ProfitPilot: Enabling Rebalancing in Payment Channel Networks through Profitable Cycle Creation

Gustavo F. Camilo[1], Gabriel Antonio F. Rebello[1,2,3], Lucas Airam C. de Souza[1], Miguel Elias M. Campista[1], and Luís Henrique M. K. Costa[1]

[1]Universidade Federal do Rio de Janeiro - GTA/Poli/COPPE/UFRJ
[2]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[3]Instituto de Pesquisas Eldorado, Brazil

***Abstract*—Payment Channel Networks (PCNs) have successfully replaced slow global consensus mechanisms with local cryptographic agreements between nodes. As PCN payments heavily depend on network topology for payment routing, strategic node positioning is critical to building cost-effective channels for users and enhancing network robustness against topological attacks. Nevertheless, existing node attachment strategies in the Lightning Network (LN), the most popular PCN, ignore crucial topology issues, such as network centralization and the scarcity of cycles for cheap off-chain rebalancing. In this paper, we first investigate the current state of the LN topology and show that the availability of topology cycles is highly unequal in the network, which exposes the network to several vulnerabilities. Then, we design ProfitPilot, a node positioning strategy that encourages cycle creation in PCNs to reverse the trend in centralization and enable cheap off-chain rebalancing. We compare our proposed algorithm with heuristics available in the Lightning Network and verify that even by focusing on creating cycles, ProfitPilot successfully increases the user's probability of collecting fees by over $2\times$ while reducing average paying fees. Furthermore, out of all the evaluated heuristics, ProfitPilot presents the fastest increase in network transitivity and mitigates the impact of targeted topological attacks by over 17% compared with the regular Lightning Network operation.**

***Index Terms*—blockchain, payment channel networks, Lightning Network.**

I. Introduction

Public cryptocurrencies still suffer from low scalability despite their undeniable success. While traditional payment methods, such as credit cards, process over 24,000 transactions per second [1], the two largest cryptocurrencies, Bitcoin and Ethereum, achieve a throughput of only 7 and 15 transactions per second, respectively [2]. This scalability problem stems primarily from the requirement imposed by public blockchains of processing every transaction through a slow global consensus mechanism [3], [4], [5], [2], [6].

Payment channels are one of the main solutions to the scalability problem of public cryptocurrencies [5], [7], [8], [9]. In a payment channel, two users transfer funds into a 2-of-2 multi-signature address in the blockchain so the funds can only be spent if both agree to sign a new transaction. Then, instead of publishing new transactions, they establish an *off-chain communication channel* and send transactions that rebalance the channel funds directly to each other. Multiple payment channels can be connected to build a payment channel network (PCN), which allows users to route payments through intermediaries under the condition of paying a small routing fee. Hashed Timelock Contracts (HTLC) guarantee the trustlessness of such process by enforcing that intermediaries only receive funds from the previous hop after transferring a nearly-equal amount of funds (fees are discounted) to the next hop. PCNs have been implemented in the two most popular cryptocurrencies, namely Bitcoin's Lightning Network (LN) [5] and Ethereum's Raiden [10], and have effectively improved transaction throughput and latency in public blockchain systems. In fact, PCNs are considered a motivation for the adoption of Bitcoin as an official payment method in El Salvador [11].

Routing payments in PCNs is challenging and radically differs from routing datagrams in computer networks. In PCNs, when a payment is routed on a channel, the channel's capacity to forward future payments in that direction is reduced. Thus, when payments concentrate on one direction, the channel can become *depleted* and unable to route payments in the direction of these payments. Figure 1 shows an example where the channel from n_1 to n_2 is depleted (the capacity is zero). The user, then, has two options to reactivate (rebalance) the depleted payment channel. The first approach is closing and reopening the channel or acquiring liquidity through a liquidity provider [12]. As these approaches rely on blockchain transactions, it is slow and costly. Furthermore, by closing the channel for reopening an active one, the user interrupts the channel's lifespan, halting payments and hindering fee collection. Besides, while the reopening transactions or liquidity purchases are unconfirmed in the blockchain, the node is unable to issue payments or collect fees in the depleted channel.

The second and most efficient approach to rebalance depleted channels is to issue a self-payment through a circular route in the network [13], [14], [15], [16], [17]. Figure 1 illustrates the process of off-chain rebalancing. The key idea is to move funds from low-demand channels to high-demand ones, keeping the high-demand channels alive for a longer period of time.

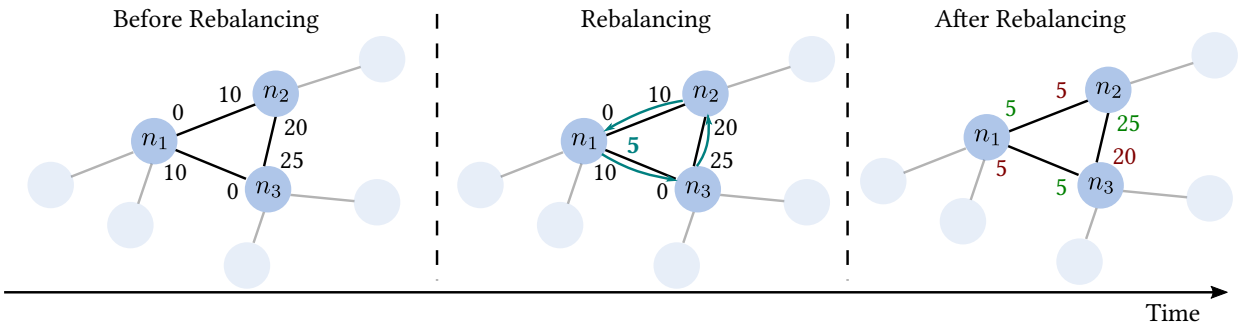


Fig. 1: Channel rebalancing through self-payment in a circular route. Node n_1 finds a cycle and issues a self-payment of value 5, rebalancing its channel with n_2.

Although this approach offers a low-cost alternative to closing and reopening the payment channel, it depends on the existence and availability of cycles in the network topology. If cycles are unavailable, users have no other option than to resort to the blockchain. In this paper, we verify that 36% of nodes in the Lightning Network, the most popular PCN currently, do not participate in cycles. Furthermore, even nodes that do participate in cycles usually have to pay high rebalancing fees, which restricts user adoption to off-chain rebalancing (see Section II). A low number of cycles in the network topology means fewer alternative paths and a highly centralized network [18]. This centralization trend exposes the network to several topological attacks [19], [20]. These attacks can partition the network, directly impacting payment success rates and payment reachability [20]. As a consequence, node attachment strategies are pivotal not only to reverse the centralization trend but also to enable cheap off-chain rebalancing operations [20], [21].

Even though network cycles avoid channel depletion, the current literature shows a gap in attachment strategies that benefit both the network and users. Thus far, most work on PCN attachment strategies focuses solely on creating profitable channels for liquidity service providers (LSP) and relay nodes [21], [22], [23]. Furthermore, Lightning Network *autopilots*, tools that automate channel creation for users, usually focus on connecting to high-degree or high-capacity nodes [23], [24]. These attachment patterns reinforce the current trend in network centralization and offer little incentive for mass user adoption.

In this paper, we present ProfitPilot, a node attachment strategy for PCNs that prioritizes cycle creation while aiming for cheap rebalance and improving PCN robustness. We first analyze the availability of cycles in the most popular PCN implementation, Bitcoin's Lightning Network (LN) [5]. We show that, despite off-chain rebalancing being the most efficient approach for restoring channels [13], [15], [16], LN's current topology restricts such operations to a limited number of nodes. Moreover, we demonstrate the impact of cycle scarcity in network centralization and how it reduces network robustness. We simulate attacks on an LN snapshot and show their long-term consequences. Next, we introduce ProfitPilot, our attachment strategy that focuses on creating cycles. In particular, the core of ProfitPilot is to produce 3-node cycles to create low-cost routes and enhance network robustness. As creating cycles requires users to open more than one channel, which is costly, ProfitPilot strategically creates profitable channels to encourage user adoption. Unlike previous works, ProfitPilot also considers regular users who might want to both collect fees and make payments in the PCN.

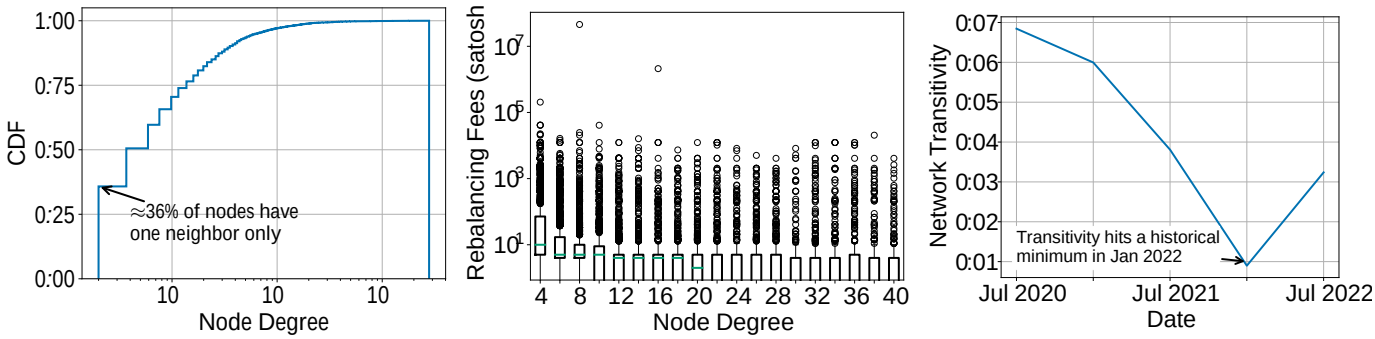
We implement ProfitPilot and compare our proposal to state-of-the-art attachment heuristics available in the Lightning Network. We evaluate our algorithm on the Lightning Network and two synthetic topologies. ProfitPilot achieves 2× higher fee collection in all evaluated topologies when compared to other heuristics and efficiently generates cheaper routes. Moreover, ProfitPilot rapidly increases network transitivity.

This paper is organized as follows. Section II presents the motivation of our work based on the LN topology of July 2022. Section III introduces ProfitPilot, our proposed node attachment algorithm. Section IV evaluates the performance of ProfitPilot. Section V reviews related work. Finally, Section VI concludes the paper and identifies future work directions.

II. Topological Analysis of the Lightning Network

We analyze the Lightning Network [5], Bitcoin's most popular PCN, to verify the availability of cycles in the network. Cycles are particularly important in PCNs to allow cheap off-chain channel rebalancing [13], [16], [15]. We use real-world data collected from the Lightning Network from July 2020 to July 2022 to build the network graph [25]. The data consists of raw node and channel announcement messages spread through the network using gossip messages to disseminate topology updates. The messages contain information that allows nodes to build the network topology, such as the fees charged on the channel, who participates on the channel, and the channel identifiers. The fields of announcement messages are detailed in the Basis of the Lightning Technology (BOLTs) [26], which describes LN's implementation standards. We reconstruct the network topology using the collected messages and the NetworkX Python library to generate bidirectional graphs. For this analysis, we only consider the largest connected component of the graph. We also fill in missing attributes on the dataset1, such as channel capacities, using publicly available information on LN explorers [27].

1The results of this attribute retrieval are available at https://gta.ufrj.br/~gabriel/files/ln-graphs.tar.gz.

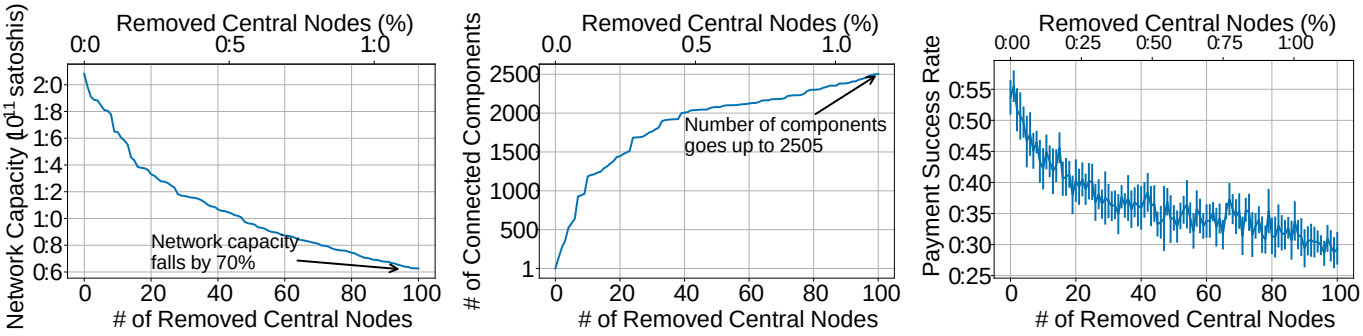


(a) Node degree cumulative distribution in the Lightning Network in July 2022.

(b) Rebalancing fees as a function of the node degree in the Lightning Network.

(c) Network transitivity of the Lightning Network from July 2020 to July 2022.

Fig. 2: Network metrics of the Lightning Network in July 2022. We verify the scarcity of cycles in the network for a high number of nodes, hindering several rebalancing operations. Furthermore, we verify how rebalancing fees are highly unequal to nodes that do participate in cycles.



(a) Network capacity as high-degree nodes are removed.

(b) Network components as high-degree nodes are removed.

(c) Payment success rate as high-degree nodes are removed.

Fig. 3: Impact of topological attacks in the Lightning Network as of July 2022. We simulate central nodes being impaired in a targeted attack and verify the effect on network capacity and partitioning.

Figure 2a shows the degree distribution in the Lightning Network in July 2022. While the majority of nodes in LN have a low degree, a small number of nodes concentrate the network connectivity. Almost 36% of nodes in the Lightning Network have only one neighbor. Leaf nodes are unable to rebalance their channels through off-chain methods and have to rely on slow and expensive blockchain payments to keep their channels alive once they get depleted. Although these nodes may create cycles, they would still have to endure high fees in the blockchain, and current automatic channel creation heuristics present no financial benefits to the user. Even worse, leaf nodes typically have low-capacity channels, in contrast with high-capacity nodes that can easily perform off-chain rebalancing. This contrast deepens the network inequality, which recent works claim is extremely high [18], [28].

We also verify the availability of cheap rebalancing routes among nodes that have more than one neighbor. We run Dijkstra's Shortest Path First (SPF) algorithm using fees as edge weights and considering the average payment value of a dataset containing Ripple's credit network transactions[2] (see Section III for a formal definition of our network model) [29]. Ripple's transaction dataset contains over 2 million transactions collected by crawling Ripple's credit network and gathering information from its creation, in January 2013, to November 2016. The dataset contains information on transaction identifiers, sender-receiver pairs, the timestamp of when the transaction was issued, and the value of the transaction in dollars. We find that even when using off-chain rebalancing, the distribution of rebalancing fees is highly unequal in the network. Figure 2b shows that, while high-degree nodes usually find cheap routes to rebalance their channels, low-degree nodes have fewer options and end up paying more for each rebalancing operation.

The scarcity of short network cycles can also be observed in the network transitivity. Network transitivity measures the ratio between the number of triangles and the number of connected triples of nodes. Transitivity varies between 0 and 1 and high transitivity implies a large number of triangles, e.g., a dense graph, while low transitivity indicates a sparse network. Figure 2c shows the transitivity of the Lightning Network from July 2020 to July 2022. Despite the small increase from January to July 2022, the LN's transitivity tends to decrease [18], [28]. The low transitivity of LN demonstrates the low number of cycles, particularly 3-node cycles, and is a consequence of the

[2]The Lightning Network keeps payment values private by design and thus no payment dataset is available. We assume Ripple's credit network operates similarly to PCNs and, therefore, has similar payment ranges.

centralization trend of the network [18].

Creating cycles in the network is important not only to enable rebalancing operations but also to improve robustness against targeted topology attacks [19], [20]. Several works point out that, despite being born a decentralized network, LN is becoming more centralized by concentrating connectivity and capacity in a small number of nodes [18], [28]. This centralization trend increases vulnerability to attacks targeting the highest-degree nodes.

Impact of Topological Attacks. Figure 3a shows that impairing only 44 of the highest degree nodes ($\approx$ 0.5% of nodes in the network) reduces the network capacity by half. Furthermore, extending the attack to the 100 highest-degree nodes ($\approx$ 1.2% of the network) leads to a decrease of 70% in network capacity. Reducing the network capacity directly affects the payment success rate, due to less available channels [30], [8]. Figure 3b also demonstrates the impact of targeted attacks on the number of network components. We verify that attacking 100 of the highest degree nodes increases the number of components from one to over 2,500. Indeed, removing only the highest-degree node breaks the network into 140 components. This network partitioning pattern has drastic effects on PCNs, as it makes several payments infeasible.

Figure 3c shows how the network partition affects payment success rate. We implement a Lightning Network simulator[3] and simulate a PCN using an LN snapshot from July 2022 as the network topology and a Ripple transaction dataset as workload [31], [29]. The success rate falls from over 55% to less than 30% after blocking the 100 highest-degree nodes, resulting in multiple failed payments. Users must complete these failed payments in the blockchain, incurring high fees and long confirmation time. It is important to note that these attacks are not only theoretical. In 2018, LN was the target of a DDoS attack, which affected 20% of its nodes [32], [18].

III. Building ProfitPilot

We present ProfitPilot, an algorithm for connecting new nodes to PCNs. ProfitPilot focuses on the creation of 3-node cycles, favoring rebalancing operations and reducing the centralization tendency mentioned in Section II. We choose to focus on 3-node cycles for two main reasons. First, smaller cycles provide higher tolerance against fee fluctuations over time. Instead of relying on $n-1$ nodes not changing their fees in a n-length cycle, the user only relies on 2 nodes, which are her neighbors. We argue that it is easier for the user to keep control or exert influence over channels with their own neighbors than over other nodes she does not share channels with. Furthermore, by focusing on creating 3-node cycles, we improve the overall network robustness that would otherwise be negatively affected by only considering connections to central nodes as in previous work [23]. ProfitPilot builds triangles on top of the original network graph, providing alternative paths and mitigating the effect of targeted attacks. To make the cycle creation economically viable, our strategy offers financial incentives to encourage user adoption. Finally, with mass adoption, 3-node cycles merge into forming cycles with higher lengths, increasing possible rebalancing paths [14]. Our focus is on establishing channels that maximize the likelihood of routing payments through them and minimize the average number of hops to other participants. This optimization reduces transaction fees and enhances the overall efficiency of the network.

[3]Available at https://github.com/gFrancoCamilo/ln-looprebalance.

A. Assumptions and Network Model

Publicly-available topology. ProfitPilot assumes that the topology of the PCN is publicly available to PCN participants before they create any channel. In particular, we assume that the available information includes public nodes, channels, their capacity, and their fee policy. The two most popular PCNs today, Lightning and Raiden [5], [10], both allow users to download the network topology before establishing a channel in the network. In fact, Lightning and Raiden network explorers store and display information about nodes and channels on public websites [27]. We also assume that substantial changes in the network topology while the algorithm is running are rare. Our assumption is based on the fact that every topology update has to go through the blockchain, which is time-consuming [21]. Thus, only minor changes occur during the execution of the algorithm.

Freedom to create channels. We assume that a new node coming to the PCN can create channels with any other node publicly announced in the network. The newcomer, however, funds all costs related to channel creation. We assume that the other party acts rationally and accepts channel creation as she does not have to fund the channel and may gain fees from payment routing. This assumption is also made by previous works in the literature [21], [22].

Source-routed payments. We strategically position a node to receive routing fees and pay, on average, lower fees than other nodes. Thus, we need to be aware of how payments are routed in the network. ProfitPilot assumes that nodes adopt the default routing strategy in the Lightning Network, i.e. source routing with paths computed by Dijkstra's SPF algorithm. Although several proposals to modify routing algorithms in PCNs have been proposed [8], [29], [30], [33], [34], the current implementation of the Lightning Network still uses Dijkstra's SPF algorithm with fees as the distance metric. Furthermore, following previous works [21], [35], we assume a uniform distribution of sender-receiver pairs when placing the node in the network.

Network model. We model the PCN as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of channels in the network. Each bidirectional channel, represented by two directed edges, holds a set of attributes with information about the routing fees, channel capacity, and a channel identifier. In particular, an edge $e_{ij} \in E$ stores two attributes regarding fees: a base fee f^B and a proportional fee $f^P(p_v)$. While the base fee is fixed for every transaction that is forwarded through the channel, the proportional fee depends on the payment value p_v being routed. Thus, the total amount of fees accounted for in a channel e_{ij} that transports a payment of value p_v is given by $f^T_{ij}(p_v) = f^B_{ij} + f^P_{ij}(p_v)$.

Similar to [19], we account for the weight of the fees by generating a payment graph $G_P = (V, E, W)$ from the original

network G, where W is the set of edge weights. Basically, the payment graph has the same nodes and edges as the original network graph, but the edges are weighted by the total fees. Thus, given a channel e_{ij}, the weight w_{ij} is defined as $f_{ij}^{T}(p_v)$ for a fixed value of p_v. Users may define different fees in both directions of the payment channel, i.e., given an edge $e_{ij} \in E$, it is possible that $w_{ij} < w_{ji}$. As total fees depend on payment values, which are kept private on the Lightning Network [5], we use the average payment value of a transaction dataset collected from the real-world Ripple credit network [29].

B. Requirements

We identify the following requirements for our attachment strategy design:

1) **Profitability:** As channel creation incurs expenses and cycle creation requires establishing more than one channel, we focus on the creation of profitable channels, which may compensate for the opening costs during its lifespan.
2) **Economy:** As the user does not know beforehand all the potential payment destinations, we create channels as close as possible to other nodes in the network so the new user pays fewer fees on average.
3) **Cycle-enabled:** As we focus on enabling cheap off-chain rebalancing operations, newly-created channels must have at least one cycle.

The above requirements aim to encourage user adoption through incentives. From the network perspective, we claim that building cycles creates redundancies, which attenuate the current trend of network centralization [18], [20].

C. Greedy Algorithm for Node Attachment

We propose a greedy algorithm that recommends channel connections to ingress nodes in the network. The goal of the algorithm is to create 3-node cycles while maximizing a user's probability of forwarding a payment and reducing the average amount paid in fees when issuing a transaction. We achieve both of these features through centrality metrics. First, as current PCNs use a simple Dijkstra's SPF algorithm to select the cheapest payment route, we can fulfill our goal of increasing the node's probability of forwarding a payment by creating channels that belong to as many shortest paths as possible in the PCN. In other words, for each channel created the user is interested in having more payments routed through her node. To that end, we employ the *node betweenness centrality*, which measures the fraction of shortest paths crossing the node, as one of the optimization objectives (Appendix A gives the mathematical formulation). Increasing a node's probability of routing is equivalent to increasing its betweenness centrality.

Second, we employ the *closeness centrality* to measure how far from every other node the new user is on the network. ProfitPilot aims to minimize the overall distance to everyone else in the network, as it assumes that the new user does not know beforehand to whom she is going to send payments. Closeness centrality suits well in our proposal as it measures the overall distance of a node to the rest of the network. Thus, the higher the closeness centrality, the closer the node is to all other nodes, meaning fewer routing fees for the new node.

Both betweenness and closeness centralities depend on the fees that will be charged on the created channel. This happens because the default Dijkstra's SPF algorithm used by some PCNs, including the Lightning Network, selects the cheapest path, i.e., the one that charges fewer fees. Thus, setting low base and proportional fees results in higher betweenness and closeness centrality, whereas setting high fees yields the opposite result. As choosing channel fees is beyond the scope of our work [4], ProfitPilot considers a user that sets the channel fees as the median of the entire network. Thus, we set the base fee to 100 milisatoshis (msat) and the proportional fee to 50 satoshis (sat)[5].

We combine the node's betweenness and closeness centrality into one incentive metric (see Appendix A), R_n, described as

$$R_n = \alpha \cdot bc_n + (1 - \alpha) \cdot cc_n, \tag{1}$$

where bc_n represents the betweenness centrality and cc_n is the closeness centrality of node n. ProfitPilot offers a tuning parameter $\alpha, 0 \leq \alpha \leq 1$, that can be set by the user to control the weight given to the betweenness centrality and the closeness centrality in the incentive computation. If the new user plans to issue a small number of transactions, she can set $\alpha > 0.5$, prioritizing gains from fee collection. Conversely, if the user plans to issue several transactions and wants to pay fewer fees on average, the user may set $\alpha < 0.5$.

At each step, ProfitPilot greedily looks for the neighbor with the highest incentive metric increase. ProfitPilot's algorithm, detailed in Algorithm 1 receives the PCN topology G and the maximum number of channels k to create as input and returns the recommended neighbors for a new node. ProfitPilot simulates the creation of multiple channels offline and computes the incentive return. The algorithm iterates through the nodes of the network creating channels with each one and stores the node N_+ that offers the highest incentive R_M on each round in a list N. This procedure continues until the incentive no longer increases, meaning it has reached a local maximum, or until the list N reaches k channels. If the incentive starts to decrease, the algorithm stops, given that any selected channel will provide a smaller incentive than the current list.

ProfitPilot aims to create triangles in the network. After selecting the first node that presents the highest incentive, it is possible to create a 3-node cycle by verifying the selected node's neighbor instead of iterating through every node in the network. Thus, our algorithm executes the same procedure of looking for the highest incentive but in a reduced search space, saving execution time. Through this procedure, ProfitPilot guarantees the creation of $|\mathcal{N}| - 1$ triangles (see proof in Appendix B) that the node participates in, where $\mathcal{N}$ is the set of the node's suggested neighbors, including already established ones, and $|\mathcal{N}|$ is the size of the set.

Although ProfitPilot only considers new nodes joining the network, it can easily be extended to nodes that already

[4]Readers interested in fee selection may refer to Ersoy et al. [21].

[5]A satoshi is the atomic unit of Bitcoin and is equivalent to 10^{-8} BTC. Nevertheless, operations in milisatoashis (1 satoshi = 1,000 milisatoshis) are common in the Lightning Network. The milisatoshis are usually rounded down when closing the channel.

Algorithm 1: ProfitPilot's main algorithm.

Input : $G = (V, E) \rightarrow$ payment channel network as a directed graph;
$n \rightarrow$ node entering the network;
$\alpha \rightarrow$ parameter that indicates whether the user prioritizes collecting fees or paying low fees;
$k \rightarrow$ maximum number of channels the user wants to create.
Output : $\mathcal{N} \rightarrow$ set of neighbors recommended by the algorithm.

```
Initialize empty set of neighbors N ← ∅;
while |N| < k do
    Initialize maximum reward R_M ← 0;
    Initialize selected node N_+ ← None;
    foreach n_i ∈ V do
        Simulate channel opening (n, n_i);
        Compute incentive
          R_n ← ComputeIncentive(G, n, α);
        Simulate channel closure (n, n_i);
        if R_n ≥ R_M then
            if |N| > 0 then
                if n_i is neighbor of n_s ∈ N then
                    R_M ← R_n;
                    N_+ ← n_i;
                end
            end
            else
                R_M ← R_n;
                N_+ ← n_i;
            end
        end
    end
    Compute current incentive
      C_R ← ComputeIncentive(G, n, α);
    if R_M ≤ C_R then
        Break out of the loop;
    end
    N ← N ∪ N_+;
    Add channel (n, N_+) to network G;
end
return N
```

participate in the PCN. In that case, instead of iterating through every node in the network, ProfitPilot attempts to form a cycle in a procedure similar to closing a triangle or a triadic closure [36]. ProfitPilot connects to the node that provides the highest increase in incentive among the neighbors of the user's already established neighbors.

IV. Implementation and Evaluation

We implement ProfitPilot and evaluate it using real data collected from the LN topology. We use Python v3.8 and the NetworkX library for graph analysis[6]. We build the network graph from node and channel announcement messages, used by nodes to announce their channel on the network publicly. The messages were collected using the `c-lightning` implementation and comprise the period from January 2020 to July 2022 [25]. Similar to the literature [8], [14], we snowball sample [37] the complete Lightning Network topology starting from the highest degree node, going from a topology with $8,676$ nodes and $80,220$ edges to a network with 512 nodes and $3,212$ edges. Our analysis focuses on evaluating ProfitPilot in the Lightning Network. Nonetheless, ProfitPilot is PCN-agnostic and can be easily adapted to other PCNs.

Besides LN, we evaluate ProfitPilot using two synthetic topologies: a scale-free Barabasi-Albert graph [38] and a small-world Watts-Stogratz [39] graph, both with 512 nodes. We chose these two topologies because the Lightning Network behaves both as a scale-free and as a small-world topology [18], [20], [28]. We sample node and channel attributes, such as channel capacity, base fee, and proportional fee, from the real Lightning Network and assign them to nodes and edges in the synthetic topology. During this assignment, node correspondence was kept, meaning that the central nodes in the synthetic topology receive their attribute from the central nodes in the Lightning Network snapshot. In particular, the 10% highest degree nodes in the synthetic topology receive attributes from the 10% highest degree nodes in the original LN topology. Unless stated otherwise, we repeat our experiments 10 times to account for statistical variations and error bars represent 95% confidence intervals. The results of ProfitPilot on LN, however, do not present error bars given that the LN topology is fixed and the greedy algorithm is deterministic.

Attachment algorithms. We compare our solution to the following attachment heuristics used by a publicly available LN autopilot [23]:

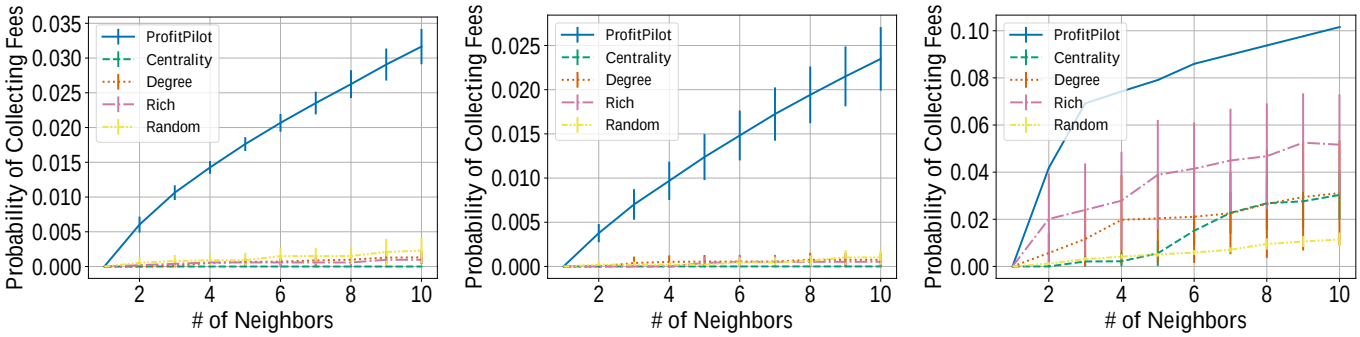
1) *Centrality:* This heuristic selects nodes as potential neighbors from a distribution proportional to the betweenness centrality of nodes.
2) *Rich:* This heuristic selects nodes from a distribution proportional to the capacity of nodes in the network.
3) *Random:* This heuristic follows the Erdos-Renyi [40] model by drawing nodes from a uniform distribution.
4) *Degree:* This heuristic draws nodes from a distribution proportional to the degree of nodes.

The above-listed heuristics were chosen due to their presence in the most popular Lightning Network implementations. The first three heuristics are available in `lib_autopilot`, the default autopilot for the `c-lightning` implementation of the Lightning Network [23].

A. Results

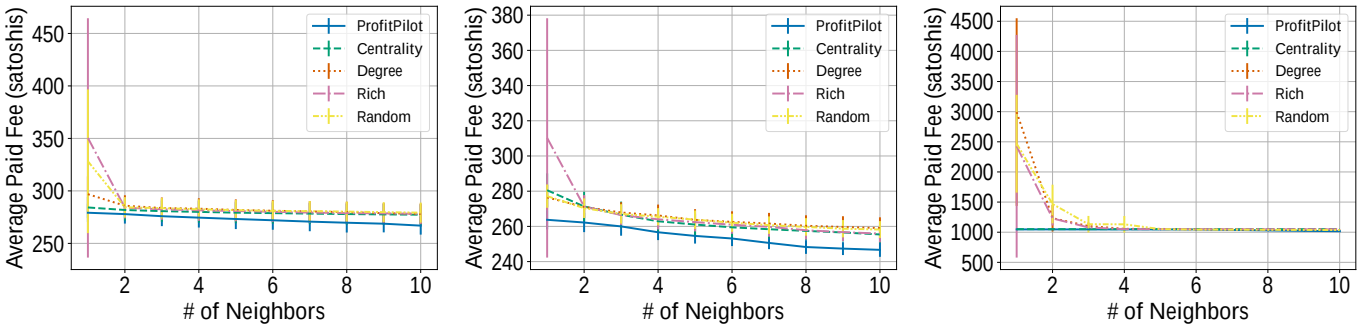
We divide our performance evaluation into two parts. First, we evaluate our proposal from the new node's point of view. We verify if our proposed solution achieves higher rewards than current solutions. Our evaluation compares the approach that creates cycles and the approach that doesn't create cycles to verify if there is a significant loss in incentives. Finally, we evaluate the difference in collected fees from our algorithm to the c-lightning ones. Then, we verify the impact of mass adoption from the network point of view and how it affects the network centralization.

[6]Available at: https://github.com/gFrancoCamilo/ln-looprebalance.



(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 4: Probability of forwarding a payment and, therefore, collecting fees in the evaluated topologies. ProfitPilot more than doubles the probability of collecting fees.



(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 5: Average paid fee for a transaction in the evaluated topologies. ProfitPilot creates cheaper routes when compared with state-of-the-art heuristics in all evaluated topologies.

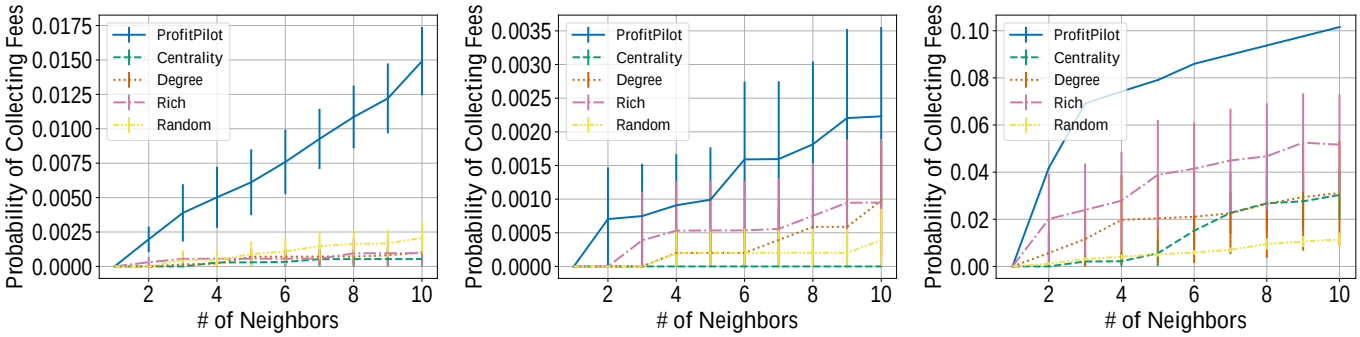
ProfitPilot without cycles. One of ProfitPilot's main benefits is allowing nodes to compensate for channel creation costs by collecting routing fees. We run an experiment that compares ProfitPilot with the currently available autopilot heuristics to quantify this profit boost. The evaluation scenario consists of a node that creates ten channels, each set to charge the network median base and proportional fees, and $\alpha = 0.5$. Figure 4 shows the probability of forwarding a payment for each heuristic in the evaluated topology. ProfitPilot increases the likelihood of forwarding a payment and thereby collecting a fee by over 5× in the Barabasi-Albert and Watts-Strogatz topologies, as shown in Figures 4a and 4b respectively, and by over 2× in the Lightning (Figure 4c). Thus, users are compensated for the cost of opening multiple channels using ProfitPilot.

ProfitPilot also creates cost-effective payment routes (Figure 5). The average paid fee is computed based on the shortest paths from the ingress to all other nodes in the network and averaging the path fees. We consider a payment value equal to the average transaction of a Ripple dataset [29] to calculate the fees. ProfitPilot presents equal or lower-cost paths compared with the other heuristics for the three topologies evaluated. Moreover, creating more channels decreases the average paying fees, notably for the Watts-Strogatz topology (Figure 5b).

Impact of cycles.

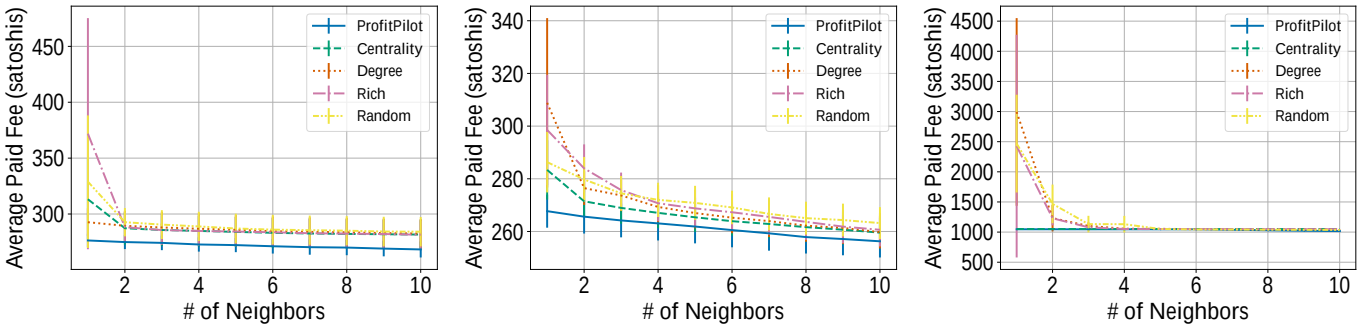
We repeat the previous experiments for the cycle creation approach. Figures 6 and 7 show the results for the average paid fees and the probability of collecting fees, respectively. Even by forcing the creation of triangles and reducing search space, ProfitPilot achieves a higher probability of collecting fees in all three evaluated topologies. In particular, ProfitPilot stands out in the Lightning Network topology, significantly improving fee collection probability. Furthermore, similarly to the approach that ignores cycle creation, ProfitPilot also offers a smaller average paid fee compared to other available heuristics from the first channel created. Thus, users are not required to create a high number of channels to receive the benefits of ProfitPilot.

We also verify the number of triangles the node participates in using ProfitPilot or the other evaluated heuristics. Triangles are valuable in PCN topologies to allow off-chain rebalancing and create redundancies. As stated in Section III-C and demonstrated in Appendix B, ProfitPilot guarantees the creation of at least $|\mathcal{N}| - 1$ triangles in all topologies, where $|\mathcal{N}|$ is the number of neighbors. Figure 8c shows that the centrality and rich heuristics generate a higher number of triangles in the Lightning Network topology. One possible reason for this difference is that central nodes, which are usually high-capacity nodes [18], are highly connected to each other, increasing the probability of creating triangles in the Centrality and Rich heuristics. Nevertheless, these heuristics are unable to ensure the creation of triangles under different topologies, as shown in the results of the Barabasi-Albert and Watts-Strogatz topologies in Figures 8a and 8b, respectively, unlike ProfitPilot.



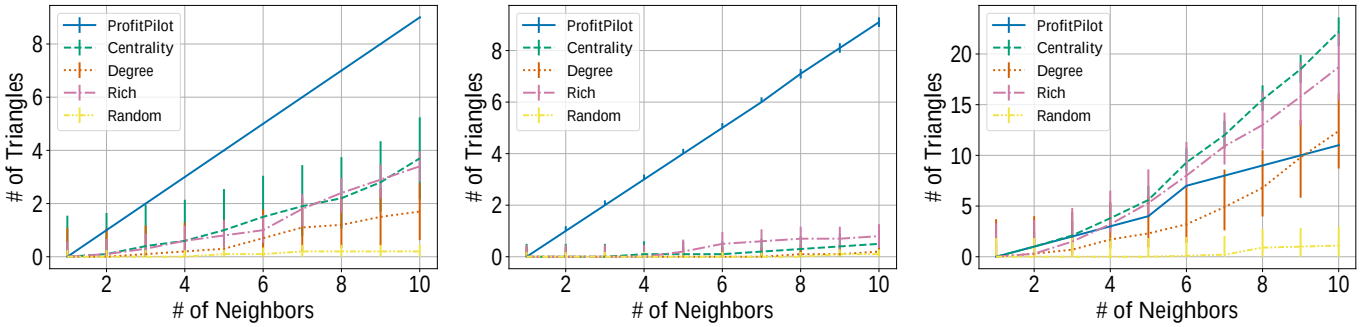
(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 6: Probability of forwarding a payment and, therefore, collecting fees in the evaluated topologies using the strategy that prioritizes the formation of 3-node cycles. Even by forcing the creation of triangles, ProfitPilot presents a higher probability of collecting fees than currently available heuristics.



(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 7: Average paid fee for a transaction in the evaluated topologies using ProfitPilot to prioritize the formation of 3-node cycles.

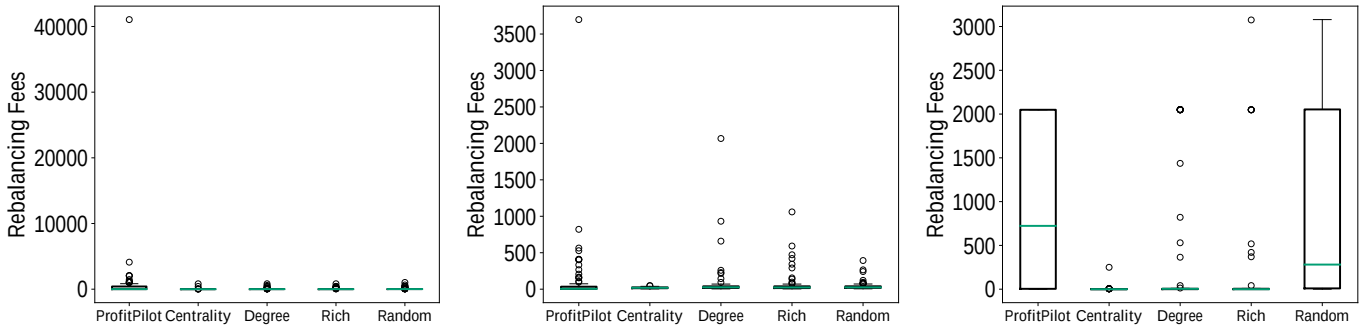


(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 8: Number of triangles created by ProfitPilot and state-of-the-art heuristics in the evaluated topologies.

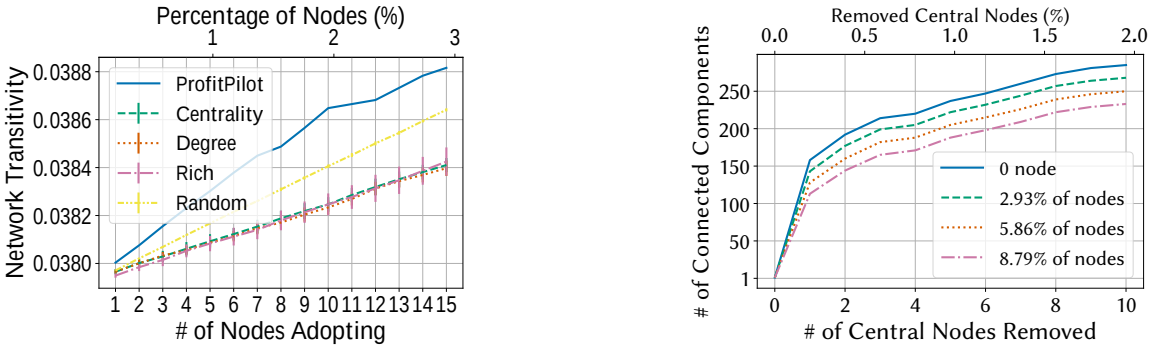
Figure 9 shows the rebalancing fees paid by the user in each topology after creating 10 channels. As ProfitPilot focuses on creating 3-node cycles, it may not produce least-cost cycles. Figure 9a shows that Centrality produces cheaper rebalancing cycles than ProfitPilot for the Lightning Network topology. Still, ProfitPilot presents equally low fees in both Barabasi-Albert and Watts-Strogatz topologies. While the centrality heuristic outperforms ProfitPilot in rebalancing fees in the Lightning Network, ProfitPilot's benefits, such as channel profitability, compensate the user for this difference. Figure 9 shows that ProfitPilot compares to the other heuristics regarding rebalancing fees, except for the Lightning Network.

Impact on the network. Finally, we apply ProfitPilot to multiple existing nodes in the network using the procedure described in Section III. Basically, ProfitPilot connects to the node that provides the highest incentive among the neighbors of the existing node's neighbor. Our goal is to verify the impact of ProfitPilot on network transitivity and robustness. In particular, we randomly select 15 nodes with a single neighbor and use ProfitPilot to create 3 bidirectional channels for each node. Figure 10a shows the impact of ProfitPilot and the `lib_autopilot` heuristics on the network transitivity as



(a) Barabasi-Albert topology. (b) Watts-Strogatz topology. (c) Lightning Network topology.

Fig. 9: Rebalancing fees in satoshis using ProfitPilot and state-of-the-art heuristics in the evaluated topologies.



(a) Improvement in network transitivity as a function of the number of nodes adopting each heuristic in the Lightning Network topology.

(b) Robustness against targeted attacks in the Lightning Network topology. The curves illustrate the number of nodes adopting ProfitPilot.

Fig. 10: Impact of our proposal on the Lightning Network topology as more nodes adopt ProfitPilot.

more nodes adopt these designs. ProfitPilot offers the fastest transitivity increase among all of the strategies. This increase is mostly due to ProfitPilot creating connections in previously intransitive triads in a process similar to closing triangles. While other heuristics also create triangles, they also create more intransitive triads, slowing the increase in network transitivity. We also note that as more nodes adopt ProfitPilot, the quicker the network transitivity reverses its downtrend [18], [28].

We also examine the robustness of the network against attacks on central nodes. This type of attack is critical in the current Lightning topology as discussed in Section II. Figure 10b demonstrates that ProfitPilot efficiently mitigates the effect of targeted attacks by 18% with only 45 nodes adopting it with 3 channels each. We also verify that the higher the number of nodes adopting ProfitPilot, the more robust the network becomes. Although our test is restricted to nodes creating 3 channels, we also expect that an increase in the number of channels results in a more robust network. ProfitPilot's features, i.e. profitability and low-cost routing, encourage mass adoption, enhancing the network's robustness even further.

V. Related Work

Payment channel networks. Most of PCN works in the literature focus on improving the standard routing algorithm [8], [30], [34], [29], [41], [33] or proposing rebalancing strategies [8], [16], [14], [15]. The problem of making rebalancing cheaper and faster is addressed by multiple proposals in the literature [8], [13], [42], [14]. Sivaraman *et al.* propose Spider, a PCN routing algorithm that uses a congestion control protocol to keep channels balanced [8]. Spider defines that payment sources must maintain an adjusted flow window to issue transactions at the same rate they receive, keeping the channel balanced according to demand. In Spider, if the flow of payments in one direction is greater than the flow in the opposite direction, the transaction waits in the queue for another transaction of the same amount in the opposite direction to guarantee equal flows in both directions. This solution, despite improving channel balancing, has some major disadvantages. First, Spider relies on the existence of unpredictable payment demands in the opposite direction to equalize the flow in both directions. If the demand does not exist, the user may have her payment halted until the payment timeout. Second, Spider ignores payment fees in its optimization, which can result in high costs for the user. Finally, implementing queues and flow control requires structural changes in Lightning Network, which currently does not have these features.

One of the most popular rebalancing techniques in PCNs is issuing self-payments through circular routes. This method allows users to move funds from a low-demand channel to a high-demand channel, reactivating the channel without resorting to the blockchain. Khalil and Gervais propose REVIVE, a secure payment method on circular routes [13]. In REVIVE, an elected leader receives rebalance requests from users and

calculates a set of transactions that must be performed. This set of transactions seeks to meet user requirements and must ensure that users' funds remain the same during the process. The algorithm, however, violates users' privacy, given that to calculate the set of rebalancing transactions, the leader must know users' channel balance. Awathare *et al.* propose REBAL, a circular rebalancing method that uses the transaction flow history of the channel to define the amount of funds that should be moved between channels [14]. In REBAL, participants run the rebalancing protocol locally, which removes the need to share private information with third parties. Otto presents a tool to automate the rebalancing of channels through self-payments in the implementation of the Lightning Network [42] language. The tool allows users to configure the amount of funds they want in each channel and calculates, based on an optimization problem, the best set of rebalancing transactions locally.

All of these proposals, however, depend on the availability of cycles in the network. As we show in this paper, these operations are not possible for a significant number of nodes in the network. Instead, these nodes have to resort to the blockchain, paying high fees and waiting for long periods of time for transaction confirmation. Our approach guarantees the creation of cycles, enabling off-chain rebalancing for nodes in the network.

Attachment strategies. Several works study different strategies for adding new nodes in PCNs [23], [21], [19]. Pickhardt presents `lib_autopilot`, a software to automate the creation of channels in the network. `Lib_autopilot` allows users to select heuristics for creating channels, such as random, central, and reduced diameter [23]. `Lib_autopilot` has been adopted as a plug-in in the c-lightning implementation of the Lightning Network. Lange *et al.* study models of attachment strategies applied to the context of payment channel networks [19]. The authors verify a trade-off between security and efficiency in the evaluated models. While connecting to higher-degree nodes usually results in more efficient payment routing, they also expose the network to several security threats. Ersoy *et al.* focus on making payment channels profitable for users [21]. The authors formalize the maximum reward problem, show that the problem is NP-hard, and propose a channel creation algorithm that returns the maximum reward for a channel. The authors, however, consider a scenario in which the nodes only seek to act as routers, without issuing payments. This consideration prevents the development of a solution close to the real scenario of payment channel networks, in which participants assume roles of both sellers and buyers [5]. In addition, all the above-cited solutions ignore the issue of rebalancing the channels, which can result in a low lifespan for the channel and high costs in channel operation.

Unlike previous works, we introduce a node attachment strategy that aims at increasing users' financial gains while simultaneously decreasing their possible routing costs. Furthermore, our paper develops an algorithm that allows users to establish cycles in the network topology, enabling rebalancing operations and extending the channel lifespan. The proposed model compensates for the cost of opening multiple channels to establish cycles. As far as we know, our approach is the first one to propose cycle creation to enable channel rebalancing and improve network robustness.

VI. Conclusion

This paper presented ProfitPilot, a strategy for connecting new nodes in payment channel networks. Our proposed model presents financial advantages to users, such as compensating the costs of channel creation by establishing profitable and low-cost routes. Furthermore, the proposed scheme is also beneficial to the network as it promotes redundancies and makes it more robust to some security threats. We expect that these features promote user adoption. We implement and test ProfitPilot on real-world Lightning Network topology and the results show that the presented solution rewards the user up to 3× more than the traditional methods of adding nodes in the network. ProfitPilot also presents cost-efficient routes with as low as one channel being created by the user when compared to state-of-the-art heuristics. ProfitPilot also benefits network security, as it quickly increases network transitivity and reduces the effects of targeted topological attacks.

Future work should offer fee-setting strategies, which allow the entry node to maintain its profitable position after joining the network.

VII. Acknowledgments

This paper was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This paper was also funded by CNPq, CAPES, FAPERJ, and FAPESP (2018/23292-0, 2015/24494-8, 2015/24514-9, 2015/24485-9, and 2014/50937-1).

References

[1] "Visa acceptance for retailers," Access: Oct 31st, 2023. [Online]. Available: https://usa.visa.com/run-your-business/small-business-tools/retail.html

[2] "Bitcoin Scalability," Access: Oct 31st, 2023. [Online]. Available: https://en.bitcoin.it/wiki/Scalability

[3] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 95–112.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 51–68.

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

[6] D. M. F. Mattos, G. R. Carrara, C. Albuquerque, and D. Mossé, "Exploring overlay topology cost-termination tradeoff in blockchain vicinity-based consensus," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1733–1744, 2023.

[7] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, p. 949–966.

[8] 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 *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, 2020, pp. 777–796.

[9] N. Ilk, G. Shang, S. Fan, and J. L. Zhao, "Stability of transaction fees in bitcoin: A supply and demand perspective," *Management Information Systems Quarterly*, vol. 45, no. 2, pp. 563–692, 2021.

[10] "Raiden network," Access: Oct 31st, 2023. [Online]. Available: https://raiden.network/

[11] CloudTweaks, "How Bitcoin Brought The Lightning Network To El Salvador," 2021, Access: Oct 31st, 2023. [Online]. Available: https://cloudtweaks.com/2021/07/how-bitcoin-brought-lightning-network-el-salvador/

[12] C. Decker, "Splicing. [Lightning-dev] Channel top-up," 2017, Access: Oct 31st, 2023. [Online]. Available: https://lists.linuxfoundation.org/pipermail/lightning-dev/2017-May/000696.html

[13] R. Khalil and A. Gervais, "Revive: Rebalancing Off-Blockchain Payment Networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 439–453.

[14] N. Awathare, Suraj, Akash, V. J. Ribeiro, and U. Bellur, "REBAL: Channel Balancing for Payment Channel Networks," in *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Nov. 2021, pp. 1–8.

[15] Z. Avarikioti, K. Pietrzak, I. Salem, S. Schmid, S. Tiwari, and M. Yeo, "Hide&seek: Privacy-preserving rebalancing on payment channel networks," in *Financial Cryptography and Data Security*, 2022, p. 358–373.

[16] Z. Hong, S. Guo, R. Zhang, P. Li, Y. Zhan, and W. Chen, "Cycle: Sustainable off-chain payment channel network with asynchronous rebalancing," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 41–53.

[17] R. Pickhardt and M. Nowostawski, "Imbalance measure and proactive channel rebalancing algorithm for the lightning network," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–5.

[18] I. A. Seres, L. Gulyás, D. A. Nagy, and P. Burcsi, "Topological analysis of bitcoin's lightning network," in *Mathematical Research for Blockchain Economy*, 2020, pp. 1–12.

[19] K. Lange, E. Rohrer, and F. Tschorsch, "On the impact of attachment strategies for payment channel networks," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–9.

[20] E. Rohrer, J. Malliaris, and F. Tschorsch, "Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2019, pp. 347–356.

[21] O. Ersoy, S. Roos, and Z. Erkin, "How to Profit from Payments Channels," in *Financial Cryptography and Data Security*. Springer International Publishing, 2020, pp. 284–303.

[22] G. Avarikioti, Y. Wang, and R. Wattenhofer, "Algorithmic Channel Design," p. 12 pages, 2018, Access: Oct 31st, 2023. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9964/

[23] R. Pickhardt, "lightning-network-autopilot," 2019, Access: Oct 31st, 2023. [Online]. Available: https://github.com/renepickhardt/lightning-network-autopilot

[24] LND, "lnd-autopilot," 2022, Access: Oct 31st, 2023. [Online]. Available: https://github.com/lightningnetwork/lnd/tree/master/autopilot

[25] C. Decker, "Lightning network research; topology datasets," https://github.com/lnresearch/topology, 2021, Access: Oct 31st, 2023.

[26] R. Russell *et al.*, "BOLT #7: P2p node and channel discovery," https://github.com/lightningnetwork/lightning-rfc/blob/master/https://github.com/lightning/bolts/blob/master/07-routing-gossip.md, 2022.

[27] "1ML - Lightning Network Search and Analysis Engine," Access: Oct 31st, 2023. [Online]. Available: https://1ml.com/

[28] G. F. Camilo, G. A. F. Rebello, L. A. C. de Souza, M. Potop-Butucaru, M. D. Amorim, M. E. M. Campista, and L. H. M. K. Costa, "Topological evolution analysis of payment channels in the lightning network," in *2022 IEEE Latin-American Conference on Communications (LATINCOM)*, 2022, pp. 1–6.

[29] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions," in *Network and Distributed System Security Symposium*, 2018.

[30] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19. Association for Computing Machinery, 2019, p. 370–381.

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

[32] "Lightning Network DDoS Sends 20% of Nodes Down," https://www.trustnodes.com/2018/03/21/lightning-network-ddos-sends-20-nodes, Access: Oct 31st, 2023.

[33] Y. Zhang and D. Yang, "RobustPay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1676–1686, 2021.

[34] W. Chen, X. Qiu, Z. Hong, Z. Zheng, H.-N. Dai, and J. Zhang, "Proactive look-ahead control of transaction flows for high-throughput payment channel network," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, p. 429–444.

[35] Z. Avarikioti, L. Heimbach, Y. Wang, and R. Wattenhofer, "Ride the Lightning: The Game Theory of Payment Channels," in *Financial Cryptography and Data Security*. Cham: Springer International Publishing, 2020, pp. 264–283.

[36] H. Huang, J. Tang, L. Liu, J. Luo, and X. Fu, "Triadic closure pattern analysis and prediction in social networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 12, pp. 3374–3389, 2015.

[37] P. Hu and W. C. Lau, "A survey and taxonomy of graph sampling," *arXiv preprint arXiv:1308.5865*, 2013.

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

[39] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998, number: 6684 Publisher: Nature Publishing Group.

[40] P. Erdős, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[41] Y. Chen, Y. Ran, J. Zhou, J. Zhang, and X. Gong, "MPCN-RP: A routing protocol for blockchain-based multi-charge payment channel networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1229–1242, 2022.

[42] C. Otto, "Rebalance-LND," 2022, Access: Oct 31st, 2023. [Online]. Available: https://github.com/C-Otto/rebalance-lnd

Appendix A
Problem Formulation

We formulate the following problem. Consider a payment channel network described by a directed graph $G = (V, E)$ and a user u interested in creating new channels in a PCN. The user u can create channel a e_{uv} with any $v \in V$. The fee is composed of a fixed base fee f_{uv}^B and a proportional fee $f_{uv}^P(p_v)$ proportional to the value of payment p_v. These fees combine into a total fee $f_{uv}^T(p_v) = f_{uv}^B + f_{uv}^P(p_v)$ charged to route p_v on the channel e_{uv}. Like Ersoy *et al.* [21], we define an event of a transaction with source s and destination t and value p_v passing through the established channel as $X(p_v, s, t)$. We model the expected gain of the user u in the newly created channel e_{uv} as a product of probabilities, given by [21]:

$$\mathsf{E}[G_u] = \tilde{\mathrm{O}}_{\substack{\forall s,t \in V \\ s<t<u}} \; \tilde{\mathrm{O}}_{p_E=1} \; \tilde{\mathrm{O}}_{v \in C} Pr[P = (s,t)] \times Pr[T = p_v] \times f_{ut}(p_v) Pr[X(p_v, s, t)], \quad (2)$$

where $Pr[P = (s,t)]$ is the probability of a payment occurring between nodes s and t, modeled by a distribution P; $Pr[T = p_v]$ is the probability of a payment, modeled by the statistical variable T, which can assume a maximum value of T_{max}, having a value p_v. Finally, $Pr[X(p_v, s, t)]$ is the probability of the payment of value p_v going from s to t passing through channel e_{uv}, given $C = V - \{u\}$. The gain is given by the product of all three probabilities.

Besides maximizing the financial gain, described by Equation 2, the user u also expects to pay fewer fees when issuing payments. Unlike previous work [21], [22] that solely focuses on collecting fees, we consider that the new user might want to issue payments in the network. As we assume that u does not know who he/she is going to transact with before joining the network, we aim to reduce u's distance to every other node in the network.

$$\mathsf{E}[C_u] = \sum_{\substack{\forall t \in V \\ t<u}} \sum_{p_E=1}^{T_G} \sum_{\substack{\forall t \in V \\ t<u}} Pr[P = (u,t)] \times Pr[T = p_v] \times f_{ut}^{T}, \tag{3}$$

where $f_{ut}^{T}(p_v)$ is the fee charged by the channel to forward a payment of value p_v from u to t.

To simplify the problem, we assume the probability of payment from s to t in Equation 2 to be modeled as a uniform distribution. Thus, $Pr[P] = \frac{1}{(|V|-1)(|V|-2)}$ is constant, and $|V|$ is the number of vertices in the network. We also consider a fixed p_v value, eliminating the second sum and the fee charged $f_{uv}^{T}(p_v)$. Equation 2 can, then, be rewritten as $\mathsf{E}[G_u] = \sum_{v \in C} Pr[X(p_v, s, t)]$. Finally, we adopt the betweenness centrality to model the probability $Pr[X(p_v, s, t)]$ of the event X. As PCNs use Dijkstra's SPF, we consider that the betweenness centrality of a node provides an accurate prediction on the probability of that node forwarding payments.

Definition A.1. ***Node betweenness centrality.*** *A node's u betweenness centrality is proportional to the number of shortest paths that traverses u and is defined as*

$$bc_u = \sum_{\substack{s<t, \\ \sigma_{BC}<0}} \frac{\sigma_{st[u]}}{\sigma_{st}},$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st[e_{DE}]}$ is the number of shortest paths between s and t that goes through u.

Thus, we rewrite Equation 2 as $\mathsf{E}[G_u] = bc_u$. The same logic is applied to Equation 3, associating it with closeness centrality.

Definition A.2. ***Closeness centrality.*** *The closeness centrality of a node u is inversely proportional to its distance to other nodes and is defined as*

$$cc_u = \sum_{v \in V} \frac{1}{d_{uv}},$$

where d_{uv} is the distance between u and v in total fees.

As we set the distance of a node to another as the total fee charged by the channel, we can use closeness centrality to infer the average amount of fees paid by our node. Thus, the sum of distances acts as a natural average for the fees encountered by u assuming a uniform distribution of transaction values [35]. We rewrite Equation 3 as $1/\mathsf{E}[C_u] = cc_u$, where cc_u is the closeness centrality of u. As closeness centrality is inversely proportional to the sum of distances, the higher a node's closeness centrality the smaller the distance between the node and the rest of the network. Therefore, instead of minimizing costs, we combine both closeness and betweenness centrality into a metric that we aim to maximize. We introduce the expected incentive $\mathsf{E}[I(u)]$ of node u as

$$\mathsf{E}[I(u)] = bc_u + cc_u. \tag{4}$$

Appendix B
Creating Triangles with ProfitPilot

As mentioned in Section III-C, ProfitPilot guarantees the creation of $|\mathcal{N}|-1$ triangles. We refrain from a more formal proof that would require showing a loop invariant proof of the inner loop in Algorithm 1 to demonstrate that it only creates channels with already selected nodes' neighbors. Instead, we intuitively state that the inner loop traverses the nodes of the network and only selects nodes that are neighbors of already selected ones. This can be logically observed by the '*if*' condition inside the inner loop that checks if the analyzed node is a neighbor of a node before storing it as a potential node as a suggestion.

Theorem B.1 (ProfitPilot's triangles.)**.** *At each interaction of Algorithm 1, the ingress node n participates in at least $|\mathcal{N}|-1$ triangles, for $|\mathcal{N}| \geq 1$.*

Proof. We prove Theorem B.1 using the loop invariant method.
Initialization: We first show that the loop invariant holds for $|\mathcal{N}| = 1$. In this scenario, the list of suggested node connections $\mathcal{N}$ is composed of a single element, meaning that n has only one channel. Thus, the loop invariant holds, given that one channel is not enough to form a triangle, i.e., n participates in $|\mathcal{N}|-1 = 0$ triangles.
Maintenance: Next, we show that each iteration maintains the invariant. First, assume that the loop invariant holds for an iteration j, that is $|\mathcal{N}| = j$ and node n participates in $j-1$ triangles. The inner loop only selects nodes that share a channel with a node in $\mathcal{N}$. Thus, if a node is selected by the algorithm, it will be added to $\mathcal{N}$ and $|\mathcal{N}| = j+1$. The algorithm will, then, establish a channel with this selected node and, as it shares a channel with a node already in $\mathcal{N}$, the node will participate in at least one more triangle, i.e., at least j triangles. If the algorithm traverses the network without selecting a node, $|\mathcal{N}|$ will remain unchanged, maintaining the number of triangles at the beginning of the interaction. Hence, the loop invariant holds at the end of the iteration.
Termination: Finally, we analyze the loop termination. There are two possible ways that the algorithm terminates. First, if the maximum number of channels that the user wants to create k has been achieved, meaning $|\mathcal{N}| = k$. In that case, node n participates in at least $k-1$ triangles. The second possibility of terminating the algorithm occurs if ProfitPilot fails to find a higher incentive than the current setting. At that point, the algorithm breaks out of the loop in the last **if** condition presented at Algorithm 1 and the node participates in at least $|\mathcal{N}|-1$ triangles. □