LockDown: Balance Availability Attack against Lightning Network Channels

Cristina Pérez-Solà*[†], Alejandro Ranchal-Pedrosa [‡]
Jordi Herrera-Joancomartí[⋆][†], Guillermo Navarro-Arribas[⋆][†],
Joaquin Garcia-Alfaro[♦]

*Universitat Oberta de Catalunya
[‡]University of Sydney
[⋆] Universitat Autònoma de Barcelona,
[†] CYBERCAT-Center for Cybersecurity Research of Catalonia
[♦] Institut Polytechnique de Paris, CNRS Samovar

Abstract. The Lightning Network (LN) is a payment network running as a second layer on top of Bitcoin and other Blockchains. This paper presents the possibility of performing a balance lockdown in the LN due to misbehaving nodes associated to a given channel. We formalize and introduce a practical attack, minimizing the economic cost of the attack. We present results that validate our claims, and provide experimental results and potential countermeasures to handle the problem.

Keywords: Bitcoin; Blockchain; Network Security; Off-chain payments channels; Lightning Network (LN); Denial of Service.

1 Introduction

The Lightning Network (LN) is a peer-to-peer (P2P) payment network running as a second layer on top of Bitcoin and other Blockchains. Two nodes in the network can create a payment channel with a fixed capacity and use it to exchange payments between them with low fees. Nodes can route payments through other nodes when no direct channel exists between a payer and a payee. To preserve some degree of privacy, the LN uses an onion-routing protocol for multihop payments. Nodes only publish the minimum information needed to establish the payment routes. Besides the capacity of a channel, its *balance* determines how this capacity is balanced between two nodes (i.e., the bandwidth of the channel in each direction). A node with 0 balance is not able to perform a payment in the channel, since all the capacity is held by the other node.

In this paper, we uncover the possibility of balance lockdown due to misbehaving nodes associated with a given channel. The attack affects the payment channels of the LN nodes. More specifically, an adversary can block LN middle nodes in multipath payments. If successful, the attack gives the adversary a dominant position in the LN, which can be later exploited either for data gathering information or for increasing the benefits of particular LN gateway nodes. We formalize and elaborate some practical evidence of our attack while minimizing the economic cost of the adversary. We present experimental results that validate our claims and discuss potential countermeasures.

Section 2 introduces the necessary background to understand the proposed attack. Section 3 describes the threat model and the attack. Section 4 provides the experimental results. Section 5 discusses countermeasures. Section 6 surveys related work. Section 7 concludes the paper.

2 Lightning Network Background

The LN is a separated P2P network, connected to the main Bitcoin P2P network with nodes that run a LN software client [4,16,5]. Each client maintains a P2P connection with other nodes of the LN and also a connection with a node in the Bitcoin main P2P network. When nodes establish connections with other peers in the Lightning P2P Network, they can open a payment channel in which they exchange Bitcoin transactions without the need for such transactions to be set down in the blockchain. Such payment channels are the core elements of the LN. Details of the LN specification can be found in [15]. High level introductions about the LN exist in the literature [11,10,3]. The background key point for our proposal is the multihop approach, that allows two users without a direct channel to perform a payment.

In the multihop approach, payments at each individual payment channel cannot be performed exactly in the same way that with a single hop. An intermediate user has to enforce he would receive the payment from the preceding node once he has performed the payment to the next one, otherwise he would lose the amount of the payment. The enforcement of this type of atomic exchange between all the nodes of the path (i.e., all simple one-hop payments have to be completed or none can be processed) is performed using Hashed Timelock Contracts (HTLCs) [2]. In an HTLC between the sender A and the receiver B, A can deposit Bitcoins that can be redeemed by B if B can perform a digital signature and provide a preimage of a hash value. Furthermore, the deposit performed by A has an expiration date after which A can retrieve the deposit providing a digital signature. For a two-hop payment, $A \leftrightarrow B \leftrightarrow C$, the idea is that C generates a random value x and sends $h(x)$ to A. A performs the single hop payment to B with an HTLC based on $h(x)$ and B also performs the single hop payment to C with an HTLC based on the same value $h(x)$. In that way, since C knows x, he can redeem the transaction from B, but redeeming the transaction implies revealing the value of x. This implies that B may also redeem the payment from A.

When node B_1 performs a payment to node B_m in the LN using the route $B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_m$, the atomicity needed in such operation implies that all route payments cannot be executed until the last node of the route, B_m, provides the corresponding preimage x of the $h(x)$ included in the HTLC. In a normal scenario, B_m reveals this preimage as soon as he receives the payment in his channel since he wants to collect the payment. However, in case that the payment gets stuck for any reason in node B_i, all payments from node B_1 to node B_i will be locked. To bound the locking time, B_1 sets a total timelock. Such time frame for the payment, determined as an absolute blockheight value, and known as its expiration blockheight, θ, limits the time that money will be locked in case the payment does not succeed. Then, when the payment is being routed every node of the route also decreases such value θ. Each node of the LN advertises for each of its channels, the value δ that will be used for decreasing θ at each hop. With such public information, the payer creates the route with an initial θ ensuring that after subtracting each δ of each intermediate node, the last node will obtain a not expired time, that is $(\theta - \sum_{i=2}^{m} \delta_i) > 0$. Notice that this mechanism allows the payer to bound the time a payment will be locked but, without any other mechanism, a malicious payer could lock the funds of intermediate nodes by setting a large initial value θ. To avoid such situation, each node sets his own T_{max} value that bounds the locking time of a payment. Then, when a node receives a payment as an intermediate node route, if $\theta > T_{max}$ the node will refuse to route the payment and the payer will have to choose another route.

3 LockDown Attack

The proposed attack is focused on a target victim A, a node of the LN. The goal of the adversary is to block the victim A as a middle node in multipath payments. By achieving such goal, an adversary may obtain a dominant position in the LN since blocking some selected nodes may let the adversary be the main gateway to route payments allowing him to have a dominant position that can be exploited either for data gathering information or just for increasing the benefits as a LN gateway node.

To simplify the description of the attack, we omit some of the maximum values that LN implementations introduce. However, we discus how such values impact the cost of real attacks in Section 4. Regarding the notation, and for the rest of the paper, we assume that the victim node is A, the adversary is M and A has open channels with a list of n different nodes, denoted by B_i for $i = 1, \cdots, n$. Furthermore, we denote by C_{AB_i} the capacity of the channel that A and B_i have open and by $balance_{AB_i}$ (resp. $balance_{B_iA}$) the balance that A (resp. B_i) has in this channel. We denote C_{attack} the capacity that M has to hold in channels in the LN to perform the attack.

3.1 Attack Design

The atomicity needed in a multihop payment enforces that the intermediate payments in a multihop route should be held until the complete route is constructed and all payments can be performed together. During the time the route is being constructed, nodes in the route lock the balance of the payment until such payment takes place. With such underlying mechanism, a malicious user can lock a total amount of p balance in a channel AB_i, during the time a payment is being constructed, by sending a payment of value p through that channel AB_i. However, such action, that we label as a naive attack, has two main drawbacks from an adversarial point of view. The first one is related to the cost of the attack and the second one is related to the time the balance is locked.

Regarding the cost, in a naive attack, to block p balance in channel AB_i, the adversary needs to perform a payment of value p so the adversary needs to hold the same capacity that the attack is locking. In that sense, we can define the Attack Effort Ratio.

Definition 1. *The* ***Attack Effort Ratio*** *(AER) is the ratio between the capacity needed to perform the attack and the capacity that the attack blocks, i.e.,*

$$AER = \frac{C_{attack}}{C_{blocked}}$$

The naive attack achieves $AER = 1$ and such attack can be considered a brute force attack since it always can be performed by design of multihop payments. Notice that AER measures the profitability of the attack, and in case an adversary can reduce the AER then, more efficient is the attack, in economic terms, and higher can be the incentive for the adversary to perform such attack.

Regarding the time during which the balance is locked, in a naive attack the adversary only locks the balance during the time the whole payment is being constructed and, in regular conditions, such period is often very short since the final receiver of the payment in a multihop route "executes" the payment as soon as the payment arrives. For more powerful attacks we can define the Δ function.

Definition 2. *The $\Delta(b)$* ***function*** *is a time based decreasing function that measures the total capacity blocked w.r.t. the time during which the attack has been conducted. The block generation time, b, is used as the time unit for this function.*

For instance, $\Delta(0) = C_{blocked}$ since it provides the total capacity blocked at the initial time of the attack. Eventually, $\Delta(b) = 0$ for a large b since the blocking effectiveness of the attack decreases over the time. In a naive attack, $\Delta(1) = 0$ since the capacity is unblocked almost instantly after the payment, long before the appearance of the first block ($b = 1$) after the attack execution.

As we will detail later, an attack is performed through multiple payments. For that reason, the $\Delta(b)$ function is computed taking into account the expiration values of each payment that forms the attack. If we define $\Delta_i(b)$ as the capacity blocked by payment i during b blocks, then $\Delta(b) = \sum_i \Delta_i(b), \forall i \in attack$.

For comparison purposes, we can represent in a single value the **Total Blocked Time** (TBT) of the attack with the following expression

$$TBT = \sum_{b=0}^{\infty} \Delta(b)$$

Once we have described how to perform a naive attack to a single channel, we now describe how to improve the efficiency of the attack, both minimizing the AER and maximizing TBT. We focus the attack goal to block the victim A as a middle node in multipath payments. In that case, the value $C_{blocked}$ is the total capacity of node A in the LN, that is $C_{blocked} = C_A = \sum_{i=1}^{n} C_{AB_i}$. Notice that regarding the attack goal, for blocking a middle node in a payment route it is sufficient to block all incoming balances to A or all outgoing balances from A. In any of both situations, A cannot route any payment. Then, the naive attack over a single node A achieves $C_{attack} = \min\{\sum_{i=1}^{n} balance_{AB_i}, \sum_{i=1}^{n} balance_{B_iA}\}$. Clearly, $C_{attack} \leq C_A$. The AER for such an attack is reduced with respect to the naive attack of a single channel. Notice that, with this approach, the AER reduction cannot be determined by the adversary, i.e., the adversary cannot directly control the balances. However, the AER can be also reduced when the same payment is used more than once to block different channels. In fact, in a multihop payment, a single payment p blocks up to $m \cdot p$ capacity being m the number of hops of the payment route. Another strategy to reduce AER is to construct the largest possible route. However, if the attack is focused on a victim A, not only the length of the route has to be computed but also the route should be kept close to A to ensure all blocking capacity obtained for that route is able to block incoming or outgoing channels of A. As we will see, the best strategy to keep the payment route close to the victim is to perform routes through A with loops as short as possible that return to A. Such possibility will depend on the topology of the payment network in which A is connected.

The improvement of the attack can also be measured regarding the time during which the attack takes place. The objective is to maximize the TBT value. To that end, the adversary can be placed at the end of the route, to hold the payment as much time as possible before the funds of the route are unlocked. As we will see in detail in the next section, this strategy increases the value C_{attack} needed for the adversary.

3.2 Adversarial Knowledge

To implement the ideas described in the previous section, the adversary needs precise knowledge of the network. To construct payments routes which pass through victim A, the adversary needs to know the topology of the network to construct such paths. This information is available using any LN implementation, since it is needed to perform standard payments. Additionally to the topology of the network, the detailed information about balances of every channel are needed to perform the attack. This information can be derived from existing attacks in the literature [6].

Furthermore, to minimize AER, the number of hops of a payment route has to be maximized. Although payment routes in the LN are bounded to 20 hops [17], the exact number of hops that a route may contain is also limited by values T_{max} and δ of each node of that route. Notice that a node does not accept a payment that locks its funds more than T_{max} time and such time is fixed by the adversary but decreased in each hop by the δ of each node. Then depending on the values T_{max} and δ of each node of the route, the total number of hops in a route could be lower than 20. For that reason, the adversary also needs to know the values T_{max} and δ of each node of the network.

3.3 Attack Description

To describe our attack we use a simple scenario where the victim A is a hub between two users, B_1 and B_2, as depicted in Figure 1. Capacity values are $C_{AB_1} = p_1 + p_4$ and $C_{AB_2} = p_2 + p_3$ being p_i the balances in each direction for each channel. The objective of the adversary Mallory, M, is to disrupt the availability of A, by either blocking the availability of incoming links or outgoing links, that is rendering $p_1 = 0$ and $p_3 = 0$ or either $p_2 = 0$ and $p_4 = 0$.

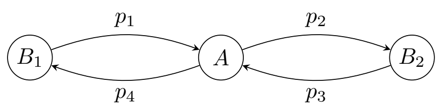


Fig. 1. Simple scenario

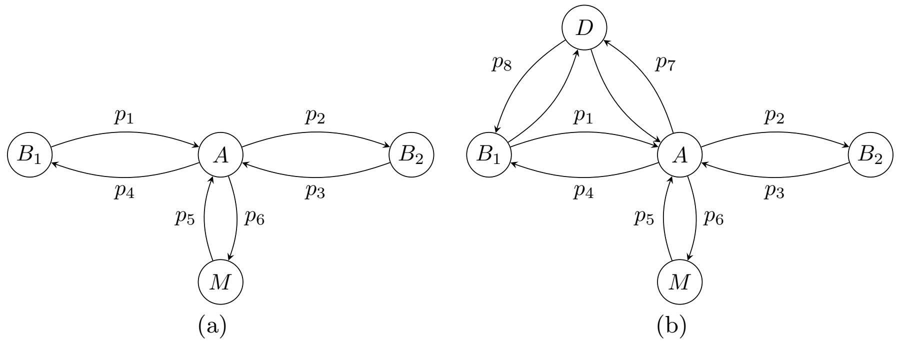


Fig. 2. (a) Simple scenario with adversary. (b) Simple scenario with external node

To perform the attack, M opens a channel with A as depicted in Figure 2(a). The attack complexity depends on the balances between A and B_i and we can distinguish the two following cases:

Shorter loop – The first case is when $p_1 \leq p_4$ and $p_3 \leq p_2$. Notice that with this conditions, $p_1 + p_3 \leq p_2 + p_4$ so M would prefer to block incoming paths to A since C_{attack} is lower than blocking outgoing connections. Then, M can block all incoming path by performing two single payments with a short loop. The first payment will follow the route $M \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow M$ with value p_1 and the second payment will follow the route $M \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow M$ with value p_3. With these payments $balance_{B_1A} = balance_{B_2A} = 0$. Notice that with this scenario the channel that M has to open with A to perform the attack needs a capacity[1] $C_{attack} = C_{MA} = 2(p_1 + p_3)$.

Longer loop – In case either $p_1 > p_4$ or $p_3 > p_2$, then the adversary needs to proceed in a different way.[2] Without lost of generality, assume that $p_1 > p_4$ and $p_3 \leq p_2$ and also that $p_1 + p_3 \leq p_2 + p_4$ so M would prefer to block incoming paths to A. With this balance distribution, M can perform a short loop as before to block the incoming path from B_2 by performing the payment of value p_3 following the route $M \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow M$. However, since $p_1 > p_4$, M cannot perform a payment route $M \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow M$ with value p_1 since the channel AB_1 has $balance_{AB_1} = p_4 < p_1$. At most, M can perform a payment

[1] The capacity that M has to open with A is the double of the payment value since the payment is performed by M but also has to return to M to extend the time that the payment is blocked.

[2] Notice that if both inequations hold, then $p_1 + p_3 > p_2 + p_4$ and M would prefer to block outgoing paths as in the "Shorter loop" case.

with value p_4 through the path $M \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow M$. Such payment locks p_4 but some balance is still available in the channel, precisely $p_4 - p_1$. For M to lock that capacity of the channel, since the path $A \rightarrow B_1$ is already exhausted, M needs another path from A to B_1 with capacity $p_4 - p_1$ with such exact direction. Figure 2(b) shows a simple example in which there exists a node D with opened channels with A and B_1 and such that $balance_{AD} = p_7 \geq p_4 - p_1$ and $balance_{DB_1} = p_8 \geq p_4 - p_1$. In that case, M can perform a second payment with value $p_4 - p_1$ with route $M \rightarrow A \rightarrow D \rightarrow B_1 \rightarrow A \rightarrow M$. This payment will lock the remaining funds of $B_1 \rightarrow A$.

The hard assumption of the existence of node D can be relaxed with the existence of multiple possible paths that all together can route the total $p_4 - p_1$ value. Notice, however, that the payment graph topology hardly determines the existence of such paths.

3.4 AER Minimization

The attack described in the previous section can be improved in terms of AER. For instance, regarding the *shorter loop* case example, the AER value depends on the difference between $p_1 + p_3$ and $p_2 + p_4$. In the extreme case in which, $p_1 + p_3 = p_2 + p_4$, such attack has the worst possible AER since $C_A = (p_1 + p_3) + (p_2 + p_4) = 2(p_1 + p_3)$ and $C_{attack} = C_{MA} = 2 \cdot (p_1 + p_3)$, so $AER = 1$. However, the adversary can reduce such value by relooping the nearer part of each route next to A. Then, if each payment route can be m hops, each original path can be transformed into $M \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow \cdots \rightarrow M$ and $M \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow \cdots \rightarrow M$. With those loops, the total amount that has to be routed is reduced to $\frac{2p_1}{m-2}$ and $\frac{2p_3}{m-2}$ respectively, so $C_{attack} = \frac{4(p_1+p_3)}{m-2}$ and $AER = \frac{2}{m-2}$. Notice that such relooping strategy can also be implemented in the *longer loop* scenario.

3.5 TBT Maximization

We recall that, to make the attack more effective, the TBT value should be maximized. To that end, the adversary takes the advantage of being at the beginning and end of each payment.

As a first node, the adversary can determine the maximum Δ_i for a particular payment i since such value depends on values δ and T_{max} of each node and the node position in the route. The first one, T_{max}, is the maximum amount that a node allows an outgoing payment in a channel to be locked. And the second one indicates the difference, in blocks, that each hop in the route requires. When a node receives a payment, he sets an expiration time[3], θ, for the payment, and subtracts his δ. In case that the resulting value is lower than his T_{max}, then he will keep forwarding the payment, in other case, the node will refuse the payment, the route will be discarded and the payer will need to find another route. Then, the best strategy for an adversary to maximize Δ_i is to simulate the route assuming that each node, instead of discarding the payment, will set the new θ as his T_{max} (see Appendix A for a detailed example).

[3] For simplicity, we assume θ as a relative block height value.

As a last node of the payment, the adversary can hold the payment during the received $\theta = \Delta_i$ value, being sure that the previous node does not cancel the payment before that time —since it fits the proper waiting values of the implementation.

4 Experimental Results

To analyze the feasibility of the proposed attack and provide a proof-of-concept, we need to ensure that nodes in the LN behave in a particular way. Firstly, to minimize AER we need to test if the type of routes with cycles used in our attack can be routed through the nodes of the LN. Secondly, to maximize TBT, we need to verify if the payee of a multihop route is able to retain a payment during a certain period of time before the payment is finally processed locking channels involved in the payment route. Furthermore, we are also interested to implement a mechanism for which the payee can cancel the payment without paying any fee to the routing nodes.

We perform a test in a simnet controlled environment to validate that our claims are correct and that the routes generated in our attack can effectively been deployed in the three most relevant available implementations of the Bitcoin LN, namely LND (lnd), C-lightning (c-lightning) and Eclair (eclair). Results can be found in Appendix A.

Once the feasibility of the attack has been proved from an implementation point of view, we have performed some attack simulations for the LN of the Bitcoin mainnet in order to measure the AER of the attack, the function $\Delta(b)$, and its economic cost. Notice that there is no technical reason that stops us from effectively executing the simulated attacks in the Bitcoin mainnet. However, for ethical reasons, we have not performed the attack on the Mainnet and, instead, we have performed a responsible disclosure to the developers of the LN implementations.

Our simulations will assess the effectiveness of the attack given the actual topology of the network. We base our simulations on the attack algorithm described in Section 3, but in order to provide accurate results, we have taken into account different restrictions that actual LN implementations take over their parameters.

Firstly, we bound to 20 the maximum hops that a payment route may have in the LN as described in [15]. Regarding the length of routes, we assume that the expiration time for a route θ at each hop cannot be lower than zero.

Secondly, all existing LN implementations fix the maximum value of a channel at 16777215 satoshis[4]. Such value may impact the channel that M has to open with the victim A. Since such payment channel needs to have a total capacity of C_{attack}, in case $C_{attack} > 16777215$ then M needs to open more than one channel with A.

[4] This bound is just an implementation parameter. There are already channels in the LN with larger values. The availability of larger channels reduces the number of channels for the attack, as well as total fees to pay for every open channel and the total cost of the attack.

Once such values have been taken into account, to perform our simulations, we take a snapshot of the topology of the LN[5] of the bitcoin mainet on July, 9th, 2019 at 12:00.

4.1 Simulation Assumptions

To execute the attack algorithm described in Section 3, the adversary needs to complement the information of the network graph with further data. The information needed is: the balance of each channel and the values T_{max} for each node of the network.

Regarding the balances, they can be obtained executing the attack described in [6]. However, instead of performing such attack, we have assigned the balances of each channel using different statistical distributions, trying to reproduce the different scenarios that could be found in the network. In order to assign balances to channels, we proceed in the following way: for each channel, first the balance of one of the nodes is randomly selected using one of the selected distributions, and taking the capacity of the channel as the maximum possible value to generate. Then, the balance of the other node in the channel is set as the remaining balance (that is, the capacity minus the balance). Five different distributions are used to assign balances to channels: *deterministic*, *uniform*, *normal*, *exponential*, and *beta*. The *deterministic* distribution always assigns half of the capacity of the channel to each of the nodes; the *normal* distribution is used with $\mu = 0.5$ and $\sigma = 0.2$; the *exponential* distribution uses $\lambda = 1$; and the *beta* distribution $\alpha = \beta = 0.25$.

The value T_{max} is a network node parameter that is not publicly available since it is not advertised by the nodes. However, such value is implementation dependent[6] Hence, by inferring the LN implementation of each node, we can obtain the values of T_{max} for that node. To infer the LN client implementation run by each node, we take into account the fee rate, the fee base rate, and the δ values announced in the nodes' channels policies. As shown in Table 1, default values for the fee rate and δ parameters allow to uniquely identify the LN implementation. We use those values to infer node implementation. Moreover, the default value for the fee base rate is always 1000. We use this third value to further validate that the node is using default values in its policies.

	`lnd (old)`	`lnd (new)`	`c-lightning`	`eclair`
Fee rate	1	1	10	100
Fee base rate	1000	1000	1000	1000
δ	144	40	14	144

Table 1. Values that help infer the implementation a node is running.

[5] Such information can be obtained, for instance, with the instruction `describegraph` of the lnd implementation.

[6] One may assume users changing some LN implementation parameters. However, the T_{max} value is not expected to be one of those easily modifiable parameters.

However, users may indeed change channel policies, or even use different policies for different channels. On the one hand, if a node is not announcing any policy with the fee rate, fee base rate, and delta values corresponding to any of the described implementations, we assume the implementation of that node is unknown. On the other hand, whenever a node announces different policies in its channels but only one of them corresponds to a default behavior, the node is tagged with this implementation. Finally, if multiple policies are announced and multiple default policies are identified, then again the node is tagged as unknown.

Taking this approach, using the selected snapshot of the network, we end up with a small percentage of unknown nodes (11.3%), for which we are not able to properly infer the implementation. In that case, we randomly tag those nodes with one of the three main implementations, with the same percentage distribution than those nodes already tagged. Using such approach network nodes for the analyzed graph have been classified as shown in Table 2.

	nodes (number)	(percentage)
`lnd`	2196	91.04%
`c-lightning`	183	7.59%
`eclair`	33	1.37%
Total	2412	100%

Table 2. Number of nodes, with at least one channel, classified in one of the main implementations for the snapshot graph used in our analysis.

4.2 Attack Simulation Results

To perform the simulation, we focus the attack on one of the most relevant nodes in the network. Such node has 600 opened active channels with a total capacity slightly above 43 BTC. Then, we test the effectiveness of the attack in case such node runs one of the three main implementations, lnd, c-lightning or eclair. For each implementation we also test each of the balance distributions. In order to present more representative results, being the balance distribution a probability distribution, we execute the experiments 10 times and take the mean values.

For each implementation and for each balance distribution, we have performed the attack and measured the AER value of the attack, the percentage of the capacity of the victim that has been blocked, the total channels needed to perform the attack, and the normalized Total Blocked Time, $\widetilde{TBT}$[7] (cf. Table 3). Furthermore, we also have analyzed the Δ function of the attack (see Figure 3).

[7] The normalized TBT, $\widetilde{TBT}$, is defined as $\widetilde{TBT} = \frac{TBT}{C_{blocked} \cdot \max\{T_{max}\}}$, where $\max\{T_{max}\}$ is the maximum default value of T_{max} seen in any implementation. Therefore, $0 < \widetilde{TBT} \leq 1$, and the ideal attack with $\widetilde{TBT} = 1$ would be blocking $C_{blocked}$ capacity during 5000 blocks, that is, more than 34 days.

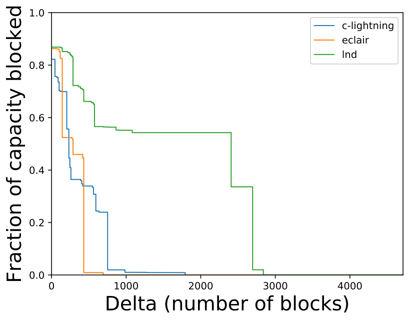
Distibution	Implementation	AER	Blocked capacity	Channels needed	$\widetilde{TBT}$
Beta	lnd	0.291	86.86 %	67.8	0.31
	c-lightning	0.203	82.23 %	49.7	0.07
	eclair	0.584	86.30 %	133.8	0.05
Deterministic	lnd	0.200	100.00 %	52.0	0.44
	c-lightning	0.100	80.40 %	26.0	0.13
	eclair	0.500	100.00 %	129.0	0.09
Exponential	lnd	0.229	92.55 %	55.1	0.36
	c-lightning	0.135	80.22 %	34.0	0.09
	eclair	0.512	92.30 %	123.4	0.07
Normal	lnd	0.230	96.79 %	62.5	0.39
	c-lightning	0.149	84.26 %	39.0	0.11
	eclair	0.479	96.89 %	134.9	0.08
Uniform	lnd	0.268	93.05 %	69.4	0.35
	c-lightning	0.149	82.87 %	45.6	0.09
	eclair	0.557	93.02 %	140.3	0.07

Table 3. Attack results for the different balance distributions.

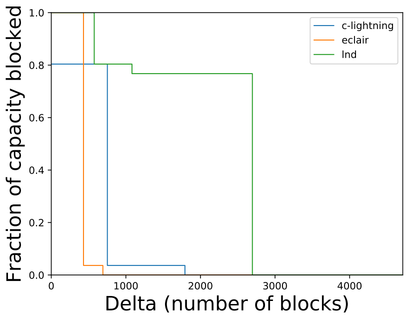
Table 3 shows that the attack is effective in all scenarios (implementations and balance distribution) since the AER is lower than 2 which was the value for a naive attack. Notice that in the worst attack, for a Beta distribution in which the node runs an eclair implementation, the AER is 0.584, which is half of the capacity of the victim to block its 86.30% capacity. In fact, the percentage of the victim capacity blocked is high for all the scenarios, never below the 80%. Moreover, the $\widetilde{TBT}$ also shows that lnd implementations are the ones allowing the adversary to block more capacity over time (as can also be observed in Figure 3).

Figure 3 plots the Δ function which shows which is the amount of time locking the funds. As expected, graphics show that the value of T_{max} of each implementation determines the length of the time. When the victim runs an lnd implementation, 80% of the capacity of the victim can be locked during 287 blocks (almost two days) in any balance distribution tested. But if we look at the 50%, such value is increased to 2407 blocks (more than 16 days). Even for the eclair implementation which has the lowest T_{max} value of all three implementations ($T_{max} = 1008$), 50% of the capacity can be blocked during 287 blocks (almost two days) for all tested balance distributions.

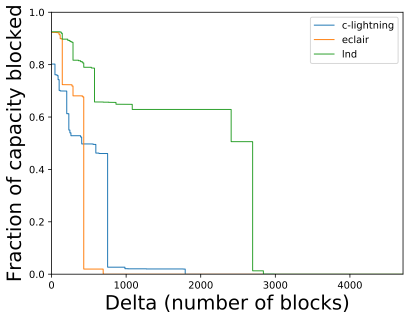
Besides the effectiveness of the attack showed so far, we also measure the economic cost of the attack. For such measure, we take the same methodology than in [6] in which the total cost of the attack can be divided between the entrance barrier cost and the economic cost. On the one hand, the entrance barrier cost takes into account the economic resources that the adversary has to control to be able to perform the attack. Such resources will be completely recovered after the attack has been finished. On the other hand, the economic cost of the attack is the amount of money that the adversary will lose due to the execution of the attack.



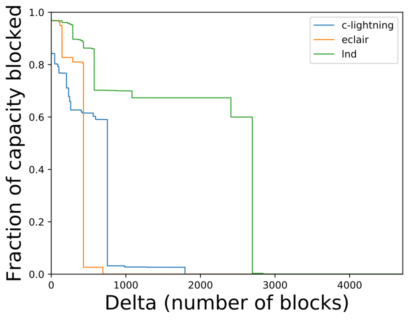
(a)



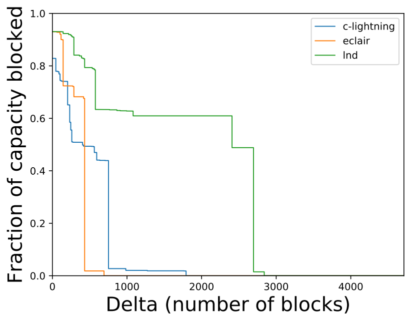
(b)



(c)



(d)



(e)

Fig. 3. $\Delta(p)$ function results for every tested distribution: (a) Beta, (b) Deterministic, (c) Exponential, (d) Normal, (e) Uniform.

Regarding **the entrance barrier cost**, the proposed attack needs to fund one or multiple LN channels with the capacity C_{attack}. Such amount is represented by the channels needed value of Table 3. For instance, the attack for the uniform distribution over c-lightning has an entrance barrier cost of 46 channels (or 7.71753546 BTC) to block 84.25% of the capacity of the node.

With regard to **the economic cost of the attack**, two values have to be taken into account: (i) the fee corresponding to the funding transaction of the channel; (ii) the fee corresponding to the transaction that closes the channel. Regarding the fees of the funding transactions, such cost depends on the number of channels needed to perform the attacks. The cost in fees for each channel depends on the size in bytes of the funding transaction. However, such size mostly depends on its inputs that will vary for each particular transaction, but a funding transaction with a single input can cost as low as 0.00001527 BTC[8]. Secondly, and regarding the closing transaction, it is also difficult to estimate the exact fee for a generic closing transaction, since again multiple parameters may affect such a value. A cost rounding 0.00000909 BTC can be achieved, as can be seen in different existing closing transactions[9].

Notice that we have not included the Lightning fees as a cost because they are never applied (since the payments never succeed). For that reason, the total number of payments needed to perform the attack does not affect the economic cost of the attack.

With such values, we can estimate the economic cost of an attack. For instance, an attack based on a normal distribution assuming an lnd node blocks the 96.79% of the capacity of the node with 63 channels, that means 0.00153468 BTC in fees for opening and closing the channels, around 15 Eur.

5 Countermeasures to Handle the Attack

The main countermeasures are focused on increasing the AER in order to make the attack less profitable. As discussed in Section 3, AER is reduced thanks to the possibility that a single payment performs a route with multiple hops. Furthermore, if the adversary may maintain the route near the victim, the AER is even more reduced. So to not allow the adversary perform such strategy different measures can be adopted.

First of all, loops in a payment route should be minimized or forbidden. In particular, cycles of length two (the ones of the form $A \rightarrow B \rightarrow A$) should be completely forbidden since are the ones that most reduce AER and keep the route close to a possible victim. We argue that imposing such restriction does not damage any possible functionality of the LN. Notice that lightning payments,

[8] See, for instance, transaction: 11b68b276453ac54c23ee49186df78d9895fbfd47071ced6371364abbddcfc6f It is the funding transaction corresponding to the Channel Id 645513381196136448 opened on July 26, 2019, by node 021607cfce19a4c5e7e6e738663dfafbbbac262e4ff76c2c9b30dbeefc35c00643

[9] For instance, Channel Id 624629257244573696 with total capacity 0.05 BTC has been closed with the following close transaction 362235c844533ff7ae0e2fca078b956e82093b92f86010bed51e990d52af6679

even those in a multihop route, are designed to be performed atomically in the sense that they are executed completely or not executed at all. A payment with a subpath of the form $A \rightarrow B \rightarrow A$, once executed it lefts the state between A and B exactly as it was previous to the payment. The implementation of such measure is straightforward even assuming that routing in the LN is performed through onion routing. Notice that in the onion routing approach, every node is aware of the previous and next node in the route so he can reject a route in case both nodes are the same.

Regarding cycles of length larger than two, it is clear that its restriction also increases AER and hinders the attack. Again, although the LN currently routes using onion packets and nodes are only aware of the previous and next hop in the route, additional information transferred between routes and shared by all nodes, such as the hash used in the HTLC can be used to detect that a cycle is passing through a node and reject such possibility. However, in contrast with cycles of length two, longer cycles do not keep the same state of the channel and it can be used for legitimate purposes, like spontaneous payments[10], whose restrictions could impact future LN features.

Besides cycles, increasing the length of a payment route also reduces AER so a possible countermeasure for the proposed attack is the reduction of the maximum length of a route. Such value is set to 20 in the LN specification and it could be reduced to increase the AER of an attack. However, such value directly impacts in the performance of the network since its reduction could potentially discard possible routes for legitimate payments. More testing should be performed before implementing this type of countermeasure.

Another straightforward countermeasure that can be performed to reduce the effectiveness of the attack is the fine tuning of some lightning parameters that, until that moment, are not properly addressed. Such parameters are T_{max} and δ, which have two different implications for our attack. On one hand, despite the maximum hop value (set to 20) T_{max} and δ can effectively determine a lower bound for the number of hops in a route.[11] Since reducing the maximum number of loops increases AER, setting the proper values could potentially prevent the attack. On the other hand, the time value during which a channel or victim can be blocked without the adversary needing to perform any action is also dependent on those two parameters. So reducing the actual values of T_{max} and δ is a countermeasure for our attack since it reduces the time during which the adversary may lock the funds. However, assessing the correct values for T_{max} and δ deserves a detailed and exhaustive analysis and test.

6 Related Work

Recent literature on the security of LN and payment channels mentions *channel exhaustion* and *payments griefing* attacks [12,7]. Rohrer *et al.* [13] suggest an adversarial combination of both techniques, to build an attack that resembles

[10] SPSP, Simple Protocol for Spontaneous Payments, https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-June/001327.html

[11] For instance, a payment route in which all nodes run an eclair implementation can be at most 7 hops.

the *naive attack* reported in Section 3.1. Rohrer *et al.* refer to the combination of channel exhaustion and payments griefing as an attack which '*requires E to first open a channel with a capacity that is equal or greater than the total balance of A's outbound channels*'. This is equivalent to the *naive* scenario reported in our work (cf. Section 3.1, *naive attack*). While Rohrer *et al.*'s attack has always an AER higher than one (i.e., $AER = 2$ if we consider that their attack requires an outbound channel), our work builds upon optimization techniques to obtain attacks with lower AER (cf. Section 4, reduction from $AER = 2$ to $AER = 0.1$). Recall that attacks with an AER higher than one must be considered as brute forcing, with marginal adversarial incentives, in economic terms.

Other differences with Rohrer *et al.*'s work include the experimental setup reported in [13]. Instead of five independent lnd instances, we report in Section 4 experimental work using three different implementations, including lnd, c-lightning or eclair. Some other improvements included in our work is the use of extended network measurements. Previous work (cf. [13], Section II.B and citations thereof) only uses the properties of the topology edges, without taking into account the balance of every node associated to the edges. This is important, since without processing this information, an adversary can estimate the use of routes that may not be used, in the end (i.e., the estimated diameter is wrong).

Privacy issues are also reported in recent literature of payment channels. Tang *et al.* address in [18] the impact of using payment channels w.r.t. privacy preservation. Since users need to route their transactions using other nodes, they must find paths through the payment network, and with enough pre-allocated funds to route their transactions. This poses the problem of hiding the balance of each payment channel node. Joancomarti *et al.* show in [6] the difficulty of hiding such balances. Their work uncovers a balance discovery attack that can be used to deanonymize the precise balance of each network payment node, hence leading to the de-anonymization of the network transactions, in the end. Tang *et al.* and Malavolta *et al.* assume in [18,9] that the adversary is passive, i.e., the adversary observes only the public information released in the network, whereas prior work by Malavolta *et al.* and Ross *et al.* [8,14] considers active adversaries acting as corrupt relay nodes, trying to learn the destination of other nodes transactions.

7 Conclusion

We have addressed the possibility of availability attacks affecting the bandwidth of payment channels of the Lightning Network (LN). We show that an adversary can take advantage of misbehaving nodes associated with a given victim in order to block its ability to act as an intermediate node in multihop payments. The attack can achieve a lockdown during a reasonable time with a low economic cost. We have formalized the attack and provided a practical implementation showing its performance in the LN from the Bitcoin mainnet. The results validate our claims showing the relatively low cost required to lock down an important percentage of the total capacity of a victim. We have discussed potential countermeasures to handle the problem by making the attack less profitable, or less cost-effective attractive for adversaries.

References

1. G. Ausiello and M.Editors Lucertini, editors. volume 109 of *North-Holland Mathematics Studies*, page 27–45. North-Holland, 1985.
2. Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.
3. Giovanni Di Stasi, Stefano Avallone, Roberto Canonico, and Giorgio Ventre. Routing payments on the lightning network. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1161–1170. IEEE, 2018.
4. Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin p2p network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.
5. Elements Project. c-lightning – a lightning network implementation in C. https://github.com/ElementsProject/lightning, 2019.
6. Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin 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*, Asia CCS '19, pages 602–612, New York, NY, USA, 2019. ACM.
7. Akash Khosla, Evan Schwartz, and Adrian Hope-Bailie. Interledger RFCs, 0018 DRAFT 3, Connector Risk Mitigations. Github, 2019. http://j.mp/2m2OvfP.
8. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *NDSS*, 2017.
9. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471. ACM, 2017.
10. Patrick McCorry, Malte Möser, Siamak F Shahandasti, and Feng Hao. Towards bitcoin payment networks. In *Australasian Conference on Information Security and Privacy*, pages 57–76. Springer, 2016.
11. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.
12. Daniel Robinson. HTLCS-considered-harmful. Stanford Blockchain Conference, Stanford, CA, USA, January 2019. http://j.mp/2m7BsKf.
13. Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. *CoRR*, abs/1904.10253, 2019.
14. Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.
15. Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. Basis of lightning technology (BOLTs), 2018.
16. Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. The lightning network daemon, 2018.
17. Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. Lightning network in-progress specifications. bolt 4: Onion routing protocol, 2018.
18. Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks. *CoRR*, abs/1909.02717, 2019.

A Simnet Network

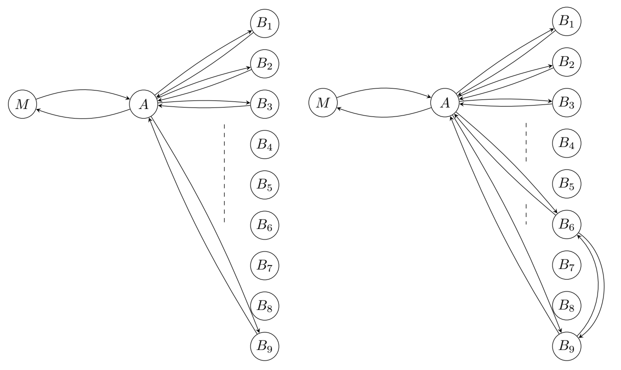
To perform our experiments, we create a Lightning simnet network with eleven nodes, $M, A, B_1, \cdots, B_9$. Node M will be the adversary and A the victim. Nodes $B_1, \cdots, B_9$ will represent victim's neighbors. To test all implementations in our simnet, we run different implementations for different nodes. More precisely, the following configuration has been taken. Nodes M, A, B_1, B_2, B_3 run the LND implementation with version 0.5.2-99-beta, nodes B_4, B_5, B_6 the c-lightning with version v0.7.0 and nodes B_7, B_8, B_9 run eclair with version *version=0.2-SNAPSHOT*. Over this configuration, we have created 10 payment channels, as shown in Figure 4(a).

With this settlement, M performs a payment to himself, following the route $M \to A \to B_1 \to A \to B_2 \to A \to B_3 \to A \to B_4 \to A \to B_5 \to A \to B_6 \to A \to B_7 \to A \to B_7 \to A \to B_9 \to A \to M$.

The correct execution of such experiment proves that the payment has been processed by all nodes and that routes can effectively contain loops. Notice that the loops tested in this experiment are the shortest possible which validates the *shorter loop* case of our attack (see Section 3). Notice that the implementation selected for each node ensures that such behavior is equivalent in all implementations.

Figure 4(b) shows a new scenario where we have added a payment channel between nodes B_6 and B_9. With this scenario, M performs a payment to himself, following the route $M - A - B_6 - B_9 - A - B_6 - B_9 - A - M$.

Again, the test shows that the payment is correctly processed by all nodes and it proves that all implementations can also accept the *longer loop* case, since we have chosen A, B_6 and B_9 all with different implementations.



(a) First scenario (b) Second scenario

Fig. 4. Simnet scenarios

Once we have ensured that routes with cycles are possible to execute in any implementation, we would like to study how to maximize Δ_p, the time that a payee can lock a payment p. Such value can be estimated using information of the nodes that are included in a route. More precisely, values δ and T_{max} of each node and the node position in the route determines the maximum time a payment can be blocked.

In our scenario, the adversary controls both the first and the last node of the route. We first describe how, as a first node, the adversary can determine the maximum Δ_p for a particular route. Then, we will detail how the adversary, as the last node of the route, may block the payment during Δ_p and how, after that time, he can cancel the payment without paying any fee to the routing nodes and, furthermore, leaving all the channels in the same setting than the initial phase of the attack being able to reexecute the attack without any cost.

As pointed out in Section 2, the parameters that determine the actions of each node of the route are T_{max} and δ. The first one, T_{max}, is the maximum amount that a node allows an outgoing payment in a channel to be locked. And the second one indicates the difference, in blocks, that each hop in the route requires. Such parameters are different for every lightning implementation as Table 4 shows. When a node receives a payment, he sets an expiration time[12], θ, for the payment, and subtracts his δ. In case that the resulting value is lower than his T_{max}, then he will keep forward the payment, in other case, the node will refuse the payment, the route will be discarded and the payer will need to find another route. Then, the best strategy for an adversary to maximize Δ_p is to simulate the route assuming that each node, instead of discarding the payment, will set the new θ as his T_{max}. For instance, suppose the following route $M - B_i - B_j - B_k - M$ and assume that B_i is a lnd implementation, B_j is an eclair implementation and B_k is a c-lightning implementation. Assuming the default values of Table 4, the simulation performed by M will start with $\theta = \infty$. When processing the first hop, B_i has a lnd implementation which means $T_{max} = 5000$ and $\delta = 144$ so for that hop, we can compute $\theta = 5000 - 144 = 4856$. In the next hop, B_j runs an eclair implementation, hence $T_{max} = 1008$ and $\delta = 144$. In that case, since the received $\theta = 4856$ is greater than 1008 we will set $\theta = 1008 - 144 = 864$. Then B_k runs a c-lightning with $T_{max} = 2016$ and $\delta = 14$ and the received $\theta = 864$ is lower than 2016, we can calculate $\theta = 864 - 14 = 850$. Since this is the last hop, $\theta = 850$ is the time during which the channel can be blocked. With this procedure, M can compute the optimal θ value that he will include in the first hop to maximize Δ_p. In that case $\theta = 850 + 14 + 144 + 144 = 1152$ will provide a maximum Δ_p, in that case 850

	`lnd`	`c-lightning`	`eclair`
δ	144	14	144
T_{max}	5000	2016	1008

Table 4. Default parameters for different implementations.

[12] Although the θ is an absolute block height value, here we will refer as a relative value to simplify the explanation.