of the network. In particular, these insights show that users tend to use the default values, which we will later use in our routing analysis.

[4] References can be found in the methods *FindRoutes* and *findPath* in *lnd*.routing.router.go and pathfind.go

[5] References can be found in routing/missioncontrol.go:261, routing/pathfind.go:531 and probability_estimator.go:145

[6] References can be found in the methods bfg_one_edge and find_route in the file gossipd/routing.c

[7] This is a configurable parameter. References can be found in: eclair-core/src/main/resources/reference.conf, and the methods FindRoute, edgeWeight in the files eclair/router/Router.scala, Graph.scala

[8] 8640 is the number of blocks Bitcoin creates in expectaion over a two month period

3.2.1 Methodology. The following results are based on measurement data we collected using a live Lightning node (*lnd*) that is connected to the mainnet (the production network). We used the CLI command *lncli describegraph* in order to extract the network structure (currently, the whole topology is stored by all nodes in order to allow source-routing). We use the topology's snapshot that was retrieved from a live mainnet Lightning node that we maintained. We queried the node five times, ending on May 20th 2020. Most of our analysis will be focused on the latest snapshot, and in Figures 19,20 we show that results do not significantly change over time.

Note that using this method, we can examine only public channels; our analysis omits private channels. We argue that since these channels are private, nodes which are not directly part of the private channel are not aware of them and will not route through them. Private channels are therefore typically used only at the beginning of routes (only their creators are aware of them) and so any transfer other than a direct one between the two will still route using the rest of the network and will be affected by our attack. Thus most routing, by design, relies primarily on the public network.

3.2.2 Network Analysis. The network is composed of 4,300 nodes, 33,600 channels, with an average channel capacity of 0.028 BTC.

Figures 2, 3, 4, 5, and 6 show some basic properties of the Lightning network. In particular, Figure 2 reveals that the base fee across channels has two highly common values: most channels simply use the default value, which is 1000. Interestingly, however, the second most frequent value, and the most frequent non-default value, is the minimum possible fee. This provides two main insights: first, most users do not configure the software beyond the default values; and second, most of those who do, do it in a way which *supports* the network. Thus, we hardly find any evidence for selfish optimizations of fees in the current network. Both properties may influence a potential attacker.

Figure 3 shows the corresponding distribution for the proportional fee. Here, the default value is 1/1000, which is also by far the most frequent value. Interestingly, however, the value 1 is also frequent; we conjecture that this may be due to a confusion with the base fee, or with units (satoshis vs millisatoshis). Other high values also appear, which one may interpret as an attempt to profit from the network—but this is unlikely: the values are still very small. Given the network scale, nodes are unlikely to be able to benefit from such fees [17]. This figure also suggests that there are altruistic nodes in the network, which are willing to hold channels without taking fees. Indeed, out of the channels with base fee 0, we find that the percentage of channels with 0 proportional fee is about twice the percentage in the whole network (about 40% compared to 24%).

Similarly to the other figures, Figure 4 shows that most of the channels use default values, but there are other manual configured values. In this case, we see that 144 blocks (a full day) or 40 blocks, are used as timeout values for most of the channels. Note that the cumulative percentage is similar to the percentage of the default configuration in Figures 2 and 3.

Figure 5 provides another interesting insight: the capacities of the channels are surprisingly large: around 5% of the nodes invest $\frac{1}{10}$ of a full Bitcoin ($10M$ satoshis) into a channel. This demonstrates a high level of commitment to the network, which fits in well with the surprisingly low fees mentioned above.

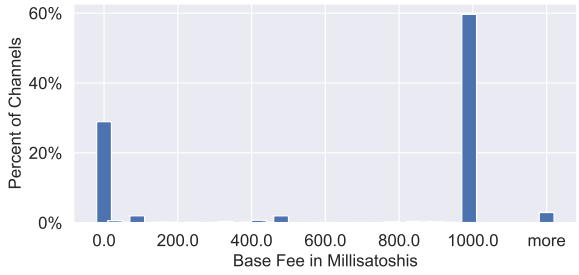

Figure 2: Channels by base fees

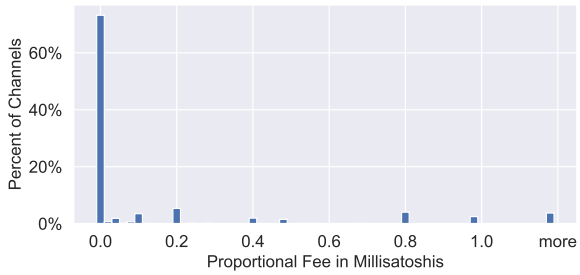

Figure 3: Channels by proportional fees

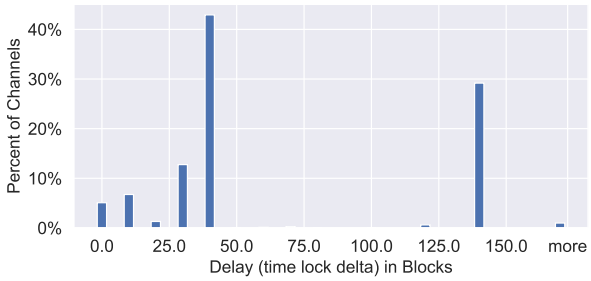

Figure 4: The delay of the channels (144 blocks is $\sim$ 24 hours)

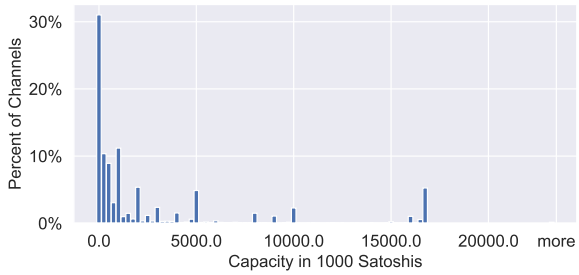

Figure 5: Channels by capacities

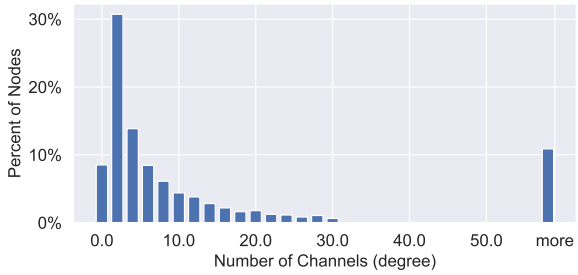

Figure 6: Number of channels per node

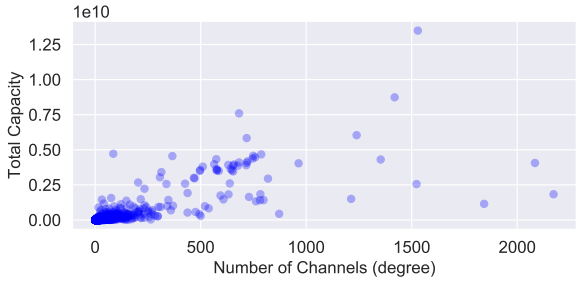

Figure 7: Liquidity of nodes vs degree. Every point represents a single node (5% of the nodes were trimmed).

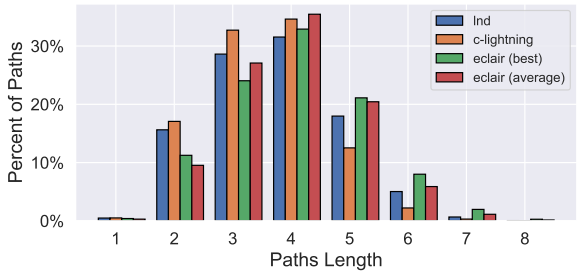

Figure 8: Number of edges in each path for each implementation (for transactions of size 1 satoshi)

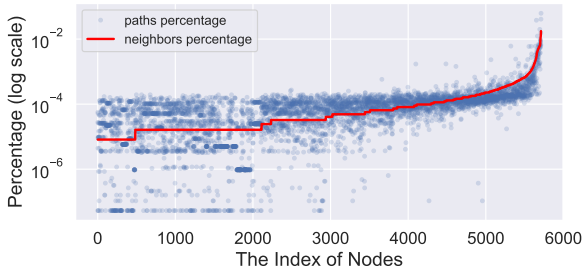

Figure 9: The correlation between the degree and the number of occurrences in paths. The nodes in this graph are ordered according to their degree.

Figure 6 shows the number of channels per node. We find that approximately 25% of the nodes are end-points (are of degree 1), and 14% have degree 2. The distribution is heavy-tailed and some nodes exhibit very high degrees. Furthermore, about 88% of channels are connected to a node with degree higher than 600, and 90% of the rest have the default configurations of base and proportional fee. This suggests that the most sophisticated nodes are usually also the more central ones.

Figure 7 compares the degree of nodes to the total amount of money they have locked. We see that there is a weak positive correlation between these quantities; nodes with higher degree tend to lock more funds.

3.2.3 Evaluation of Routing Properties. We next take a deeper dive into routing properties. In particular, we examine the paths that are selected by the different routing algorithms used in the three main implementations. For every two nodes we determine the paths for transactions of size 1 satoshi. Figure 8 displays the difference in distributions of path lengths that the different algorithms create.

In Figure 9 we see the correlation between the degree of nodes and the percentage of nodes pairs that route through them. Not surprisingly, high degree nodes appear on more routes.

3.3 Feasibility of the Attack

We now evaluate the feasibility of a DoS attack, in which the attacker's goal is to hijack as many routes as possible. We consider two main scenarios:

- **Collusion by existing central nodes:** We consider the case that a small number of highly central nodes collude and jointly launch a DoS attack using their existing resources and connections.
- **External attacker:** An attacker joins the network, creates new channels to existing nodes and "hijacks" routes using low fees and other channel properties, introducing an attractive alternative to existing routes.

In what follows, let us assume that all pairs of nodes in the network attempt to transfer 1000 satoshis (around 0.1$ in May 2020) between them exactly once. For our analysis, we compute the percentage of disrupted pairs of nodes.

Note that we decided to measure transactions between every pair of nodes, since the real distribution of payments is unknown: Transactions in the Lightning network are private by design. It is hard to infer the real distribution since (i) information about transactions is hidden in the private state of channels and since (ii) routes are obscured by onion encryption. Following this idea, our analysis will examine the potential to disrupt transfer between a large fraction of pairs of nodes.

3.3.1 Colluding Nodes. Figure 10 plots the cumulative centrality of nodes: the number of paths going through the k most central nodes. We can see that the five highest ranked nodes can disrupt roughly 60% of all pair connections, and that there are only minor differences between implementations. Clearly, if these nodes collude and start a DoS attack, they will cause major disruptions to the network.

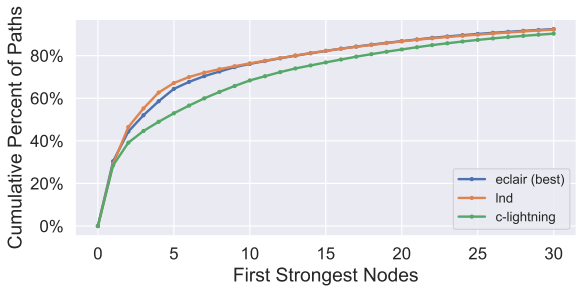

Figure 10: Percentage of paths that go through the most common nodes (assuming transaction sizes of 1000 satoshis)

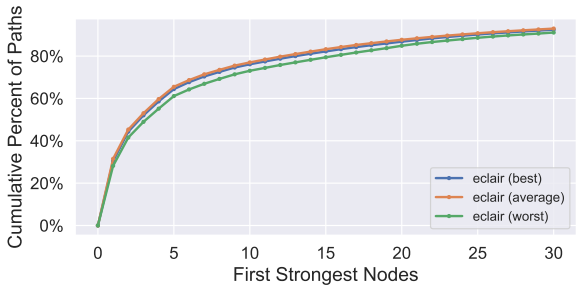

Figure 11: Percentage of paths that go through the most common nodes. Average: increase the probability to hijack a created path; Worst: increase the probability to hijack nodes (all possible paths between the two nodes).

As *Eclair*'s implementation chooses uniformly between the best three routes, we dive deeper with respect to that implementation. If even one of the top three paths between a given pair of nodes does not pass through the attacker, there is a chance that the payment will succeed. Therefore, in Figure 11, we examine three metrics: (i) [Best] The fraction of hijacked best routes (lowest weight route of the 3 options), (ii) [Worst] the fraction of pairs for which we hijack ***all*** the top 3 routes, in order to build an attack that always works, and (iii) [Average] the *expected* fraction of hijacked routes from the top 3. The main lesson from the figure is that all metrics are very similar. Thus *Eclair*'s randomization between the top 3 routes helps very little to avoid attackers.

Digging deeper, the figure shows that (iii) yields the highest hijack rate, (i) the second highest and (ii) the lowest. Figure 12 illustrates a possible explanation to the results in Figure 11.

3.3.2 An External Attacker. We now consider attacks by an external adversary that creates links to the network, in order to hijack as many paths as possible (i.e., to maximize his centrality), and eventually perform a denial-of-service attack. We measure the success of the attacker as a function of the number of new channels that he creates (which directly represent the cost of the attack).

Figure 13 shows the consequences of an attack on the network for different implementations. More specifically, we establish channels from a single attacker node to the nodes identified by Algorithm 3 (which will be discussed broadly in Section 3.1) and calculated the

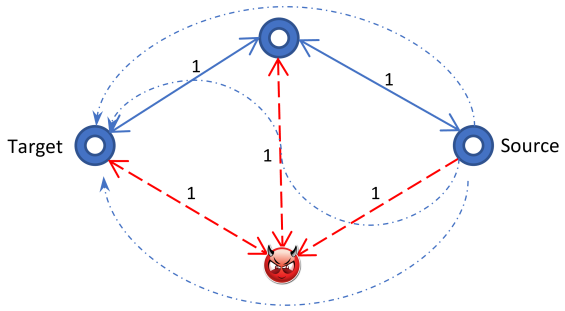

Figure 12: If the attacker creates the red edges, the best 3 routes are illustrated in blue dashed lines. The approach to "hijack the best path" results in the value 0.5 (one of the two best paths passes through the attacker). The approach "hijack all the top 3" results in a value of 0 (as one path does not pass through the attacker), and "hijack as many from the top 3" gives a value of 0.66.

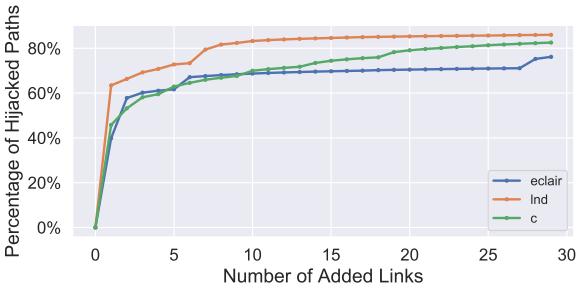

Figure 13: Number of channels (with zero fees and minimum delay) we need to create to hijack the paths of transactions of 1000 satoshis.

percentage of hijacked paths, comparing the different implementations. Note that in both *Eclair* and *C-lightning* the routing algorithm is probabilistic. We therefore use the average success rate of the attacker over of multiple runs for each path.

Alternatively, we can also consider the hijacked routes with respect to all the available connections between two nodes. Figure 14 shows that the adversary sometimes creates new paths between nodes that were not previously connected. These new paths now become available as the route weight and number of hops are decreased (below the threshold used by *lnd*).

Figure 15 shows the effectiveness of the weight-fuzzing method of *C-lightning*. We used the weight function without any fuzzing in order to greedily find the channels that the attacker should create, and then evaluated our results against routing with different fuzz factors (choosing the fuzz within this range randomly 4 times per pair). The figure indicates that the re-introduction of the default fuzz (±5%) does little to prevent the attack. Our suggested explanation is that the fuzz multiplies only the channel's fee, which is very low, and thus does not substantially change routing decisions.

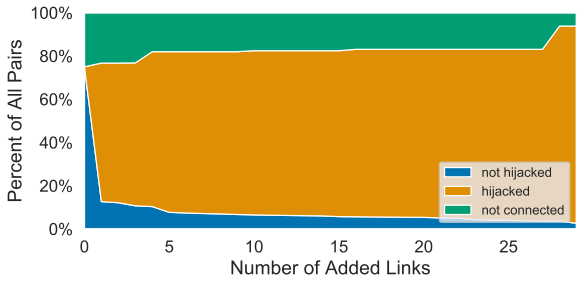

Figure 14: Disconnected and hijacked nodes (*lnd*)

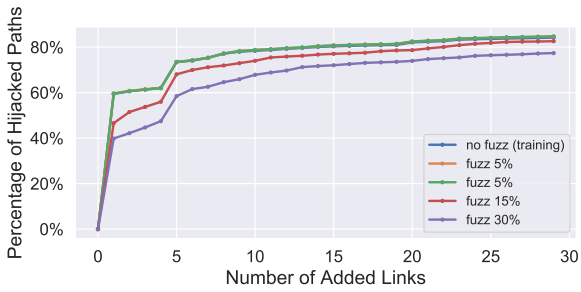

Figure 15: The hijack percentage in the trained sampling (with no fuzz) compared to the percentage in other samples with different fuzz magnitudes.

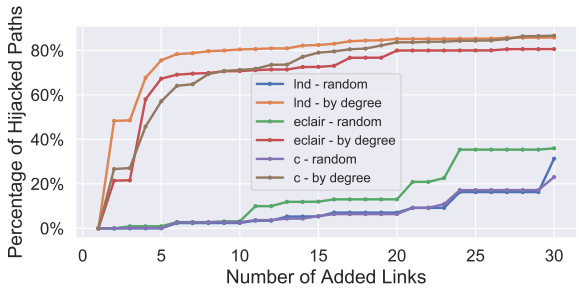

Figure 16: The hijack percentage of each implementation for the attacker that creates channels using a strategy that is not implementation specific.

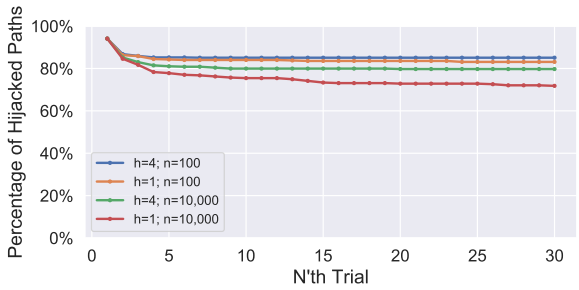

Figure 17: The hijack percentage of *lnd* with the attacker's best 30 channels, considering different penalty parameters

In addition to the above insights, in which we used Algorithm 3 (which assumes knowledge of the implementation of nodes), Figure 16 describes an attacker that uses a channel creation strategy which is not based on a specific implementation. In this figure the attacker chooses the channels based on two "absolute" algorithms: (i) he chooses nodes uniformly at random, or (ii) he chooses the nodes with the highest degree. The first strategy provides us with a "control group" or baseline for our other experiments. The second way of choosing connections, gives us two important insights: The first is that the attacker does not need to know the distribution of the current nodes' implementations in order to attack the system efficiently. The second is that the topology plays an important role for the vulnerability of the network, not only the routing algorithms.

Figure 17 shows the effectiveness of the *lnd*'s penalty, which "punishes" channels over previously failed routes relatively to the time that passed since the last routing attempt through these channels. In these experiments, we find that it takes time for a transaction to fail. In particular, we examined the "punishment" formula: $\frac{n}{0.6-\frac{0.6}{2^h}}$ (in *lnd* it is $n = 100$) and parametrized both h (number of hours between trials) and the fraction's numerator n. The figure compares the impact of increasing the time between the trials and changing n. As we can see, the effectiveness of this method is questionable, with the current default parameters, as well as with other parameters that we considered. Since senders cannot know exactly which node along the route had failed (due to the onion routing), the node is forced to "punish" all channels. Unfortunately, it is often the case that attractive routes are very short, and there is only a single route to the closest "hub". The node then penalizes all routes with this prefix equally, and the attacker's channel is still much cheaper than alternatives. An example is the base fee: one of the parameters to the channel's weight. The attacker creates channels with 0 base fee, while the default is 1000. It is hence much preferred, even when given the max penalty, which is around 400.

3.4 Amplified Attack With Delays

We now show a way to amplify the DoS attack, by increasing the time that the attacker holds the hijacked transaction (the delay parameter). We suggest the following enhancement: the attacker will report a high delay value for his node, which will then affect the delay of all preceding HTLCs in the path (recall that delays accumulate in the reverse order of the path to guarantee intermediaries that the outgoing HTLCs expire before the incoming ones).

Note that there is a trade-off for the attacker, because this delay is one of the properties used to calculate the channel's weight, it makes the route less appealing. Higher delay values imply a stronger attack that will work on fewer pairs of nodes. Figure 18 shows the hijack rate when the delay of the 30 channels that were created earlier is increased. Note that there is a large drop around the delay of 144 blocks, which is related to the fact that many nodes use lower values as their delay (Figure 4).

Figure 18: The hijack percentage when we create edges for increasing delays, using the top 30 new links from Figure 14

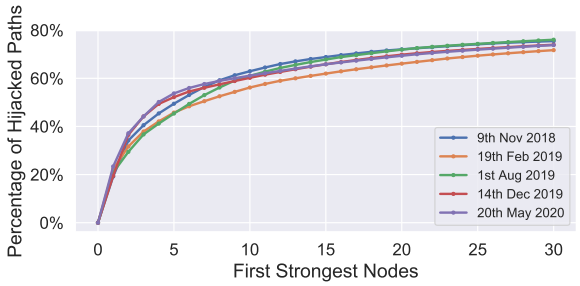

Figure 19: The hijack percentage of the strongest nodes in the network across different timestamps.

3.5 Validation Across Time

In order to validate our results, we use snapshots of the topology over the last year at five points in time: 9th November 2018 at 12:00pm, 27th February 2019 at 11:00am, 27th July 2019 at 17:00pm, 14th December 2019 at 16:30 pm and 20th May 2020 at 16:30 pm (all times are in UTC+2).[9]

The following results show the persistence of the above attacks over these timestamps. Figure 19 shows the number of paths going through the k most central nodes, and Figure 20 considers the external attacker that creates links to the network. We see that the discussed attack is indeed something that is persistent across past states of the Lightning network, and its impact remains high.

4 ANALYSIS AND OPTIMIZATION OF ATTACKER STRATEGY

Having demonstrated the feasibility of the attacks empirically, we now explore the optimization problems underlying the attack from an algorithmic perspective. To this end, we propose an analytical model for the adversary. In particular, we will show that while determining the best adversarial strategy is an NP-hard problem, efficient polynomial-time approximation algorithms exist. To this end, we revisit the connection to graph centrality theory, which turns out to come with a twist in our setting.

[9]The initial release of the Lightning Network was in 15th March 2018

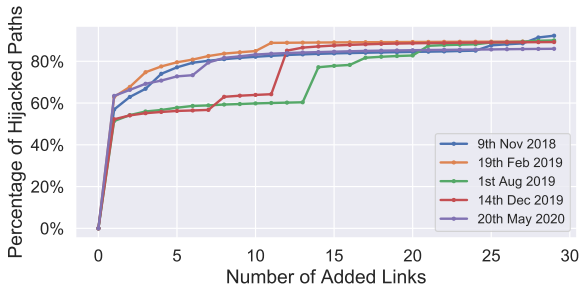

Figure 20: The hijack percentage of an attacker that adds new links across different timestamps.

4.1 Preliminaries

Let V be the nodes that participate in the network and E be the channels, i.e., $(u, v) \in E \subseteq V \times V$ are nodes with established channel.[10] A valid path from a source node $s \in V$ to a target node $t \in V$ is a list of edges $((u_1, v_1), \cdots, (u_n, v_n)) \in E^*$ where $u_1 = s, v_n = t$ and $v_i = u_{i+1}$ for all i.

The path selection algorithm $\mathcal{A}_W$ is an algorithm with the inputs: source node, target node, and the channel graph. It returns a valid path from the source to the target. The centrality of a channel e is the percentage of the network's transactions that pass through this channel. In the same way, define the centrality of a set of channels $e_1, \cdots, e_n$ (note: this is not necessarily the sum of their individual centralities). Denote this function with $C : 2^E \longrightarrow \mathbb{R}$. Note that although this definition is close to the notion of *betweeness centrality* [10], we consider here routing algorithms that do not simply choose the min weight path, like *Eclair*'s top-k randomization.

Note that in this simple model, the transaction size is constant and all the channels have enough capacity to relay it. Therefore it should be considered as a subgraph of the topology that contains only relevant channels.

4.2 Attacker's Algorithms

In general, computing an optimal attack is hard, as the problem of computing optimal link additions is already NP-hard for shortest paths, i.e., betweenness centrality [7]. We hence explore the possibility of polynomial-time approximation algorithms: algorithms which are fast enough to scale at least to all the nodes and channels of the Lightning network (about 4000 nodes at the moment).

In the following, we will explore an opportunity introduced by submodularity, and consider the connection to the problem of betweeness maximization with bounded budget [5].

Lemma 1. *The centrality of* ***existing edges*** *for a given node is a non-negative, monotone, sub-modular function. That is, for* $\forall A, B \subseteq E$ *it holds that* $C(A) + C(B) \geq C(A \cup B) + C(A \cap B)$

Proof. Recall that $C(A)$ is the number of transactions that go through the channels in A. The non-negative and monotone properties follow directly from the definition. Regarding the sub-modularity, we consider the different cases: (i) We count transactions that go through only one of A and B exactly once, on both sides of the

[10]We are not interested in the P2P network itself, only in the channels graph

equation. (ii) Transactions that do not go through A or B, we do not count on both sides. (iii) Transactions that go both through A and B and that are in $A \cap B$, we count twice on every side. (iv) Transactions that go both through A and B, but that are not in $A \cap B$, we count twice on the left side, but only once on the right.
Overall, the left side can be larger then the right side, as desired. □

Remark 1. *The above lemma can be rephrased also to $A, B \subseteq V$ (a set of nodes instead of edges).*

Lemma 2. *The centrality of a node creating **new edges** is a non-negative, monotone, sub-modular function. I.e. $\forall N_1 \subseteq N_2 \subseteq V$ and $x \in V \setminus N_2$, denote by A, B the sets of new channels that connect N_1, N_2 to a new node v, respectively, and e that connects x to v. Then it holds that $C(A \cup \{e\}) - C(A) \geq C(B \cup \{e\}) - C(B)$.*

The proof is equivalent to the proof of Theorem 5.2 in [7]. The key ideas are: (i) If we consider two sets of new edges $X \subseteq Y$, then the distance between every two nodes in the graph with the new edges from X is greater equal the distance when adding Y. (ii) If all new edges are connected to only the attacker's nodes, then the attacker's centrality when adding X is less than or equal to its centrality when adding Y. (iii) Strong inequality in (i) implies strong inequality in (ii).

Let us now consider a repetitive attack, in which our goal is to attract others to always route through our node. To achieve this goal, we will add many edges with 0 fees and delay. Each such channel bears some costs for the attacker due to the need to lock funds. To decrease costs we therefore wish to minimize the number of channels.

Function `GreedyApproach`(k, f, E, V, $\bar{v}$)**:**
  for $i = 1, \cdots, k$ **do**
    $e = \arg\max_{v \in V} f(E \cup \{(v, \bar{v})\})$
    $E = E \cup \{e\}$

Algorithm 1: Greedy perspective to find k channels that maximize the function f

Theorem 1. *A greedy algorithm that given edges E, node n and number k, iteratively finds an edge e ($e \in E$ in the existing edge case, or $e \notin E$ in the new edges case) that maximizes the centrality of n and updates $E = E \cup \{e\}$ (Algorithm 1), gives a $1-(1-\frac{1}{k})^k$ approximation.*

Proof. As in § 4 (corollary of Prop. 4.3) of [29], we can apply the greedy heuristic on the function C, which is a sub-modular set function according to Lemmas 1 in the existing edge case, or 2 in the new edges case. □

It remains to show an efficient method to calculate $\arg\max_e f(E \cup \{e\})$. This can simply be achieved by dynamic programming: find the best edge to add and update the state accordingly. Algorithm 2 describes this idea.

In our algorithm, we made some further improvements, based on the fact that there are no valid paths between **all** the pairs (because of defaults of max hops or max fee). See Algorithm 3 for details.

It is important to notice that the above algorithms are indeed not optimal and are just an approximation. This only strengthens our results: these algorithms yield, in practice, very good results

Function `Preprocessing`(E, V)**:**
  $dbPaths = \emptyset$
  $dbVertexes = \emptyset$
  for $v \in V$ **do**
    $dbPaths$.update(perform dijkstra and get shortest paths and weights to v)
  $dbVertexes$.update(map between vertex to all the participated paths)

Function `FindNextNaive`(E, V, $\bar{v}$)**:**
  $best, value$ = null, 0;
  for $candidate \in V$ **do**
    $counter = 0$
    for $src, dst \in V \times V$ **do**
      if *shortest(src, candidate) + shortest($\bar{v}$, dst)* $\leq$ *shortest(src, dst)* **then**
        $counter$ ++
    if $counter > value$ **then**
      $best, value = candidate, counter$
  return $best$

Algorithm 2: calculate $\arg\max_e f(E \cup \{e\})$ efficiently

Function `FindNext`(E, V, $\bar{v}$)**:**
  $best, value$ = null, 0;
  for $candidate \in V$ **do**
    $counter = 0$
    for *src with path to candidate* **do**
      for *dst with path from src* **do**
        if *(src, dst) already been hijacked* **then**
          continue
        if *shortest(src, candidate) + shortest($\bar{v}$, dst)* $\leq$ *shortest(src, dst)* **then**
          $counter$ ++
    if $counter > value$ **then**
      $best, value = candidate, counter$
  return $best$

Algorithm 3: Our implementation of findNext, minimizing runtime and the number of calls to the database

(for the attacker), and more sophisticated attackers may inflict even more damage.

5 LESSON LEARNED - SUGGESTED IMPROVEMENTS

Let us now explore methods to increase the robustness of the network, and at least partially address the tradeoffs observed above.

We suggest first ideas based on the empirical analysis that we performed in §3.3. We will focus on insights that aim to increase the cost of a successful hijack attack. In Figures 21,22,23 we show the impact of slight changes to *Eclair*'s weight function.

The first lesson is related to the vulnerability of *Eclair* to the delay attack. Here, the weight is determined by multiplying the channel's parameters with the fee. This creates a tradeoff between

the fee and the delay: multiplying the delay and dividing the fee by the same factor, will result in the same weight, but will strengthen the attack. Therefore we suggest to either multiply the delay by the total amount of the transaction, or to add it to the weight function as another summand.

The second lesson is how to create a non-deterministic routing algorithm. *C-lightning* adds noise to the channel's fee (fuzzing). As we saw, this has a low effect in case of an attack because of the exceptionally low fees set by the attacker. On the other hand, *Eclair* chooses a path uniformly from the set of best paths, which also has a low effect in case of an attack because the attacker can create many different low-weight paths using very few resources. Therefore, we suggest to add fuzz to the total weight of the channel, and avoid choosing one of the top-k paths.

Additional lessons are worth pointing out: (i) older channels are better because they came with a cost: interest rate on locked liquidity (thus they are less likely to belong to the attacker, or they exact a cost on it); (ii) high capacity is safer than low capacity; (iii) high delay is important (and should be penalized).

We have considered adding the betweeness rank of channels to the weight calculation, but we suspect that it will make the routing algorithm computationally expensive, and therefore diverges from the goal of creating simple improvements to existing weight functions.

Improve Eclair One may consider more rigorous approaches to creating a weight function that will be resilient to hijacking attacks and preserve important network properties (such as connectivity, low fees, etc.). Here, we will not suggest an optimal weight function, but only try to improve upon existing ones, following the structure of current implementations. We base our suggestion on *Eclair*'s weight function that was presented in §3.1.3.

We suggest to take into account the following channel properties. Weight parameters should be determined according to the network and the user's configuration. The general structure is:

$$fee = ams[i+1] \cdot p[i].propFee + p[i].baseFee$$

$$\begin{aligned} weight[i] = scale \cdot (&normalizedDelay \cdot delayRatio \\ &+ normalizedHeight \cdot ageRatio \\ &- normalizedCapacity \cdot capacityRatio \\ &- capacity \cdot height \cdot IntrestRatio + \frac{fee}{ams[i+1]} \cdot feeRatio) \end{aligned}$$

We use $scale \sim \mathcal{N}(1, \sigma)$, which is the standard Gaussian distribution around 1 with variance σ^2 and some normalization factor. Note that the fee is not normalized and that the scale multiplies globally, and observe the negative sign in the capacity parameters.

In order to evaluate the above function, we use the following parameters: $\sigma = 0.2, delayRatio = 0.5, ageRatio = 0.5, capacityRatio = 0.3, feeRatio = 100$, and the normalization parameters of *Eclair*. We implemented this weight function and evaluated it using the same experiments as before. The results are presented in Figures 21,22,23.

We note that these results require further exploration, specifically, it is important to evaluate other features of the new weight function, including the average fees for paths that it finds, and the failure rates of paths it selects due to liquidity imbalances. We leave such deeper evaluations for future work.

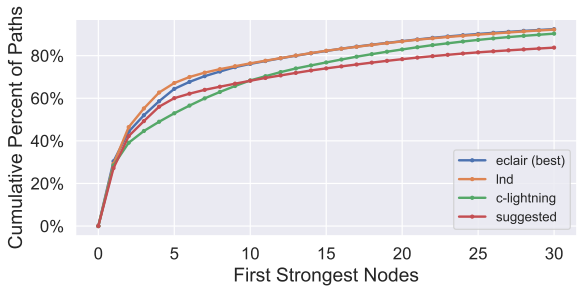

Figure 21: Percentage of paths that go through the most common nodes (assuming transaction sizes of 1000 satoshis)

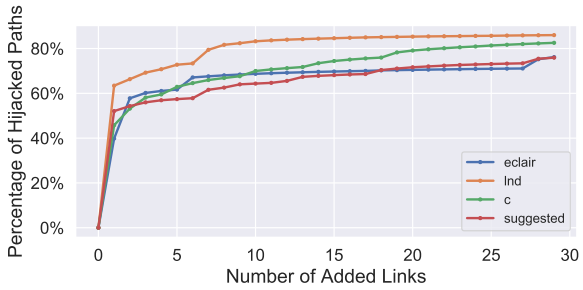

Figure 22: The hijack percentage in the trained sampling (with no fuzz) compared to the percentage in other samples with different fuzz magnitude

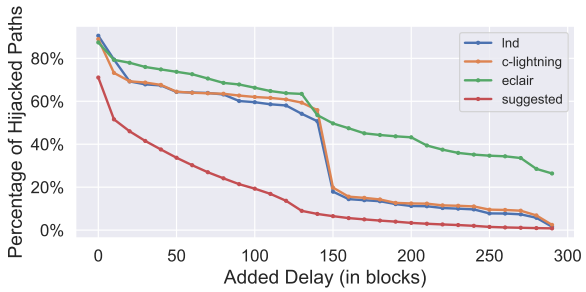

Figure 23: The hijack percentage when we create edges with higher and higher delay, using the top 30 new links

6 RELATED WORK

Since Bitcoin was first deployed in 2009 [27], it received significant interest in academia, including research on its security aspects. While security research initially focused on the analysis of the double-spending attack [35], many additional vulnerabilities were identified later [22].

The P2P network of blockchain systems has been analyzed intensively, mainly for Bitcoin [20], Ethereum [19] (including the centralization analysis in [12]) and the Lightning network [6],[36],[39], but similar analysis exist also in other contexts, e.g., for Skype [14]. Our routing attacks can generally be understood from the perspective of

centrality. Betweenness centrality and its generalizations have been studied much in the literature already [10]. A particularity in our context is that the weight of an edge is calculated using properties of the edges that may be altered by an incentivized attacker. For an overview of research on networking aspects of cryptocurrency networks, we refer the reader to a survey by Dotan et al. [11].

Many attacks on the network level are already known, e.g., the eclipse attacks on Bitcoin [16] and Ethereum [25]. Route hijacking attacks were researched in a variety of fields, such as wireless ad-hoc networks [9], general P2P networks [28], and Bitcoin [1], however, we are not aware of any work on the type of denial-of-service attack considered in this paper. That said, an interesting recent work also discusses path hijacking in the Lightning network [33]. There, the focus is on isolation attacks: the authors consider only the graph of the channels, without referring to the different implementations of the routing algorithms. Our work continues this idea and generalizes it to a general DoS attack, where the attacker tries to damage the transactions of the network and not the nodes themselves. The delay amplifier and the analysis of the differences between the weight functions and randomized path selection has not been researched yet.

Off-chain networks, in particular, attracted many researchers to study possible attacks. Some of these attacks overload the channels with pending transactions to DoS the network [26], create congestion in the underlined blockchain to steal money [15], and deanonymize users with active and passive timing attacks [31], [34].

Routing attacks on different types of networks were studied as well. As an example, in MANETs [37], there exist low-resources routing attacks on TOR [4] that deanonymize its users and create "blackholes" (non-existing nodes that drop routes). Following these papers' perspective, the novelty of our work is fundamentally based on the incentives of the nodes that need to pay fees to channels that participate in the route. This incentive supports the attacker, that will offer lower fees than other channels.

Several solutions were suggested for different types of networks. As an example, in [21] it was suggested to do multipath routing, which is not possible in the "all or nothing" approach, which is typical to cryptocurrencies. In [18] it was suggested to use routing that is based on geographic locations, which is impossible here due to the privacy in the network.

7 DISCUSSION & CONCLUSIONS

This paper identified and demonstrated the feasibility of a novel attack on off-chain networks, in which an adversary can attract many routes at low cost, in order to block a large volume of transactions. We discussed the underlying optimization tradeoffs for both the attacker and for rational defenders, and pointed out the risk of accepting channel connections from untrusted entities. We found that random fuzzing on the fee of every channel only provides weak protection in practice compared to fuzzing the overall weight of the channel. Furthermore, we showed that if the defender considers the fee as a multiplier to the weight, then it will be especially vulnerable to the increasing delay attack.

We see our work as a first step and believe that it opens several interesting avenues for future work. Generally, it will be interesting to analyze properties that weight functions should have, and build optimal functions accordingly. It would also be interesting to study mechanism design aimed at incentivizing nodes to choose routes that will increase the overall security of the network. One may also examine this attack on future features, such as node "switchboards" for message passing, which limit the connections of the attacker.

More generally, we believe that the inherent tradeoffs identified in this paper force us to revisit today's transaction fee based approach to design off-chain networks, and explore radically different solutions. For example off-chain cryptocurrency network providers, which similarly to ISPs charge monthly subscription fees, could provide an interesting alternative economic model that separates fee payments from the underlying relay and routing mechanisms.

Acknowledgments. Research in part supported by the Vienna Science and Technology Fund project WHATIF, ICT19-045, 2020-2024 and by the Israel Science Foundation (grant 1504/17) as well as a grant from the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau.

REFERENCES

[1] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 375–392.
[2] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. 2015. Ripple: Overview and outlook. In *International Conference on Trust and Trustworthy Computing*. Springer, 163–180.
[3] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. 2013. Have a snack, pay with Bitcoins. In *IEEE P2P 2013 Proceedings*. IEEE, 1–5.
[4] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. 2007. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 11–20.
[5] Xiaohui Bei, Wei Chen, Shang-Hua Teng, Jialin Zhang, and Jiajie Zhu. 2011. Bounded budget betweenness centrality game for strategic network formations. *Theoretical Computer Science* 412, 52 (2011), 7147–7168.
[6] Ferenc Béres, Istvan Andras Seres, and András A Benczúr. 2019. A cryptoeconomic traffic analysis of bitcoins lightning network. *arXiv preprint arXiv:1911.09432* (2019).
[7] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. 2018. Improving the betweenness centrality of a node by adding links. *Journal of Experimental Algorithmics (JEA)* 23 (2018), 1–5.
[8] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.
[9] Hongmei Deng, Wei Li, and Dharma P Agrawal. 2002. Routing security in wireless ad hoc networks. *IEEE Communications magazine* 40, 10 (2002), 70–75.
[10] Shlomi Dolev, Yuval Elovici, and Rami Puzis. 2010. Routing betweenness centrality. *Journal of the ACM (JACM)* 57, 4 (2010), 25.
[11] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. 2020. Survey on Cryptocurrency Networking: Context, State-of-the-Art, Challenges. *arXiv preprint arXiv:2008.08412* (2020).
[12] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. 2018. Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998* (2018).
[13] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2019. SoK: Off The Chain Transactions. *IACR Cryptology ePrint Archive* 2019 (2019), 360.
[14] Saikat Guha and Neil Daswani. 2005. *An experimental study of the skype peer-to-peer voip system*. Technical Report. Cornell University.
[15] Jona Harris and Aviv Zohar. 2020. Flood & Loot: A Systemic Attack On The Lightning Network. *arXiv preprint arXiv:2006.08513* (2020).
[16] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 129–144.
[17] Alyssa Hertig. 2018. Coindesk: You Can Now Get Paid (a Little) for Using Bitcoin's Lightning Network.
[18] Zdravko Karakehayov. 2005. Using REWARD to detect team black-hole attacks in wireless sensor networks. *Wksp. Real-World Wireless Sensor Networks* (2005),

20–21.
[19] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Measuring Ethereum network peers. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 91–104.
[20] Philip Koshy, Diana Koshy, and Patrick McDaniel. 2014. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*. Springer, 469–485.
[21] Patrick PC Lee, Vishal Misra, and Dan Rubenstein. 2007. Distributed algorithms for secure multipath routing in attack-resistant networks. *IEEE/ACM Transactions on Networking* 15, 6 (2007), 1490–1501.
[22] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A survey on the security of blockchain systems. *Future Generation Computer Systems* (2017).
[23] Lightning Network In-Progress Specifications [n.d.]. Basis of Lightning Technology (BOLTs). https://github.com/lightningnetwork/lightning-rfc.
[24] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability.. In *NDSS*.
[25] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. *IACR Cryptology ePrint Archive* 2018 (2018), 236.
[26] Ayelet Mizrahi and Aviv Zohar. 2020. Congestion attacks in payment channel networks. *arXiv preprint arXiv:2002.06564* (2020).
[27] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[28] Naoum Naoumov and Keith Ross. 2006. Exploiting p2p systems for ddos attacks. In *Proceedings of the 1st international conference on Scalable information systems*. ACM, 47.
[29] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14, 1 (1978), 265–294.
[30] Raiden Network-Fast. 2018. cheap, scalable token transfers for Ethereum.
[31] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. 2020. Toward Active and Passive Confidentiality Attacks On Cryptocurrency Off-Chain Networks. *arXiv preprint arXiv:2003.00003* (2020).
[32] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.
[33] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. *arXiv preprint arXiv:1904.10253* (2019).
[34] Elias Rohrer and Florian Tschorsch. 2020. Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks. *arXiv preprint arXiv:2006.12143* (2020).
[35] Meni Rosenfeld. 2014. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009* (2014).
[36] István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. 2020. Topological analysis of bitcoin's lightning network. In *Mathematical Research for Blockchain Economy*. Springer, 1–12.
[37] Sheenu Sharma and Roopam Gupta. 2009. Simulation study of blackhole attack in the mobile ad hoc networks. *Journal of Engineering Science and Technology* 4, 2 (2009), 243–250.
[38] Yonatan Sompolinsky and Aviv Zohar. 2013. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive* 2013, 881 (2013).
[39] Saar Tochner and Stefan Schmid. 2020. On Search Friction of Route Discovery in Offchain Networks. *arXiv preprint arXiv:2005.14676* (2020).
[40] Manny Trillo. 2013. Stress test prepares VisaNet for the most wonderful time of the year. *URl: http://www. visa. com/blogarchives/us/2013/10/10/stress-testprepares-visanet-for-the-most-wonderful-time-of-the-year/index. html* (2013).