Structural Attacks on Local Routing in Payment Channel Networks

Ben Weintraub
Northeastern University
weintraub.b@northeastern.edu

Cristina Nita-Rotaru
Northeastern University
c.nitarotaru@northeastern.edu

Stefanie Roos
TU Delft
s.roos@tudelft.nl

***Abstract*—Payment channel networks (PCN) enable scalable blockchain transactions without fundamentally changing the underlying distributed ledger algorithm. However, routing a payment via multiple channels in a PCN requires locking collateral for potentially long periods of time. Adversaries can abuse this mechanism to conduct denial-of-service attacks. Previous work focused on source routing, which is unlikely to remain a viable routing approach as these networks grow.**

In this work, we examine the effectiveness of attacks in PCNs that use routing algorithms based on local knowledge, where compromised intermediate nodes can delay or drop transactions to create denial-of-service. We focus on SpeedyMurmurs as a representative of such protocols. We identify two attacker node selection strategies; one based on the position in the routing tree, and the other on betweenness centrality. Our simulation-driven study shows that while they are both effective, the centrality-based attack approaches near-optimal effectiveness. We also show that the attacks are ineffective in less centralized networks and discuss incentives for the participants in PCNs to create less centralized topologies through the payment channels they establish among themselves.

***Index Terms*—Cyrptocurrency; Routing Attack; Payment Channel Network; Lightning Network; Local Routing; Centrality**

1. Introduction

Payment channel networks such as Lightning [1] are the predominant solution to scaling blockchains without fundamentally changing the underlying consensus algorithm [2]. At their core, they rely on the concept of a *payment channel*, wherein two parties who wish to perform transactions off-the-ledger put funds in escrow dedicated to such operations. For parties that do not directly share a channel, payment channel networks facilitate transactions by forwarding the payment along one [1] or multiple [3], [4], [5] paths in the graph created by all existing direct payment channels. The chosen paths need to have the necessary funds to complete the transaction. The process of finding suitable paths is referred to as *routing*, as an analogy to routing in networks, with the caveat that the main goal here is completion of the payment.

Routing payments along one or multiple paths proceeds in two phases: First, all involved nodes commit to paying their successor on the path. Such a commitment implies that they *lock* the funds required for the payment as *collateral*. As a consequence, these funds are not available for any concurrent payments. Second, the payments are finalized. If the payment fails, the collateral is released after a timeout. These timeouts tend to be on the order of minutes or even hours, *e.g.*, for Lightning, 40 Bitcoin blocks in the future from the current block number (at the start of the transaction) — more than 6 hours [6]. Adversarial parties can abuse this mechanism for a denial-of-service attack. By forcing collateral to remain locked in a maximal number of channels for long periods, they can drastically reduce the funds available for other concurrent payments. As a consequence, concurrent payments can fail due to a lack of available funds [7], [8]. Such an attack is called a griefing attack [9].

Previous work [8], [7], [10] on griefing attacks has focused on single-path source routing as this is the current type of algorithm in the Lighting Network. In such a setting the attacker is the source of the payment and no intermediate nodes are involved in the attack. Initiating payments for an attack is costly, however. The attacks exploit either the specifics of Lightning's source routing protocol by selecting a cheap path, or leverage properties of the underlying PoW blockchain (*e.g.*, block size, transaction limit).

It is unlikely that source routing will remain the routing in the Lightning Network as network usage increases. Not only does source routing require storing a snapshot of the global topology at each node, it also prevents intermediary nodes from adjusting the path if a channel does not have sufficient funds, thus leading to routing failures even in the absence of attacks [5]. A naïve solution would be to maintain information about current funds at the source. However, keeping such information up-to-date is infeasible in a large network, because each successful transaction entails changes in the available funds of one or more channels. In a network of one million nodes, every update due to a transaction has to be forwarded at least one million times. If PCNs, indeed, settle thousands of transactions per second, as the VISA network does [11], billions of update messages have to be sent per second; an unacceptable overhead.

To overcome the limitations of source routing, routing algorithms solely based on local information have been developed [12], [3], [13]. Such approaches allow intermediaries to choose suitable channels based on the currently available funds. However, giving more power to intermediate nodes also opens these routing algorithms to attacks. Prior work did not evaluate the effect of denial-of-service attacks on such routing algorithms. Given the severity of such attacks for source routing, it is essential to evaluate novel algorithms in the presence of such attacks before deploying them. To the best of our knowledge

DOI 10.1109/EuroSPW54576.2021.00046

such attacks have not been studied for payment channels routing with local knowledge.

In this work, we focus on attacks against local routing algorithms in payment channel networks, where the attacker is an intermediate node on a payment path. More precisely, we perform two versions of a denial-of-service attack on SpeedyMurmurs [3], a routing algorithm based on local information, which is considered a promising alternative to source routing [2]. In the first variant of our attack, the attacker drops payments entirely. In the second variant, the attacker performs griefing by delaying the payment without causing it to fail. All attacks are performed by intermediaries rather than the source, with the intermediaries being selected strategically based on their position in the network. To conduct the attacks, the attacker needs to know the topology of the network. Note that attackers cannot be prevented from learning the topology as in all currently deployed PCNs (including SpeedyMurmurs), channel opening and closing are recorded on the public blockchain.

In contrast to source routing, SpeedyMurmurs's ability to let intermediaries detect and avoid channels with blocked collateral should make it more resistant to such attacks. Our simulations show the vulnerability of SpeedyMurmurs to attacks. We observe, in all simulated scenarios, that dropping is more damaging than griefing. However, network operators can more easily detect dropping than griefing. This is because in a griefing attack, a transaction may fail due to an attacker delaying a transaction on a partially overlapping path, whereas with dropping, the transaction must be directly routed through the attacker's node.

Our results indicate that selecting attackers by graph centrality is the most effective selection method. Specifically, a centrality-based attacker must corrupt just 0.1% of the nodes to reduce the fraction of successful transactions to near zero, which is only slightly less effective than an ideal attacker that selects the nodes based on the number of transactions they relay. In contrast, our SpeedyMurmurs-specific attack, which selects attackers based on spanning tree-depth, requires 3%, while a random node selection requires 20% to do the same degree of damage. Though SpeedyMurmurs is a tree-based algorithm, it allows the use of channels not in the tree. Thus, nodes in central positions in the tree do not forward as many transactions as nodes with a central positions in the graph, leading to the lower effectiveness of the tree-based attack.

We perform a cost-analysis and show that our most powerful attack, which requires only 10 attackers in a network of 10,000, would cost an estimated one million USD to perform. This may seem expensive, however, as our simulated network resembles the size of Lightning, which as of Mar. 22, 2021 [14] has a capacity of 64 million USD, the lost income due to nearly all the transactions failing could dwarf the attack cost.

Based on the observation that centrality drastically increases the impact of the attacker, we evaluate the attacks in a small-world network with homogeneous node centralities and find that the attack is indeed less effective. Consequently, we discuss incentives for nodes to transform existing payment channel networks to more suitable topologies.

In summary, our contributions are:

- A design of dropping and griefing attacks specific to local routing with intermediaries as attackers.
- A simulation-based evaluation of the proposed attacks revealing that high-centrality nodes allow for highly effective attacks. The strongest attacker uses just 0.1% of such nodes to reduce the fraction of successful transactions to near zero, close to what an ideal attacker could do.
- A cost-analysis of our attacks with current transaction pricing, indicating that the attack would cost 1 million USD in a network worth 64 million USD.
- A strategy for reducing a network's susceptibility to our attacks by incentivizing nodes to build less centralized topologies.

2. Payment Channel Networks

In this section, we introduce the key ideas of PCNs as well as components of PCNs relevant for the remainder of the paper.

2.1. Payment Channels

A *payment channel* defines the relationship between two parties who wish to perform monetary transactions in a common digital currency. In the most general form, a channel is defined by the two parties that establish it and the amount of funds that they make available for transactions to each other. There are three operations that can be performed on each payment channel: i) opening the channel, ii) performing transactions, and iii) closing the channel.

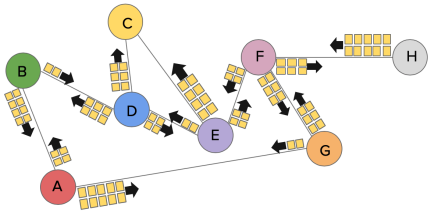


Figure 1: A payment channel network.

Open channel. The *channel balance* is initially established by a *channel open* operation. This operation may be a verified transaction as is the case for Bitcoin's [15] Lightning Network [1] or Ethereum's [16] Raiden Network [17], where channel creation is a blockchain transaction that uses smart contracts to hold the party's funds in escrow. Payment channels may be either bidirectional or unidirectional. We focus on bidirectional payments channels, *i.e.*, payments can be sent in either direction. A diagram of a payment channel network can be found in Figure 1.

Perform transaction. A *transaction* in a payment channel is initiated by one party, referred to as the *sender*, proposing a new state of the channel to the other party, referred to as the *recipient*. The transaction changes the *balance* of the channel, *i.e.*, the amount the sender can send to the recipient. There are a number of mechanisms that enable secure transactions on a channel [2] — mechanisms that ensure the recipient receives exactly the promised funds.

Close channel. Either party on the channel can decide to close the channel. When closing the channel, one or both parties publish the latest state of the channel on the blockchain to regain the coins corresponding to the balance they have on their side. Disputes between the two parties are resolved by the blockchain consensus [2].

2.2. Payment Channel Networks

An open payment channel requires at least one party to escrow funds. As a result, the number of channels a party is willing or able to open is limited. Payment channel networks were proposed to facilitate payments between parties who do not have a direct channel between them. If one considers the parties and the channels between them as a graph, then as long as at least one path with enough liquidity exists between the two parties, they can conduct transactions without having a direct channel by performing a series of pairwise transactions along each channel of each path. If a payment is split over multiple paths, the sum of the partial payments must equal at least the total payment value.

A brief note on semantics: we will follow the convention of Bagaria *et al.* [18] and use the term *payment* to indicate the high-level task that a user might wish to accomplish, and the term *transaction* to mean the components that make up that payment; these components include individual hops along a single path and also payment splits in multi-path routing.

Depending on the implementation, there are various mechanisms in place to guarantee atomicity so that either all of the pairwise transactions succeed, or none of them do [1], [19], [4]. These typically proceed in two rounds: During the *commitment* phase, all involved parties agree to participate in the payment using a smart contract that enforces cooperation later on. In the *payment* phase, parties then finalize the payment if all parties agree to make the commitment. Otherwise, the commitments expire after some time and the parties are able to use those funds for other payments.

When making a transaction, both parties must lock the value of that transaction as collateral which cannot be used for other, concurrent transactions. The key idea of a griefing attack is to have parties lock collateral for longer than intended periods of time. In this manner, the attacker prevents benign transactions from succeeding as the locked collateral is not available and hence the liquidity of the network reduced [7].

2.3. Routing Algorithms

Finding payment paths is one of the core challenges of PCNs. Several algorithms have been proposed with different properties and goals [1], [20], [12], [3], [5], [21]. Many of the algorithms use source routing [1], [20], [12], [3], [5], [21]. Of the remaining algorithms, Flare [20] seems unable to deal with network dynamics and Celer [21] has not been evaluated for more than 100 nodes. The only algorithms based on local information with an in-depth analysis are SilentWhispers [12] and SpeedyMurmurs [3]. Both algorithms provide various privacy properties with SpeedyMurmurs showing considerably better performance [3].

Hence, we choose SpeedyMurmurs for our attack investigation and use Ford-Fulkerson as a baseline. Ford-Fulkerson makes a good baseline for the success ratio, however, it results in an unacceptably high overhead to be a suitable algorithm in practice [3]. In the following, we describe SpeedyMurmurs in detail, for more information on an implementation of Ford-Fulkerson suitable for PCNs, please refer to [3], which is also the implementation we use for our evaluation.

SpeedyMurmurs. SpeedyMurmurs [3] is a privacy-preserving routing algorithm for PCNs based on local knowledge; it consists of three stages. In the first stage, n spanning trees are created. The number of spanning trees corresponds to the number of paths a payment can use. Increasing the number of spanning trees may improve the success ratio and privacy properties but comes at a cost of performance as overhead operations will increase as well. The SpeedyMurmurs protocol uses Perlmans's distributed algorithm for building spanning trees [22]. The protocol starts by selecting the root nodes the details of which are not included in the SpeedyMurmurs paper, but are covered in works such as Byrenheid *et al.* [23]. Next, the nodes in the network organize into a spanning tree with each node connecting to another that is already in the spanning tree — searching through the topology until it finds such a node. The newly joined node then alerts its neighbors of its connection and the index of the tree it is connected to.

In the second stage, which can be interleaved with the spanning tree generations, nodes construct a network embedding for each spanning tree, *i.e.*, each node receives a coordinate from its parent for each spanning tree based on its position in the respective tree. These coordinates enable defining a distance between two nodes, U and V, that corresponds to the length of the path when restricted to the spanning tree. Concretely, for each tree, the root node has the empty vector as coordinate. A child adds a random 64-bit number to the vector representing its parent's coordinate to form its own coordinate. The shortest path between two nodes in a rooted spanning tree is the sum of the length of the paths to their least common ancestor in the tree. Let $|u|$ denote the length of a coordinate u and $cpl(u, v)$ be the common prefix length of coordinates u and v, *i.e.*, the number of leading elements they have in common. Then the shortest path length in the tree is a distance function d with

$$d(u, v) = |u| + |v| - 2cpl(u, v). \quad (1)$$

In the third stage, transactions are routed through the network. The routing algorithm first splits the payment into n randomly sized shares and then routes each of them along a different spanning tree. Nodes forward the transaction shares to whichever neighbor is closest to the recipient according to the coordinate distance of the respective spanning tree, also taking care the channels used have sufficient liquidity. Note that this path-finding algorithm need not follow *only* spanning tree channels; a node should choose its direct neighbor that is closest to the recipient node, which might not be a parent or child in the spanning tree. Channel balances are decreased by the value of the payment that is routed through them. If the balance of a channel reaches zero, then the channel is removed from all spanning trees. A node that is connected

to the spanning tree by a now zeroed channel will leave the spanning tree and reconnect with a non-zeroed channel. The affected subtrees adapt locally by choosing alternative channels; this process is called rebalancing.

As stated above, local routing algorithms are needed to increase the scalability of payment channel networks. However, none of the existing local algorithms has been evaluated in terms of providing availability in the presence of adversarial nodes. Maintaining a high degree of availability in the presence of attacks is of the utmost importance, hence our work focuses on providing the necessary analysis and experimental evaluation of local routing algorithms in the presence of attacks.

3. Attacks against PCNs

In this section, we describe our threat model and attacks we consider in this work. We focus on *internal*, malicious attackers that aim to undermine the availability of the payment service. In other words, the attacker wants to maximize the fraction of failed payments. Adversaries aiming to abuse the protocol for monetary gain or to learn confidential information have been addressed in previous work [24], [3], [4].

3.1. Threat Model

Computation capabilities. We consider an internal, active, colluding attacker that is computationally bounded. More precisely, the attacker has powerful computational resources but cannot break cryptographic primitives.

Incentives. We assume that malicious actors may be interested in damaging the system without monetary gain. This could include nation-state actors attempting to destabilize an economy.

Collusion. An attacker can create nodes that they fully control, and they are able to corrupt formerly honest nodes through means such as social engineering. Attacker nodes are geographically distributed in arbitrary locations and collude with each other. Colluding nodes communicate out-of-band, which might be faster than PCN information. Thus, we assume that an attacker node is aware of any information gathered by other adversarial parties. As motivated in previous work, nodes that are created by the attacker may be arbitrarily connected to other nodes in the network, even those controlled by honest participants [25], [24]. However, they have no access to an honest node's locally stored information.

Attacker knowledge (topology, initial channel balances). As is common in PCNs, an attacker is aware of the complete network topology — all of which is openly readable on the blockchain — this includes all connections and their *initial* capacities, but not any transactions or capacities changed by transactions. While the attacker is aware of the initial capacities, it may not have up-to-date information about the *available* balances on channels to which it is not directly connected. This is because while the initial balance of a channel is public information that is published on channel opening, transactions between two connected parties can change the channel balance without publishing updates (unless a dispute or channel closure occurs). Thus, a node can never be sure of the balance of a channel it is not part of. Rather, it can only say that it is between zero and the total capacity of the channel.

If SpeedyMurmurs is used, nodes are assigned coordinates based on their position in spanning trees. When the attacker establishes a connection, it learns the coordinates of its neighbors for all trees. As every node adds one element to its parent's coordinate, the coordinate length corresponds to the level of the node in the tree. Thus, the attacker can know how close it is to the root.

The attacker does not know the complete tree and cannot necessarily map nodes to coordinates. If the attacker does not have a connection to a node, it cannot tell which of the neighbors of the node are its parent (unless it has a connection to all but one).

Furthermore, the attacker knows the routing algorithm and its properties. For instance, prior work showed that nodes close to the spanning tree root forward more traffic. However, the load on the root node itself is not necessarily high. Recall from Section 2.3 that nodes forward the transaction to the node closest to the recipient in terms of coordinates. If the recipient shares a subtree with the source, the chosen path does not contain the root node. If the recipient and the sender are not in the same subtree, the chosen path might still not contain the root node as there might be a shortcut, *i.e.*, a channel between the two subtrees that is not part of the tree. The probability of finding such a shortcut is reasonably high in a densely connected graph [26].

Attacker placement. We focus on *on-path* adversaries that manipulate payments they are involved in. In contrast, *off-path* attackers aim to affect transactions that are not on their channels, *e.g.*, by sending their own payments along certain paths or crafting the payments in particular ways. Off-path attackers have been discussed in detail in prior work [10].

3.2. Attack Design

Utilizing the above capabilities, the attacker has two options to explore for attack: the selection of malicious nodes in the network topology and the actions performed by these nodes.

3.2.1. Attacker Selection. For the considered *on-path* attackers, selecting the position of the node in the topology is closely related to the number of transactions routed by the attacker and hence the strength of the attack. An obvious strategy is random selection (referred to as a *Random Attacker*). Several other selection strategies are particularly interesting:

Graph-oriented. There are many ways to quantify a node's position within a graph *e.g.*, connectivity, centrality [27], communicability [28], etc. An attacker can choose to optimize for one or several of these properties, and choose their location in the graph accordingly.

Centrality-based Attacker. Of particular interest to us is *betweenness centrality* [27], which for a specific node, z, is the ratio of shortest paths between every pair of nodes that include z. The betweenness centrality $c_b(z)$ of node, z, is given by

$$c_b(z) = \sum_{s,r,z \in N} \frac{\sigma_{srz}}{\sigma_{sr}}$$

where σ_{sr} is the total number of shortest paths between s and r, and σ_{srz} is the number of shortest paths between s and r that include z. In a routing protocol that always selects the shortest path, the betweenness centrality hence correlates with the fraction of payments forwarded via a node z.

Routing algorithm-oriented. Routing protocols typically do not select the shortest path due to constraints such as lack of liquidity or knowledge of topological information. As such, it makes sense to adapt the attack to the path selection of the routing protocol. Concretely, the path selection of the routing algorithm may prefer certain nodes, which should then be chosen for corruption.

Tree-based Attacker. We can exploit that Speedy-Murmurs is based on spanning trees [3]. As stated in Section 3.1, corrupting the root node is not necessarily the optimal strategy in terms of transactions an on-path attacker can affect. Furthermore, there exist protections against attacks on root nodes [23]. Instead, the attacker can easily obtain a position close to the root. Initially, the attacker connects to random nodes. If it does not obtain a position close to the root, it connects to all neighbors of its parent. These include the parent's parent, which will elevate the attacker by one level. The process is then continued until the attacker is connected to the root. Once the attacker knows the root, it can establish a connection to it with multiple nodes.

Transaction-oriented. We also considered selecting nodes based on the balance of their channels and the rate of changes. However, such information is only explicitly known to the parties in the channel. Thus, the information should only be available to the attacker after corrupting the node and it does not seem sensible to base the selection strategy on such unknown information in a real attack.

Ideal Attacker. Given that corrupting the nodes based on the cumulative value of transactions handled is the strongest attack, we consider it as an *Ideal Attacker* and use it as baseline for comparison in our evaluation.

3.2.2. (On-path) Attacker Actions. We consider insider attacks, and are primarily interested in intermediary nodes on the path as they are less detectable. For a node to have a valid suspicion of an attacker, it is enough to simply know which direction the failed payments came from — not necessarily the source.

There are various actions an attacker could take when it is on-path. This is in contrast to Rohrer *et al.* [7] who fully remove their selected nodes from the topology, whereas our selected nodes perform malicious actions, which are harder to detect.

Dropping. An attacker could simply drop a transaction that it *should* forward to the next node in a multi-hop payment. More maliciously, the attacker could refuse to forward the transaction, but still send a confirmation that the transaction has been forwarded to the previous node on the path. If a transaction is dropped during the *commitment* phase (Section 2.2), all previous hops would need to maintain their locked collateral until the transaction expires. We follow Miller *et al.* [29] in assuming that a rational investor's preference is to obtain and use money now rather than later, and therefore a forced restriction on using one's own money constitutes an attack. Locked collateral also cannot be used to route other transactions that may have a higher chance at succeeding, and thereby deprives the collateral holder of potential revenue from fees. Note that dropping is not performed during the *payment* phase (Section 2.2) by a rational attacker, because then they would be forced to pay their guaranteed funds to the node one hop closer to the recipient but, in dropping, would not be reimbursed from the node one hop closer to the sender.

Griefing by delaying. A node can wait to forward a payment to the next node until some specified amount of time has passed or some condition is met. For example, one such condition could be to wait until just before the transaction times out before forwarding it. Doing so will allow the payment to complete, but would force the collateral to be locked up for the maximum amount of time, which constitutes an attack. Performing this attack inconsistently, *e.g.*, on only 50% of transactions, would also make it difficult to detect because of its similarity to network delays. In addition, failures from this attack can be indirect and thus deniable by the attacker. This has a similar effect to *payment griefing* [9], in which an attacker sends payments to colluding nodes that then delay and drop the transaction. Our variant is more general, however, because *any* node on the transaction path can grief by delaying, whereas in traditional griefing, a transaction has to be addressed to the griefing node.

4. Attack Implementation

In this section, we describe the attacks we implemented. First we describe the changes we made to the simulator and then the realization of the attacks.

4.1. Concurrent Transactions

We extend the sequential transaction simulator from Roos *et al.* [3] to build a *concurrent* transaction simulator, which we make publicly available [1]. In the simulator, each channel is capable of locking collateral for a configurable number of concurrent transactions. This is done through the use of threading. Each transaction is executed in a distinct thread that operates on a shared state in memory. Transactions arrive uniformly and begin executing as soon as a thread is available. For each transaction, the configured routing algorithm is executed in two steps. First, a path is found using the methods of the configured routing algorithm. This path discovery is done hop by hop, and with each hop, collateral along the corresponding channel is reserved for the current transaction, which cannot be violated by any other concurrent transaction. To simulate the effects of network delays, a *simulated network delay* of 30 milliseconds is used for each hop during path finding. The delay of 30 ms was chosen as it is similar to delay within Europe or the US (delays vary between 30 to 50 ms depending on the provider). The end-to-end delay is bigger than 30 ms as it is multi-hop. We experimented with a variety of delays, but the results were similar since the dominant factor is the griefing delay of 10 seconds.

Second, when the routing algorithm has found a path with sufficiently high balances and has locked collateral on each channel, it can then complete the transaction in

1. https://github.com/iowaguy/pcn-simulator

reverse order. This payment phase involves unlocking the collateral and updating the channels' balances.

Not only do channel updates occur on every completed transaction, but they also occur during collateral locking. It is necessary to track how much collateral is locked, as locked collateral is not available for other transactions. Thus, the important aspect in deciding whether routing via a channel is possible is not the balance but the difference between the balance and the locked collateral. Locked collateral leads to paths being diverted by insufficient available funds or even to failures if a path with sufficiently high available balances cannot be found. A secondary reason for this information to be maintained is that we need to be able to rollback transactions that have claimed some collateral but were unable to be completed for some reason. Upon aborting such a partial transaction, the channel must be returned to its prior state, while still retaining any changes from any other partial or complete transaction that may have occurred concurrently.

4.2. Attack Implementation

We implement two types of attacks: *dropping* and *griefing by delaying*, as introduced in Section 3. For a dropping attack, transactions are executed normally unless they encounter an adversarial party on the path. If an adversary is selected as the next hop, the payment is immediately marked as *failed* and any collateral locked for that payment will be rolled back.

For a griefing attack, upon encountering an adversary, the transaction is delayed for a configurable amount of time. For our simulations, we find that the ratio between *attack delay* and *simulated network delay* is decisive, and therefore choose a value of attack-delay to be 10 seconds. With a simulated network delay of 30 ms, the attack does not increase in effectiveness with longer than 10 second delays. After the delay time has elapsed, the payment is allowed to continue.

For each of these attacks, we consider three attacker placement methods: *random*, *tree-based*, and *centrality-based*, *i.e.*, by the fraction of shortest paths going through a node [27]. The networkx [30] Python library was used for calculating betweenness centrality. Finally, we also evaluate an *ideal* attacker placement as an oracle for the maximum damage an attack can create on a given dataset.

We implemented the *tree-based* attacker placement as follows. As described in Section 3, an attacker can gain a high position in the tree by opening connections and closing them again if he is not sufficiently close to the root. In our experiments, we assume that the node has already achieved a favorable position by the time the simulation starts. Concretely, if we have x attackers and t trees for our attack, we select the x attackers as the x/t nodes closest to the root for each tree.

5. Evaluation

In our evaluation we set out to answer three questions:

1) How effective are dropping and griefing by delaying attacks?
2) How does the attacker placement in the topology influence its attack power?
3) How effective are the tree-based and centrality-based attack placement strategies when compared to the ideal attacker that has complete information about transaction values and available balances?

5.1. Methodology

As we are interested in the effects of these attacks on large-scale networks, we are forced to simulate — it would be unethical to perform such attacks on operational financial networks. Testbeds are also insufficient for our purposes because they do not let us evaluate at scale. In response to these shortcomings, we generated synthetic datasets with representative network conditions to evaluate our attacks as described in Sec. 5.2.

In all scenarios, we vary the number of attackers. For randomly selected attackers, that amount is varied between 5–30% of total nodes in the network. For the centrality-based and tree-based attackers, we vary the fraction of attackers between 0.1-5% of nodes in the network.

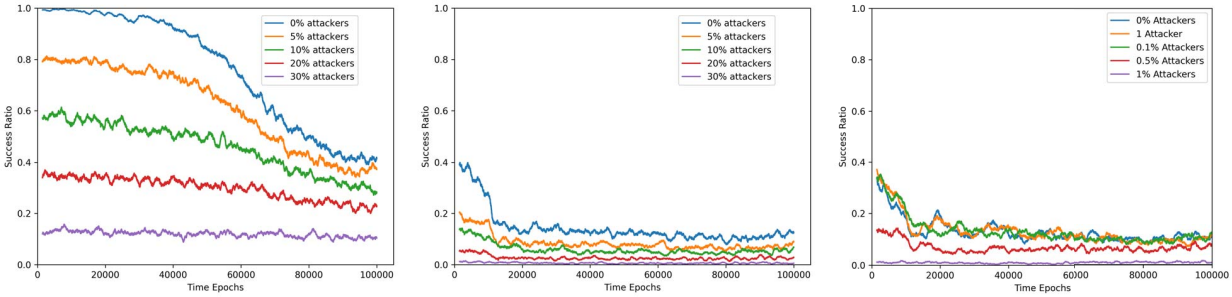
We consider the success ratio to gauge the effectiveness of our attacks, *i.e.*, the fraction of payments for which paths with sufficient funds have been discovered. Results for each transaction are grouped into simulated time buckets called *epochs*; this is based on when the transaction was started, not when it finished. In other words, the results for epoch i contain all transactions initiated in epoch i, regardless of when they terminate. For maximum time-step granularity, we consider each transaction to be started in a distinct epoch unless stated otherwise. All results are presented as a running average over 1500 epochs, unless stated otherwise. Note that our success ratios are considerably lower than those reported in [3] as their data was cleaned such that all transactions were possible. In addition, all the presented success ratios drop over time — this is expected because channels become depleted as transactions traverse them. This has been previously shown in [5], [31], [32].

5.2. Generated Datasets

Existing datasets suffer from either a dearth of successful transactions, a lack of information about transactions, or a small number of nodes. The Raiden Network, for instance, has 37 nodes [33] and Lighting 2337 [34]. Thus, we devise a methodology to generate representative datasets. Our datasets consist of three components: i) the PCN topology, ii) the transaction set, and iii) the initial channel balances.

To generate topologies, we used the ready-made implementations of the Barabási-Albert algorithm [35] for the scale-free graph generation and the Newman-Watts-Strogatz algorithm [36] to generate small-world graphs. Both implementations are from the Python module networkx [30]. We also used a measured topology from the Lightning Network as collected by Rohrer *et al.* [34].

Transaction sets were generated by sampling from representative distributions. For the transaction values, we use a Pareto distribution as it has been shown to reliably represent consumer spending patterns [37]. Sender/recipient pairs were generated by sampling a Poisson distribution independently for both the sender and recipient. A Poisson



(a) Random selection, Ford-Fulkerson (b) Random selection, SpeedyMurmurs (c) Tree selection, SpeedyMurmurs

Figure 2: Success ratios during a dropping attack.

Name	Graph Generation	Transaction Set
Scale-Free	Barabási-Albert	10k nodes, 100k transactions, 0.5 multiplier
Small-World	Newman-Watts-Strogatz	10k nodes, 100k transactions, 0.5 multiplier
Lightning Network	Measured[34]	2337 nodes, 100k transactions

TABLE 1: Description of datasets used in experiments. Unless otherwise stated, our simulations use the scale-free dataset.

distribution was chosen as prior work has shown it to be good for modeling consumer transaction rates [38].

Initial channel balances were generated with the following procedure. Start by assuming that all channels have balance of zero, then calculate the shortest path between each transaction's sender and recipient. For each channel used in the shortest path, add the value of that transaction to its current balance. This process is akin to starting with a fully depleted network, and then routing transactions in reverse time. As a real network is unlikely to have the exact amount of funds needed to route all transaction, we reduce the balances of a fraction of channels by a 0.5 multiplier.

Unless otherwise stated, we used the the scale-free dataset for all experiments. All datasets used are presented in Table 1. More details on limitations of existing datasets and on our generation techniques can be found in Appendix A.

5.3. Impact of Dropping and Griefing

We start by analyzing the dropping attack on both Ford-Fulkerson and SpeedyMurmurs comparing random and tree-based attacker placement. In Fig. 2a, where Ford-Fulkerson is evaluated with a random attacker selection, we see significantly degraded performance for each increase in the number of attackers with no diminishing returns. With more attackers, the success ratio may eventually drop to zero. As displayed in Fig. 2b, which evaluates random attacker selection against SpeedyMurmurs, the success ratio is impacted significantly in the early epochs with more than 5% attackers. Increasing the number of attackers gives rise to a concomitant increase in the attack's effectiveness. Note that in both these plots, as well as all that follow, the success ratio has a consistently negative slope. This is expected because payment senders and recipients are selected randomly from non-uniform distributions, so most channels will be used more in one direction than the other and will, therefore, end up depleted.

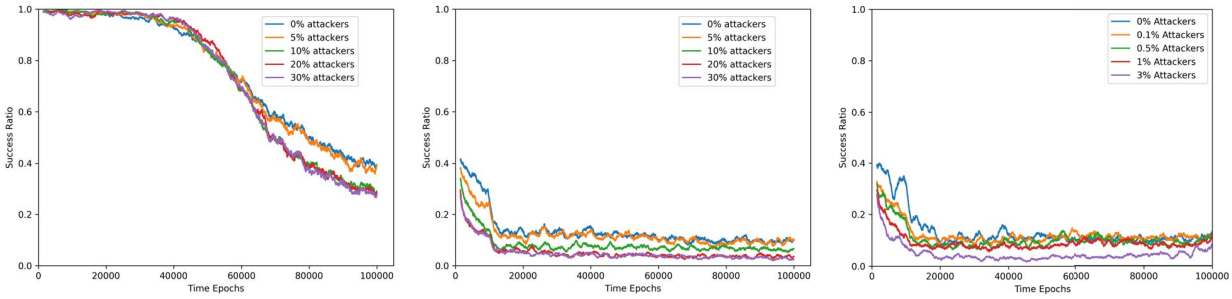
We present the results of our tree-based attacker in Fig. 2c. This attack is specific to the spanning-tree structure of the SpeedyMurmurs routing algorithm, and therefore cannot be attempted against non-tree based algorithms such as Ford-Fulkerson. The results show a much stronger attack than when nodes are chosen randomly. We start to see a noticeable drop in performance with only 0.5% attackers, and with 1% attackers, the performance has dropped to only slightly above zero. The difference in attack severity results from the tree-based attack strategically selecting nodes with a high probability of being involved in many transactions.

Fig. 3 shows the results of our griefing by delaying attack. Against Ford-Fulkerson (Fig. 3a), the griefing by delaying attack shows no effectiveness until the later epochs, and requires at least 10% of the network to be attackers. Fig. 3b shows the results from performing this attack against SpeedyMurmurs. Having 5% attackers leads to a slightly lower success ratio initially but then stabilizes at roughly the same success ratio as a network without adversaries. A higher fraction of attackers leads to a lower success ratio but follows the same pattern of an initially high success ratio dropping and then stabilizing. At some point, the effect per additional attacker seems to decrease. For instance, 30% attackers is only barely more effective than 20%. This is not surprising, because once such large swaths of the network are controlled by attackers, most paths already contain one adversary so that an additional attacker does not increase the number of affected paths.

The tree-based attack is more effective than a random one, the random attacker requires 30% of the nodes to achieve the same impact as the tree-based attack with only 3%.

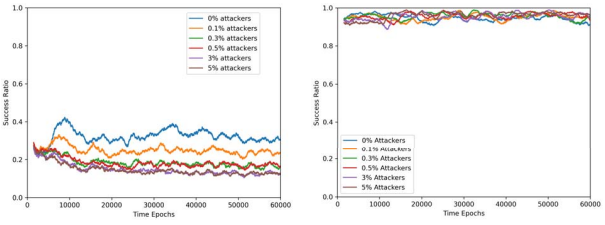
We also present our result for the dropping and griefing by delaying attacks on the Lightning Network topology. Due to space constraints, we include only the griefing attack with attackers selected randomly (Fig. 4) — however, in all cases, the results were consistent with the attacks against synthetic datasets.

We observe, in all scenarios, that dropping is a more effective attack than griefing by delaying. However, network operators can more easily detect it than the griefing attack. This is because in a griefing attack, a transaction may fail due to an attacker delaying a transaction on a partially overlapping path, whereas with dropping, the transaction must be directly routed through the attacker's node.



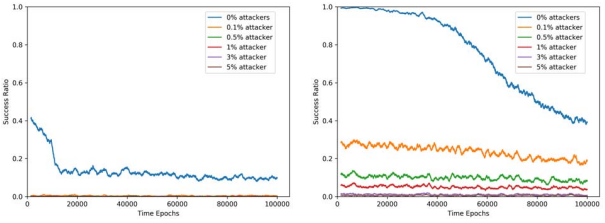
(a) Random selection, Ford-Fulkerson (b) Random selection, SpeedyMurmurs (c) Tree selection, SpeedyMurmurs

Figure 3: Success ratios during a griefing by delaying attack.



(a) SpeedyMurmurs (b) Ford-Fulkerson

Figure 4: Griefing attack against the Lightning topology with generated transactions. Attackers selected randomly.



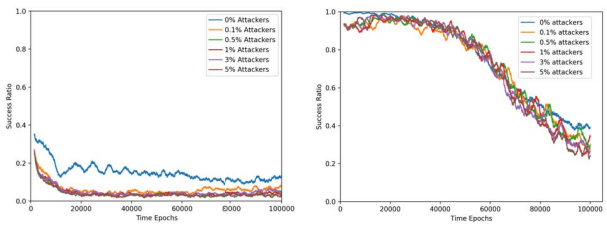
(a) SpeedyMurmurs (b) Ford-Fulkerson

Figure 5: Success ratios during a dropping attack when selecting attackers by betweenness centrality.

5.4. Centrality-based Attacker Nodes Selection

If the attacker has the ability to strategically choose which nodes to corrupt, it can use the available topology information to corrupt the nodes with the highest betweenness centrality. We next evaluate our graph-oriented attack for both SpeedyMurmurs and Ford-Fulkerson (Fig. 5). Against SpeedyMurmurs, this attack reaches maximum efficacy with only 0.1% attackers. As the success ratio is brought down to zero by so few attackers, adding more attackers does not offer any additional benefit. We see a similar increase in efficacy for Ford-Fulkerson in Fig. 5b. With just 0.1% attackers, the success ratio drops more than 70% in the early epochs. More attackers offer diminishing returns, but still bring down the success ratio to almost zero with only 5% attackers.

Fig. 6 displays the effects of griefing by delaying against both SpeedyMurmurs and Ford-Fulkerson when selecting attackers by highest betweenness centrality. For SpeedyMurmurs (Fig. 6a), 0.1% attackers are sufficient to reduce the success ratio to almost zero. Given that the success ratio is already so low, additional attackers do not considerably decrease it further. The effects on Ford-



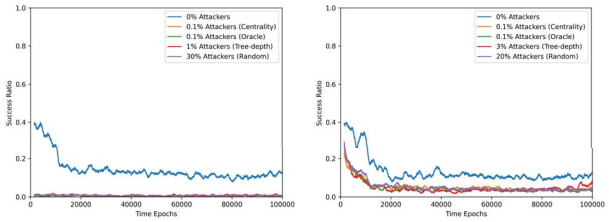
(a) SpeedyMurmurs (b) Ford-Fulkerson

Figure 6: Success ratios during a griefing by delaying attack when selecting attackers by betweenness centrality.

Fulkerson are less pronounced (Fig. 6b), as it can more easily route around failures.

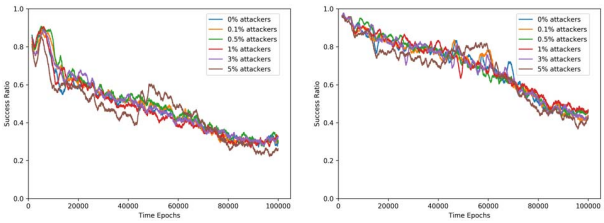
The effectiveness of the centrality-based attacks indicates conclusively that a node's position in the topology does impact attacker power. As SpeedyMurmurs is a tree-based algorithm, it may seem surprising that the tree-based attacker is less effective than the tree-agnostic centrality-based attacker. However, the most important factor in whether an attacker will be effective is how many transactions pass through it. This in turn is affected by the sender and recipient locations within the topology. In our datasets, we used a Poisson distribution for assigning the sender and recipient of each transaction, which means that many of the transaction senders and recipients will be concentrated in relatively few nodes. An attacker near one of those would be much more likely to impact many payments. A transaction from a sender or to a recipient farther from the root may be able take shortcuts before reaching the attackers that are closer to the root (see Section 2.3 for details about SpeedyMurmurs routing), which means that an attacker close to the root will not necessarily be involved in more transactions than other nodes. A centrality-based attacker is not as affected by the distribution of payment senders and recipients, because the high number of paths passing central nodes includes paths with short cuts as well as those without.

Fig. 7 shows a comparison of the number of malicious nodes needed to make an attack effective. With enough attackers, all three selection methods (random, centrality, and tree-depth) achieve optimal efficacy, however, the centrality-based attack is able to do so with nearly as few malicious nodes as the oracle (0.1%), whereas the tree-depth attack requires 3% and the random attacker requires 20%.



(a) Dropping (b) Griefing by delaying

Figure 7: Comparison of how many malicious nodes are needed to reach the maximum attack efficacy for dropping and griefing by delaying attacks.



(a) SpeedyMurmurs (b) Ford-Fulkerson

Figure 8: Mitigation against griefing by delaying; attackers selected by betweenness centrality.

5.5. Attack Cost

There are two ways to insert corrupt participants into a payment channel network: i) corrupting existing nodes or ii) inserting new malicious nodes into the network. Both come at a monetary cost and probably relate to a party's centrality, meaning that achieving a more central position is more expensive. We speculate this is the case because a node with more valuable channels may employ stringent security measures. This assumption only affects our cost analysis, and has no bearing on the attack.

Intuitively, corrupting influential participants, *e.g.*, through malware, should be more costly if nodes invest in better protection, however, to the best of our knowledge, there is no study on the costs of protections employed by blockchain users. Hence, we focus on the scenario when the attacker integrates nodes into the network by establishing channels.

Centrality-based Attacks. As in other approaches [24], we assume that honest nodes are willing to accept requests to open channels, which offer more payment opportunities and potential income in the form of fees, but only if the party initiating the channel opening provides all necessary funds. Thus, for each channel, the attacker has to provide the blockchain fee for a channel opening f^t_{open} with t indicating that the fee depends on the time of the opening. In addition, they have to supply the channel capacity cap, which has to be high enough that parties choose this channel. In Lightning, the average and median channel capacity are 0.03 and 0.05 Bitcoin, which is more than 500 and 900 US dollars as of Dec 6, 2020 [2]. So, cap should be on the order of hundreds of dollars. In comparison, f^t_{open} varies from about half a dollar to more than 10 dollars. Hence, the cost to initialize K channels of the same capacity at time t is

$$cost_{BET}(K) = K \cdot (f^t_{open} + cap). \quad (2)$$

When closing the channel, the adversary regains the funds locked in his direction, *i.e.*, funds they can still spend. However, for a node to forward a payment to the attacker, funds in the other direction are needed, meaning that the attacker has to make payments to achieve a suitable channel balance, optimally with most of the funds available to its partners. As a consequence, the attacker is unlikely to receive a considerable amount of the invested funds back. For our experiments, the total number of channels of the adversary is $K = 1376$ for 0.1% attackers, *i.e.*, the attacker amount in Fig. 7b, meaning the attack cost is close to 1 million US dollars when inserting new nodes.

Tree-based Attacks. In a tree-based attack, the attacker can establish a channel with another node and immediately tear it down upon learning that the node is not a suitable partner due to its position in the tree. Let X be the random variable indicating the number of connections attempts needed to connect to a root node. In addition, let f^t_{close} be the fee for closing the channel. Hence, if a node aims to establish T connections in total with one connection to a root node, the expected cost of the attack are

$$cost_{Tree}(T) = T \cdot (f^t_{open} + cap) + \mathbb{E}(X)(f^t_{open} + f^t_{close}). \quad (3)$$

We observe an average of $T = 3307.7$ for 3% attackers in our experiments. The random variable X depends on the node's knowledge of the topology. Therefore, to achieve the same effect as the centrality-based attack, the cost for the tree-based attack is higher.

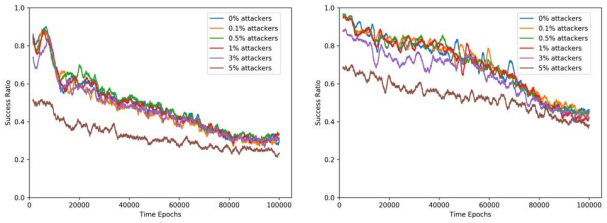
5.6. Mitigation

We focus on the centrality-based attack as it was the most effective attack. Intuitively, the absence of nodes with a high centrality should reduce the attack severity. We first check if this intuition holds true and then name incentives for nodes to form networks with less central nodes.

We generate a Newman-Watts-Strogatz (NWS) network [36], a connected small-world graph where all nodes have a very similar degree. The parameters for the NWS algorithm are p, the probability of a node adding a connection to a node that is not one of its neighbors, and k, the number of close neighbors to connect to. We choose a value of 0.01 for p and 20 for k. For assigning initial balances, we used the algorithm from Section A, with the addendum that if a channel had a balance below some minimum value, then we increased it to that minimum value to avoid having too many channels with no capacities. Channels of capacity 0 are unlikely due to the fact that opening a channel costs a fee.

As expected, the effect of our attacks are considerably lower in a small-world network with attackers selected by highest betweenness centrality. For the griefing attack, the attack has almost no impact for both SpeedyMurmurs and Ford-Fulkerson, as can be seen in Figure 8. When dropping, there is necessarily an effect as at least the dropped payments fail. However, the fraction of payments routed via the nodes with the highest betweenness centrality are

2. https://1ml.com/statistics



(a) SpeedyMurmurs (b) Ford-Fulkerson

Figure 9: Mitigation against dropping; attackers selected by betweenness centrality.

tiny in number in comparison to the scale-free network. Hence, the effect of dropping only becomes visible when a higher number of nodes is corrupted, as can be seen in Figure 9 for 5%.

Thus, incentivizing a more homogeneous topology is a suitable mitigation. A simple incentive is to warn participants not to form channels with strangers that are willing to pay the complete fees as such behavior often precedes an attack. Indeed, there are blockchain congestion attacks that can lead to monetary loss due to delayed disputes that have a similar attack setup [39], further emphasizing the need for preventing an attacker from establishing channels to random nodes.

6. Related Work

We refer to an in-depth survey for a detailed review of payment channel networks [2] and focus on the work related to attacks and defense mechanisms.

Payment Griefing and Channel Exhaustion. Rohrer *et al.* introduce various attacks on Lightning, namely denial-of-service, channel exhaustion, payment griefing and node isolation, as well as combinations of these [7]. Their attacks leverage the same broad ideas as ours but focus on fundamentally different routing algorithms, datasets, and threat model. They focus on the attacker initiating payments rather than affecting payments which they are on the path of. Our attacker model is more realistic as well. Rohrer *et al.*'s *node isolation* assumes that a node will not take action if its channels are nearing depletion. In fact, a node can rebalance its channels using either the blockchain or a circular payment. Perez *et al.* design and analyze a more effective payment griefing attack [8]. However, the attack is not applicable for local routing algorithms.

The effects of payment griefing can be mitigated by replacing the LN's collateral-locking protocol, Hashed-Timelock Contracts (HTLC). Lightning's current protocol might lock a payment for time $\mathcal{O}(l\Delta)$ if the path length is l and an upper bound for initiating a dispute is Δ. Several approaches could reduce this maximal timeout [29], [40], [41]. As all approaches still require timeouts in the order of minutes, our griefing by delaying attack remains applicable.

Dropping and congestion. Dropping attacks have been evaluated in the context of Lightning, finding that Lightning's hub and spoke topology as well as the predictable routing algorithm results in a high rate of failed payments if attackers are integrated into the network strategically [42]. Protections against dropping are randomization [42] and redundancy [18].

An alternative attack is network congestion [10]. In addition to the above attacks on Lightning, there are attacks on its privacy, indicating that channel balances and payment relations can be revealed [43], [44], [45]. Suddenly closing many channels can lead to congestion on the underlying blockchain and hence loss of funds due to disputes not being raised [39].

7. Conclusion

We examined the effectiveness of attacks against payment channel networks that use local routing to complete transactions. The performance of such routing algorithms degrades gracefully when the number of randomly placed attackers increases. However, they are very vulnerable if the attacker controls nodes of a high centrality and hence controls the majority of paths. We propose to incentivize payment channel networks with homogeneous node degrees.

Acknowledgments

The authors would like to thank Matthew Jagielski for his help designing the balance allocation algorithm, and Jaison Titus for many insightful discussions.

References

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

[2] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Off the chain transactions," in *Financial Cryptography and Data Security*, 2020.

[3] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *NDSS*, 2018.

[4] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability." in *NDSS*, 2019.

[5] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *NSDI*, 2020.

[6] Bolt #7: P2p node and channel discovery. [Online]. Available: https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md

[7] E. Rohrer, J. Malliaris, and F. Tschorsch, "Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks," *arXiv preprint arXiv:1904.10253*, 2019.

[8] C. Pérez-Sola, A. Ranchal-Pedrosa, J. Herrera-Joancomartí, G. Navarro-Arribas, and J. Garcia-Alfaro, "Lockdown: Balance availability attack against lightning network channels," in *Financial Cryptography and Data Security*, 2020.

[9] Connector Risk Mitigations — Interledger. [Online]. Available: https://interledger.org/rfcs/0018-connector-risk-mitigations/draft-3.html

[10] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," in *Financial Cryptography and Data Security*, 2021.

[11] Visa fact sheet. [Online]. Available: https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf

[12] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *NDSS*, 2017.

[13] Y. Zhang and D. Yang, "Robustpay$^+$: Robust payment routing with approximation guarantee in blockchain-based payment channel networks," *IEEE/ACM Transactions on Networking*, p. 1–11, 2021.

[14] Lightning Network Statistics — 1ML - Lightning Network Search and Analysis Engine - Bitcoin mainnet. [Online]. Available: https://1ml.com/statistics

[15] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[16] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[17] "The Raiden Network," https://raiden.network, May 2019.

[18] V. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment networks," in *Financial Cryptography and Data Security*, 2020.

[19] O. Osuntokun, "Amp: Atomic multi-path payments over lightning," available at: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

[20] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, 2016.

[21] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," *arXiv preprint arXiv:1810.00037*, 2018.

[22] R. Perlman, "An algorithm for distributed computation of a spanningtree in an extended lan," *ACM SIGCOMM Computer Communication Review*, vol. 15, no. 4, pp. 44–53, 1985.

[23] M. Byrenheid, T. Strufe, and S. Roos, "Attack resistant leader election in social overlay networks by leveraging local voting," in *Proceedings of the 21st International Conference on Distributed Computing and Networking*. ACM, Jan 2020, p. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3369740.3369770

[24] O. Ersoy, S. Roos, and Z. Erkin, "How to profit from payment channels," in *Financial Cryptography and Data Security*, 2020.

[25] Z. Avarikioti, L. Heimbach, Y. Wang, and R. Wattenhofer, "Ride the lightning: The game theory of payment channels," in *Financial Cryptography and Data Security*, 2020.

[26] S. Roos, M. Beck, and T. Strufe, "Anonymous addresses for efficient and resilient routing in f2f overlays," in *IEEE INFOCOM*, 2016.

[27] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[28] E. Estrada and N. Hatano, "Communicability in complex networks," *Physical Review E*, vol. 77, no. 3, p. 036111, Mar 2008.

[29] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security*, 2019.

[30] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[31] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.

[32] S. Lin, J. Zhang, and W. Wu, "FSTR: Funds Skewness Aware Transaction Routing for Payment Channel Networks," in *DSN*, 2020.

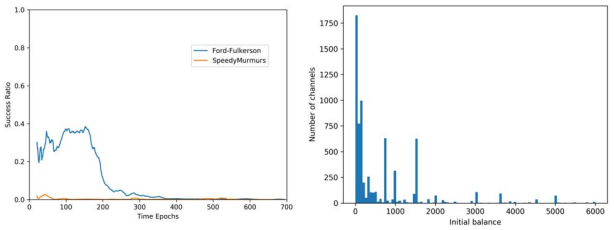
[33] Raiden Explorer. [Online]. Available: https://explorer.raiden.network

[34] "Elias Rohrer / discharged-pc-data," Jun 2020, [Online; accessed 25. Jun. 2020]. [Online]. Available: https://gitlab.tu-berlin.de/rohrer/discharged-pc-data

[35] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[36] M. E. Newman and D. J. Watts, "Renormalization group analysis of the small-world network model," *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, 1999.

[37] R. Sanders, "THE PARETO PRINCIPLE: ITS USE AND ABUSE," vol. 1, no. 2, pp. 37–40. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/eb024706/full/html



(a) Success ratio. (b) Initial balances

Figure 10: Ripple dataset. On the left, performance of Ford-Fulkerson and SpeedyMurmurs with sequential transactions and no attack, on the right initial balances.

[38] P. S. Fader, B. G. S. Hardie, and K. L. Lee, ""Counting Your Customers" the Easy Way: An Alternative to the Pareto/NBD Model," vol. 24, no. 2, pp. 275–284. [Online]. Available: http://pubsonline.informs.org/doi/10.1287/mksc.1040.0098

[39] J. Harris and A. Zohar, "Flood & loot: A systemic attack on the lightning network," in *AFT*, 2020.

[40] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *CCS*, 2019.

[41] M. Jourenko, M. Larangeira, and K. Tanaka, "Payment trees: Low collateral payments for payment channel networks," 2021.

[42] S. Tochner, A. Zohar, and S. Schmid, "Route hijacking and dos in off-chain networks," in *AFT*, 2020.

[43] J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal-Pedrosa, C. Pérez-Solà, and J. Garcia-Alfaro, "On the difficulty of hiding the balance of lightning network channels," in *AsiaCCS*, 2019.

[44] S. Tikhomirov, R. Pickhardt, A. Biryukov, and M. Nowostawski, "Probing channel balances in the lightning network," *arXiv preprint arXiv:2004.00333*, 2020.

[45] G. Kappos, H. Yousaf, A. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn, "An empirical analysis of privacy in the lightning network," in *Financial Cryptography and Data Security*, 2021.

[46] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing*. Springer, 2015, pp. 163–180.

[47] C. N. Cordi, "Simulating high-throughput cryptocurrency payment channel networks," Master's Thesis, University of Illinois, 2017.

Appendix

The effectiveness of PCN routing algorithms, and attacks against them, is highly dependent on the topology, channel balances, transactions pairs, and transaction-value distributions. We discuss limitations of existing datasets and then describe our approach.

1. Limitations of Existing Datasets

Due to the lack of available data sets for PCNs, early approaches [3], [12] relied on data from Ripple's credit network [46] for topology, capacities, and transactions. This dataset was collected by crawling the Ripple network, taking note of which accounts are funded, (*i.e.*, can make payments to other accounts) and removing inconsistencies. Early approaches not based on Ripple do not consider transaction and capacity distributions at all [20].

Unfortunately, the Ripple dataset has a very small ratio of transactions that are successful as seen in Figure 10a, where we show SpeedyMurmurs and Ford-Fulkerson with

sequential transactions. Performance for concurrent transactions would be even worse. Previous work addresses the issue by excluding transactions that were not successful for Ford-Fulkerson, leading to a very limited transaction set [3]. As shown in Figure 10b, the majority of initial channel balances are zero or very close to zero. While zero balances make sense for a credit network, where credit corresponds to trust and channel establishment does not come at a fee, it is unrealistic in the context of payment networks such as Lightning. Opening a channel requires paying the fees for one blockchain transaction, which is the same regardless of the amount locked in the channel. Thus, channels with little or no funds are unattractive as they do not provide the opportunity to make payments and hence users are unlikely to invest the fee for opening them.

After Lightning data became available, many studies relied on the Lightning topology and capacities for their evaluation [5], [7], [8], [42], [10], although some used synthetic random graphs and scale-free topologies either as their main dataset [18] or in addition to Lightning data [5].

The Lightning data set does not contain transactions. A dataset for transactions requires both sender-recipient pairs and values. Prior works have chosen sender-recipient pairs uniformly at random [18], [42], [7]. Only the evaluation of the Spider routing algorithm uses an exponential distribution [5], indicating that few senders and recipients are very active whereas the majority of nodes participate only occasionally. Transaction values have been modeled on real-world data unrelated PCNs, *e.g.*, Ripple transactions in [7], [18] and credit card transactions in [5], [47]. Using data from the Raiden Network [17] is a non-starter as it only has 37 unique accounts [33].

In summary, there are no real-world datasets available for payment channel networks capturing all the information about transactions and with realistic initial balance sets. As a result, it has become common in the literature to use synthetic graphs in lieu of the smaller LN graphs in addition to the synthetic transactions [18], [5].

2. Our Approach

A payment channel system is defined by the network graph, G, created by the channels between participants in the system, the transactions performed between these participants, T, the set of balances available on each channel, B, and the routing algorithm, R, used to find payment paths; we write $PC =< G, T, B, R >$. B is initialized with a set of initial balances B_0. Our high-level approach to generate datasets is as follows:

(1) Channel network: We first generate the channel network, modeled as a graph $G =< N, C, B >$, where N is the set of participants, C the set of channels, and B is the set of balances on each channel. Note that B is undefined at this point, we will assign an initial balance set B_0 later. We use scale-free graphs as they were shown to be representative for PCNs [7] and we write $G = SF(n, c)$, where SF is the scale-free graph generation algorithm, n is the total number of participants and c is a connectivity parameter that models how many channels a party forms when added to the graph.

(2) Transaction set: We then select a transaction set. A transaction is defined by a pair of sender and recipient and the value of the transaction. We separate the selection of the transaction pair from the selection of the value of the transactions and write $T = GT(N, nt, v_{fix}, D_v, D_n)$, where GT is our procedure of generating a transaction set and $T = \{t_i, t_i =< s_i, r_i, v_i >\}$ is the resulting transaction set with s_i and r_i as the sender and recipient, respectively, for transaction t_i of value v_i. The parameters of GT are the set of participants, N, the number of transactions, nt, sampled from distribution D_v, v_{fix} is an additional parameter determining D_v (*e.g.*, minimum, maximum, average, etc.), and D_n is the distribution used to sample pairs of parties.

(3) Initial balance set: Finally, we generate the set of initial balances for each channel. Our approach starts with a balance set of channels with capacity 0, a given set of transactions, and a percentage, tc, of transactions for which we want to have the guarantee that they can be successfully completed. We iterate over the set of transactions, and, with probability tc, execute the routing algorithm R to find one or several channels in the graph between sender and recipient. For each channel along the paths returned by R, we add the transaction value v to the balance. In this manner, we know that there is a possibility for the transaction to be successful. Formally, we write $B_0 = CIB(T, G, R, tc, m_c, p_c)$ and $B = \{b_i, b_i = (s_i, r_i) \mapsto v_i\}$, where CIB is our algorithm to compute the initial balance set, T is transaction set, G is the channel network, R is the routing algorithm and tc is the percentage of completed transactions. In addition, we might reduce the final channel balance by a factor, m_c, to generate a network with less liquidity. p_c is the probability of applying such a multiplier to a channel, *e.g.*, $p_c = 1$ indicates the multiplication is applied to all channels. We also write $P_i, G_{i+1} = R(t_i, G_i)$, where routing a transaction t_i using algorithm R on network G_i results in finding a path $P_i = \{c, c =< e_1, e_2, b >\}$, and a network with updated balances, G_{i+1}. For each channel in P_i, e_1 and e_2 are the endpoints of the channel, and b is the value being routed through that channel.

3. Generating Datasets

3.1. Generating Transaction Sets. Each transaction is a tuple of the transaction value, the sender, and the recipient, with the latter two forming the transaction pair. We treat the selection of transaction values and transaction pairs independently.

Transaction value. We consider five distributions: constant, Pareto, exponential, normal, and Poisson. For the experiments presented below, we only show results using Pareto distributions as it has been shown to most reliably represent consumer spending patterns [37].

Transaction pair. We sample the sender and recipient independently according to a distribution D_n. For this, we first randomize the order of the nodes and then map each of them to the index they have in this random order. We then sample the node based on its index according to the distribution. In particular, we focus on the Poisson distribution as existing literature has shown it to be the best model for consumer transaction rates [38].

3.2. Generating Initial Balance Set. The initial balance of a channel represents its expected use. Channels with higher balances are expected to have a higher volume of transacted funds and vice versa for lower balances. This is a result of the opportunity cost to escrowing funds in payment channels. A rational actor will want to participate in as many transactions as possible to collect fees. Also, larger channels will permit transactions that may have been too large for smaller channels to route, thus participating in more transactions. The ratio between the balance from $A \rightarrow B$ to the balance from $B \rightarrow A$ represents the ratio of transacted funds from $A \rightarrow B$ to the transacted funds from $B \rightarrow A$.

We assign balances to channels as follows. We consider a PCN, which is some graph $G_0 = (N = \{u\}, C = \{(u, v)\})$. A transaction in a PCN can occur between a sender, s, a recipient, r, and with a value, v—the full transaction t_i is written as (s_i, r_i, v_i). A routing algorithm, R, is run on a graph, G, and with a transaction t_i; it returns a route, P_i, which is the set of all modified channels, and also G_{i+1}, the graph after all channels are modified. We write this as $P_i, G_{i+1} = R(t_i, G_i)$. The routing algorithm, R, may return multiple paths and may also be randomized. If a route with sufficient balances cannot be found, *i.e.*, there is not a high enough funds on the channels connecting s_i to r_i, R returns $\perp$ and G_i (G_i is not modified). The modified graph, G_{i+1}, is equivalent to, for each hop $(a, b, m) \in P$, either subtracting m from the channel (a, b)'s funds in G or adding m to the channel (b, a)'s funds.

Reducing Graph Weights. Generating the perfect channel balances is useful for testing the system, but to simulate certain concurrency scenarios, we need some of the channels to have insufficient credit to complete *all* of the transactions in T. To achieve this, we simply take all edge weights in the graph, and scale them by some multiplier, *i.e.*, $E' = \{(u, v, k \cdot w)\}_{(u,v,w)\in E}$, where k is a constant scale factor.

We also experimented with the option of only applying the factor k probabilistically, *i.e.*, applying it only with probability p_c. A complementary approach for reducing the capacity of the network is to select transactions for the generation of the initial balance generation randomly, with probability tc.

3.3. Generated Datasets. We used the implementation of the Barabási-Albert algorithm [35] for the scale-free graph generation from the Python module networkx [30]. The datasets were generated according to the methodologies discussed above and are shown in Table 2. In deciding the number of nodes to model, we settled on a value of 10k as it was larger size than the current number of active LN nodes but still small enough that it would reflect the network in the near future. While we experimented with all datasets in the table, we present our results for the scale-free dataset.

Name	Topology	Transaction Set	Initial Balance Set
0	SF(100k, 5)	GT(100k, 1m, 1, Pareto, Pareto)	CIB(T, G, shortest path, 100%, 1, 1)
6	SF(100k, 5)	GT(100k, 1m, 1, Pareto, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 1)
7	SF(100k, 2)	GT(100k, 1m, 1, Pareto, Pareto)	CIB(T, G, shortest path, 100%)
8	SF(10k, 2)	GT(10k, 1m, 1, Pareto, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 1)
10	SF(10k, 2)	GT(10k, 1m, 1, Pareto, Pareto)	CIB(T, G, shortest path, 100%, 1, 1)
12	SF(10k, 2)	GT(10k, 1m, 1, Pareto, Constant)	CIB(T, G, shortest path, 100%, 1, 1)
13	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Poisson)	CIB(T, G, shortest path, 100%, 1, 1)
14	SF(10k, 2)	GT(10k, 1m, 1, Exp, Exp)	CIB(T, G, shortest path, 100%, 1, 1)
15	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Poisson)	CIB(T, G, shortest path, 100%, 0.5, 0.5)
16	SF(10k, 2)	GT(10k, 1m, 1, Normal, Normal)	CIB(T, G, shortest path, 100%, 1, 1)
17	SF(10k, 2)	GT(10k, 1m, 1, Normal, Normal)	CIB(T, G, shortest path, 100%, 0.5, 0.5)
18	SF(10k, 2)	GT(10k, 1m, 1, Normal, Normal)	CIB(T, G, shortest path, 100%, 0.5, 0.5)
19	SF(10k, 2)	GT(10k, 1m, 1, Normal, Normal)	CIB(T, G, shortest path, 80%, 1, 1)
20	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 1, 1)
21	SF(10k, 2)	GT(10k, 1m, 30, Poisson, Normal)	CIB(T, G, shortest path, 100%, 1, 1)
22	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Constant)	CIB(T, G, shortest path, 100%, 1, 1)
23	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 0.5)
24	SF(10k, 2)	GT(10k, 1m, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 1)
Scale-Free	SF(10k, 2)	GT(10k, 100k, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 0.5)
Small-World	NWS(10k, 20, 0.01)	GT(10k, 100k, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 0.5, 1, 100)
Lightning	Measured	GT(2337, 100k, 1, Poisson, Pareto)	CIB(T, G, shortest path, 100%, 1, 1, 100)

TABLE 2: Description of datasets we generated and experimented with. Unless otherwise stated, our simulations use the scale-free dataset. The datasets analyzed in this paper are highlighted.